Tell us about your PDF experience.

What's new in the Azure Well-Architected Framework

06/04/2025

Find out about recent changes in the Azure Well-Architected Framework.

May 2025

Updated articles

- Recommendations for designing a reliable scaling strategy: Explore updated content including: Choosing the right technology for scaling; Automating scaling operations, including use of Infrastructure-as-Code; Selecting and optimizing "scale units"; Scaling data stores using sharding and partitioning, and optimizing partition strategies; Monitoring scaling operations and log analysis.
- Design a data partitioning strategy: This article can now be found under "Design guides."
- Cost Optimization design principles: Find actionable recommendations, such as treating different environments differently, using dynamic scaling, and collaborating with licensing teams. We added guidance on governance and cost guardrails and expanded examples and practical steps for budgeting, rate optimization, and maximizing resource utilization.
- Operational Excellence cloud design principles: We refreshed this article, consolidating guidance and updating safe deployment practices.

Service guides

This month, we made significant updates to some of our service guides. Here are the highlights:

- Architecture Best Practices for Azure Database for PostgreSQL: Find recommendations for features like high availability, geo-redundant backup, private networking, Microsoft Entra ID integration, cost modeling, automation, monitoring, and intelligent performance tuning.
- Architecture Best Practices for Azure Cosmos DB for NoSQL: Explore guidance updated to reflect current Azure Cosmos DB best practices, including new features, policy links, and recommendations for using Microsoft Entra ID, private endpoints, Azure Policy, and Azure Advisor.

April 2025

New articles

- Architecture Best Practices for Azure Container Apps: Explore key recommendations and design checklists for implementing Azure Container apps effectively and securely. This guide covers design principles, strategies, and recommendations for achieving architectural goals, including security, performance, and cost optimization.

Updated articles

- Application Delivery Considerations for Azure Virtual Desktop Workloads: We made significant updates to refactor and refresh the guidance about Azure Virtual Desktop including updated recommendations and best practices.
- Recommendations for designing a disaster recovery strategy: We added a new section on Azure Backup facilitation and updated the content to include new Azure Backup features.

Service guides

This month, we made significant updates to some of our service guides. Here are the highlights:

- Architecture Best Practices for Azure API Management: We refactored the guidance for Azure API Management to improve clarity and usability. The updated content includes refreshed design principles, strategies, and recommendations for achieving architectural goals.
- Architecture Best Practices for Azure NetApp Files: We added new recommendations for cost optimization and operational excellence. Explore the fully updated content including new recommendations for configuring Azure NetApp Files to protect your workloads.
- Architecture Best Practices for Azure SQL Database: We refreshed and expanded this guide to introduce new design considerations and recommendations, including guidance on using Azure SQL Database for secure data storage, managing database performance, and optimizing costs.
- Architecture Best Practices for Azure Virtual Machines and Scale Sets: We made updates to the guidance about Azure Virtual Machines and Scale Sets. Explore the fully updated content including new recommendations for configuring automatic recovery options to protect your workloads.

March 2025

Updated articles

- Architecture Best Practices for Azure Front Door: We added details on deployment strategies and the importance of caching static content to the design checklist. Recommendations were updated to include new links and advice on managing traffic, health probes, and optimizing caching.
- Design review checklist for Operational Excellence: We simplified the design review checklist for Operational Excellence to make the recommendations more concise, focused, and actionable.

February 2025

Updated articles

- Recommendations for self-healing and self-preservation: We updated this guide for easier readability by streamlining several sections. We also moved the guides for handling transient faults and developing background jobs into the design guide area to make the Reliability pillar easier to use.
- Design review checklist for Reliability: We simplified the design review checklist for Reliability to make the recommendations more concise, focused, and actionable.

Service guides

This month, we made significant updates to some of our service guides. Here are the highlights:

- Azure Front Door: We improved content clarity by rephrasing various sections and adding more detailed explanations. We updated links and references to other relevant Azure documentation and removed redundant or outdated information. We aligned content with the latest Azure Front Door features and best practices.
- Azure IoT Hub: We refreshed and expanded this guide to cover all Well-Architected pillars. We also introduced new design considerations and recommendations, including guidance on using Azure IoT Hub for secure device-to-cloud communication, managing device identities, and optimizing costs.

January 2025

Updated articles

- How to use the Azure Well-Architected Framework documentation: We added a new section on adopting a phased learning process, emphasizing the importance of improving

workload quality iteratively.

Service guides

This month, we made significant updates to many of our service guides. Here are some highlights:

- Azure App Service (Web Apps): Added and updated several design considerations and recommendations, including guidance on redundancy, taking advantage of the App Service auto-heal feature, and leveraging the App Service Resiliency Score Report.
- Azure Functions: We refreshed and expanded this guide to cover all Well-Architected pillars. We also introduced new design considerations and recommendations, including guidance on the use of managed identities, network security controls, cost monitoring, CI/CD pipelines, and autoscaling.
- Azure Kubernetes Service (AKS): We refreshed this guide to improve the structure and presentation of the content, and to reflect the latest best practices for configuring Azure Kubernetes Service (AKS) to protect your workloads and data, optimize costs, and improve operational efficiency.
- Azure Load Balancer: We refreshed and expanded this guide to cover all Well-Architected pillars. We also introduced new design considerations and recommendations, including guidance on optimizing your load balancing configurations, using the Global tier to load balance across Azure regions, and how to improve operational efficiency.
- Azure Service Fabric: We refreshed this guide to improve the structure and presentation of the content and introduced new guidance to help you optimize your Service Fabric workloads.
- Azure Traffic Manager: We refreshed and expanded this guide to cover all Well-Architected pillars. We also introduced new design considerations and recommendations, including guidance on load balancing across regions, enhancing DNS security, and using diagnostic logs and traffic view dashboards to optimize and troubleshoot Traffic Manager profiles.
- Azure Virtual Network: We refreshed and expanded this guide to cover all Well-Architected pillars. We also introduced new design considerations and recommendations, including guidance on adding redundancy, segmenting networks for security, and monitoring traffic patterns. Use infrastructure as code in your deployments, optimize network traffic, and leverage Azure Network Watcher for performance insights.

December 2024

New articles

- Azure Well-Architected Framework Perspective on Azure Disk Storage: Find new design considerations and configuration recommendations for optimizing Azure Disk Storage within the Azure Well-Architected Framework. Explore guidance on key areas such as reliability, security, cost optimization, operational excellence, and performance efficiency, as well as strategies and best practices to enhance storage management for Azure Virtual Machines.

Azure IoT Hub workload guidance retirement

- This month, we announced the deprecation of Azure IoT Hub workload documentation in the repo. The content was outdated and no longer aligned to the Azure Well-Architected Framework.

November 2024

New articles

- Well-Architected Framework Perspective on Azure Monitor Application Insights: Explore design considerations and configuration recommendations for Azure Monitor Application Insights. Azure Monitor Application Insights is an extensible Application Performance Management (APM) service that helps you monitor the performance and usage of your live web applications. It provides real-time insights into your application's performance and user behavior, enabling you to detect and diagnose issues and understand what users actually do with your app.

New workload: AI on Azure

This month, we introduced new guidance for designing AI workloads on Azure. This documentation is appropriate for roles that are accountable for designing, building, and maintaining a solution for running AI workloads in a cloud environment. Use the AI Workloads on Azure documentation as your go-to resource to build and optimize AI solutions on Azure.

- AI Workloads on Azure
- Design Methodology for AI Workloads on Azure
- Design Principles for AI Workloads on Azure
- Application Design for AI Workloads on Azure
- Application Platform for AI Workloads on Azure
- Design Training Data for AI Workloads on Azure
- Grounding Data Design for AI Workloads on Azure
- Data Platform for AI Workloads on Azure

- MLOps and GenAIOps for AI Workloads on Azure
- AI Workload Operations on Azure
- Test and Evaluate AI Workloads on Azure
- Responsible AI in Azure Workloads
- Workload Team Personas Involved in AI Workloads
- AI Workload Assessment

New workload: Software as a service (SaaS) on Azure

This month, we added a new workload for SaaS on Azure. This documentation provides actionable and authoritative guidance that applies Well-Architected best practices as the technical foundation for building and operating a SaaS solution on Azure at-scale. Use the SaaS Workloads on Azure documentation to build scalable, performant, reliable, and secure SaaS solutions.

- SaaS Workloads
- Design Methodology for SaaS Workloads on Azure
- Design Principles of SaaS Workloads on Azure
- Billing and Cost Management for SaaS Workloads on Azure
- Governance for SaaS Workloads on Azure
- Resource Organization for SaaS Workloads on Azure
- Identity and Access Management for SaaS Workloads on Azure
- Compute for SaaS Workloads on Azure
- Networking for SaaS Workloads on Azure
- Data for SaaS Workloads on Azure
- DevOps Practices for SaaS Workloads on Azure
- Incident Management for SaaS Workloads on Azure
- Assessment Review Tool for SaaS Workloads on Azure

October 2024

Updated articles

- Architecture decision record (ADR): We refreshed the guidance on what an ADR should include, including consistent elements like problem statements, options considered, and decision outcomes. Explore updates including a new section on suggested characteristics of an individual record with guidelines for maintaining consistent and useful ADRs

- Azure Well-Architected Framework perspective on Azure Application Gateway v2: We made significant updates to the guidance about Azure Application Gateway v2. Find

important notes and links to additional resources for Azure Application Gateway configurations. Explore enhanced content with specific design principles, strategies, and recommendations for achieving architectural goals.

- Azure Well-Architected Framework perspective on Azure ExpressRoute: We made significant updates to provide more comprehensive and structured guidance on Azure ExpressRoute, enhancing and expanding upon best practices, design principles, and optimization strategies. The detailed checklists and recommendations help in better planning and implementation, ensuring improved reliability, security, and cost efficiency.

We reviewed all tradeoff and design pattern articles for alignment with the content structure and to ensure that the guidance is up to date. Tradeoffs are an essential part of the Well-Architected Framework, as they help you understand the implications of design decisions on other pillars. Design patterns are reusable solutions to common problems that you might encounter when designing a workload. They help you understand how to design your workload to meet the goals of the Well-Architected Framework. Check out the updated articles:

- Cloud design patterns that support cost optimization
- Cost Optimization tradeoffs
- Cloud design patterns that support reliability
- Reliability tradeoffs
- Cloud design patterns that support security
- Security tradeoffs
- Cloud design patterns that support operational excellence
- Operational Excellence tradeoffs
- Cloud design patterns that support performance efficiency
- Performance Efficiency tradeoffs

September 2024

New articles

- Azure Well-Architected Framework perspective on Azure NetApp Files: Explore design considerations and configuration recommendations for Azure NetApp Files. Azure NetApp Files is a fully managed file share service that supports the Server Message Block (SMB) protocol and Network File System (NFS) protocol. Learn how you can use Azure NetApp Files for file sharing, high-performance computing, home directories, and databases.

Updated articles

- Azure Well-Architected Framework perspective on Azure Firewall: We made significant updates to the guidance about Azure Firewall. Explore the fully updated content including new recommendations for configuring Azure Firewall to protect your workloads.

August 2024

Updated articles

- Recommendations for defining reliability targets: We made significant updates, including new guidance on composite SLOs. The updated content links to guidance about how to focus on realistic expectations and build a health model to define system states.

We added guidance about ensuring that the VMs in your backup environment run on OSs that have supportability:

- Operations considerations for Azure VMware Solution workloads
- Optimize business continuity and disaster recovery

Hybrid retirement

- This month, we announced the deprecation of Hybrid documentation in the repo. The content was outdated and no longer aligned to the Azure Well-Architected Framework.

July 2024

New articles

- Azure Well-Architected Framework perspective on Azure Local: Explore design considerations and configuration recommendations for Azure Local. Azure Local is a hyperconverged infrastructure (HCI) solution that hosts virtualized Windows and Linux workloads and their storage in a hybrid on-premises environment. Learn how you can use Azure Local and Azure Arc capabilities to keep business systems and application data on-premises to address data sovereignty, regulation and compliance, and latency requirements.

Updated articles

- Workload architecture design specification: Learn about best practices for creating functional and technical specifications and the importance of consistency in your workload design documentation.

- Architecture design diagrams: We updated the guidance to include Microsoft Entra ID architecture icons.

Find updated guidance on using flexible virtual machine scale sets instead of availability sets for deployment across multiple zones:

- Application platform considerations for mission-critical workloads on Azure
- Application delivery considerations for Azure Virtual Desktop workloads

Carrier Grade retirement

- This month, we announced the deprecation of the Carrier Grade documentation. The content was outdated and no longer relevant to the Azure Well-Architected Framework.

June 2024

Updated articles

- Recommendations for defining reliability targets: Find updated SLO and SLA guidance to help you set and measure reliability targets for your workload.
- Design methodology for sustainability workloads on Azure: Learn how Azure carbon optimization can help you reduce the carbon footprint of your workload by providing granular emissions data.

What is the Azure Well-Architected Framework?

Article • 01/10/2025

The Azure Well-Architected Framework is a design framework that can improve the quality of a workload by helping it to:

- Be resilient, available, and recoverable.
- Be as secure as you need it to be.
- Deliver a sufficient return on investment.
- Support responsible development and operations.
- Accomplish its purpose within acceptable timeframes.

The framework is founded on the five pillars of architectural excellence, which are mapped to those goals. They are: **Reliability**, **Security**, **Cost Optimization**, **Operational Excellence**, and **Performance Efficiency**.

Each pillar provides recommended practices, risk considerations, and tradeoffs. The design decisions must be balanced across all pillars, given the business requirements. The technical and actionable guidance is broad enough for all workloads and applies to a specific scenario. This guidance is centered on Azure.

Workload architecture isn't the same as its implementation. The Well-Architected Framework can set you up for success through architectural design, but the implementation choices depend on the business requirements and constraints of your organization.

Audience

The Well-Architected Framework applies to teams that are responsible for improving workloads and addressing cross-cutting concerns.

The Well-Architected Framework provides valuable insights and recommendations for anyone involved in the lifecycle of a workload. Regardless of your role in a workload team, whether architect, developer, operator, or business stakeholder, if you have the authority to make decisions within the scope of a workload, you can benefit from this framework.

This guidance is beneficial regardless of the scale of your organization. Whether you're part of a large enterprise, a small business, or an independent software vendor, you can move a step closer to optimal design. The framework caters to a wide range of

organizational structures and sizes, ensuring that all workload users can effectively use its benefits.

If you're seeking guidance for improving a portfolio of workloads through centralized controls, this content might not fully apply. We recommend that you refer to Cloud Adoption Framework. If you have no vested interest in designing workloads on Azure, this content isn't relevant for you.

For information about the role and duties of an architect, see Architect's fundamentals and Architect's checklist.

Goals

The primary objective of the Well-Architected Framework is to set you up for success when you deploy your workload on Azure.

- **Successful implementation**: A well-architected design leads to successful implementation. Given the breadth and depth of coverage in concepts, you're well-equipped to make informed decisions.
- **Confidence in success**: Proven assessments, seen on numerous workloads deployed on Azure, back the tenets of the framework.
- **Understand tradeoffs and risks**: The framework helps you understand that adopting the recommendations might require making choices against the other pillars. It highlights the tradeoffs and also the potential risks that you might want to address in the short term.
- **Optimize over time**: The framework is designed for iterative use and as a tool for continuous improvement. Measure the maturity of your workload against the guidance. Treat that evaluation as a moving score that evolves with your workload, ensuring that the design remains efficient and effective in meeting your business objectives.

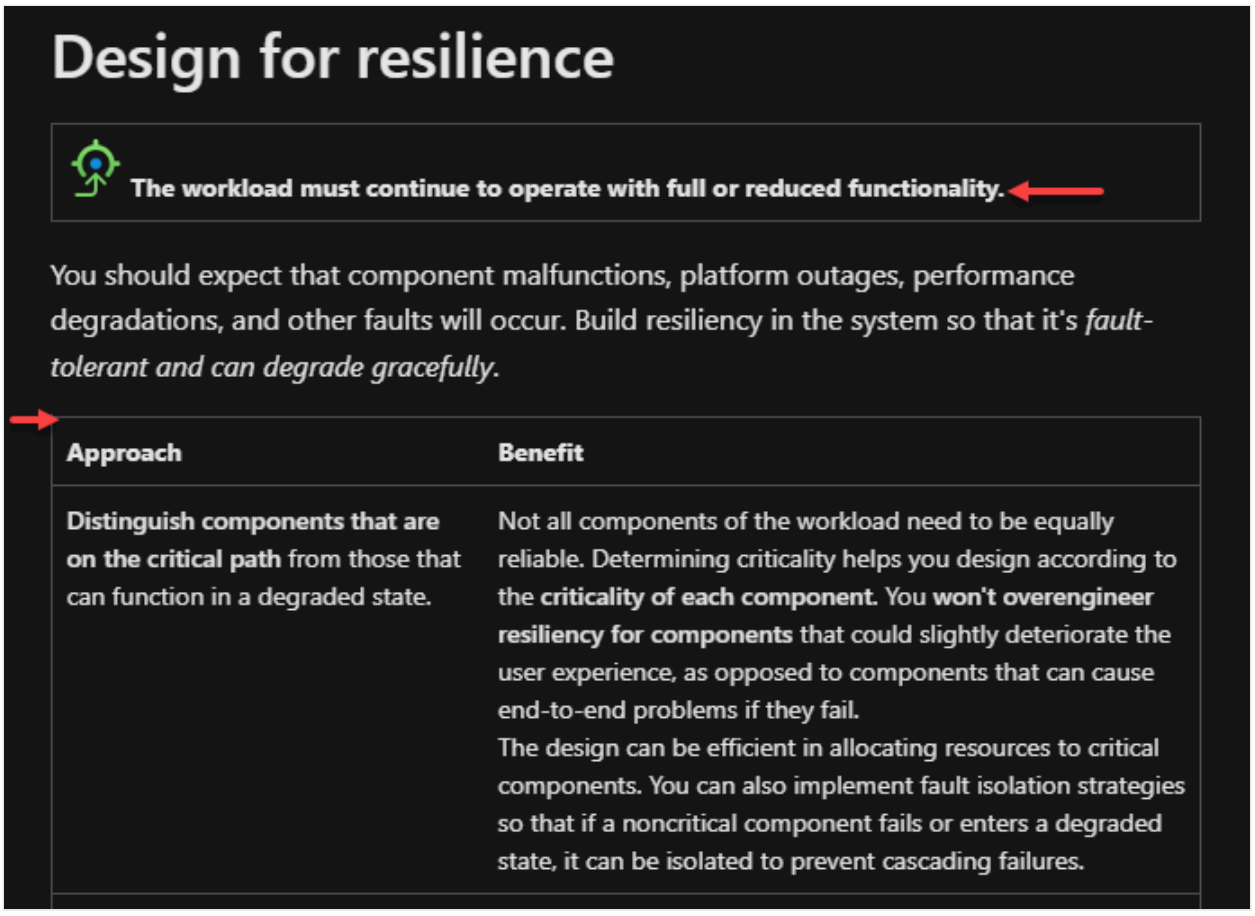
Layers of the Well-Architected Framework

The Well-Architected Framework is structured in a layered approach: pillars, workload, and service guides.

Pillars

The foundation of this framework lies in the pillars. If you don't have a comprehensive understanding of these pillars, the subsequent layers—the workload layer and service guides—might not be fully comprehensible.

At the pillar level, start your journey with the **Design principles**, each of which has a specific goal. Within each principle, follow the approaches to craft your design strategy. These approaches aren't optional and must be taken into account.



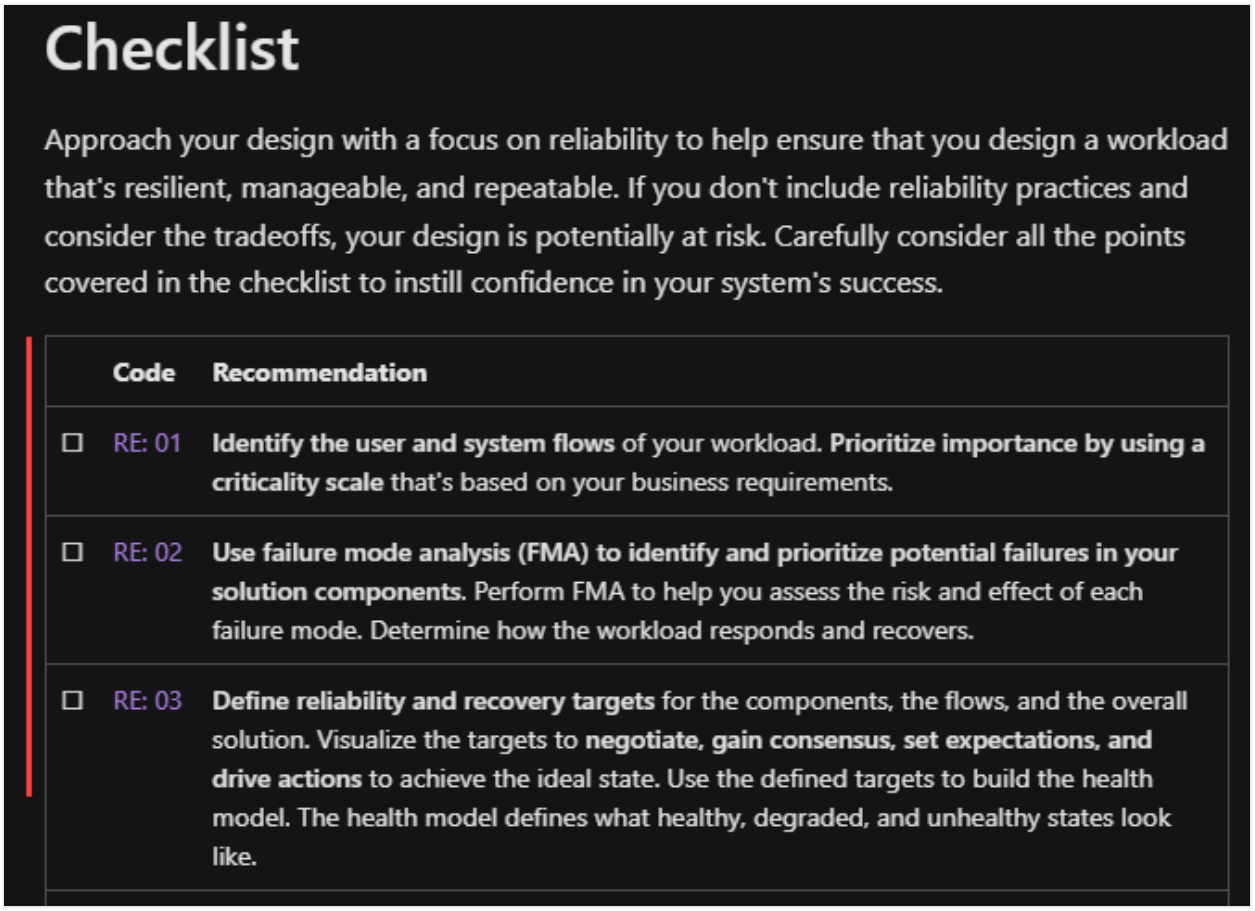
Design for resilience

The workload must continue to operate with full or reduced functionality.

You should expect that component malfunctions, platform outages, performance degradations, and other faults will occur. Build resiliency in the system so that it's *fault-tolerant and can degrade gracefully*.

Approach	Benefit
Distinguish components that are on the critical path from those that can function in a degraded state.	Not all components of the workload need to be equally reliable. Determining criticality helps you design according to the **criticality of each component**. You **won't overengineer resiliency for components** that could slightly deteriorate the user experience, as opposed to components that can cause end-to-end problems if they fail. The design can be efficient in allocating resources to critical components. You can also implement fault isolation strategies so that if a noncritical component fails or enters a degraded state, it can be isolated to prevent cascading failures.

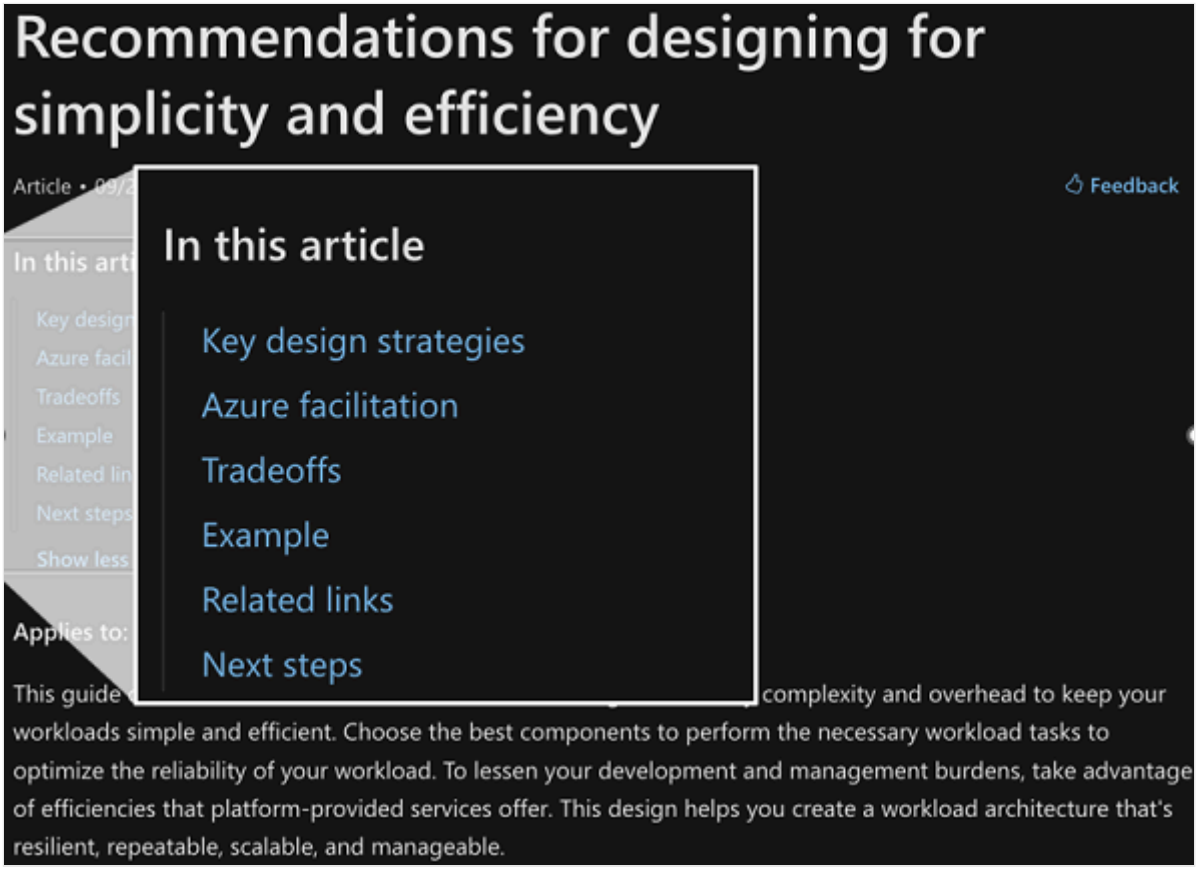
Next, move on to the **Checklist**, which is always your starting point for evaluation. Each item on the checklist is accompanied by one or more **Recommendation guides** that describe key strategies and how Azure helps you attain the recommendation.



Checklist

Approach your design with a focus on reliability to help ensure that you design a workload that's resilient, manageable, and repeatable. If you don't include reliability practices and consider the tradeoffs, your design is potentially at risk. Carefully consider all the points covered in the checklist to instill confidence in your system's success.

	Code	Recommendation
☐	RE: 01	**Identify the user and system flows** of your workload. **Prioritize importance by using a criticality scale** that's based on your business requirements.
☐	RE: 02	**Use failure mode analysis (FMA) to identify and prioritize potential failures in your solution components.** Perform FMA to help you assess the risk and effect of each failure mode. Determine how the workload responds and recovers.
☐	RE: 03	**Define reliability and recovery targets** for the components, the flows, and the overall solution. Visualize the targets to **negotiate, gain consensus, set expectations, and drive actions** to achieve the ideal state. Use the defined targets to build the health model. The health model defines what healthy, degraded, and unhealthy states look like.



Be sure to understand the related **cloud design patterns**. They're mapped to the pillars they directly support.

Each architectural decision entails a series of considerations. These **tradeoffs** represent recognized and accepted compromises that balance the various aspects of the

framework. Tradeoffs are noted with this icon  and risks are noted with this icon  .

For more information, see About the Well-Architected Framework pillars.

Workload

The workload layer represents how the pillars apply to a specific class of workload. During the initial design phase, workload architecture is segmented based on utility, and each segment represents the prioritized or design areas. These design areas are specific to the workload class and serve as focal points for optimization. The Well-Architected Framework includes several workloads. Read the one that closely matches your business requirements. You do not need to read workload guidance for classes of workloads that are not aligned with your scenario.

Begin with **Get started** to understand the solution context. As a refresher, read the **Design principles** to understand how the workload adopts the pillar guidance. Then, dive deep into **Design areas** that focus on the technical decision points with recommendations that follow. Workload guidance also includes an assessment that helps you evaluate your readiness in production.

For more information, see About the Well-Architected Framework workloads.

Service guides

Service guides are instrumental in decision-making that's related to the individual Azure components of a workload. They offer the core features and capabilities of each service that are necessary to attain architectural excellence. It's important to note that these guides aren't configuration guides. Also, they aren't a compiled list of all features and capabilities. The intent is to highlight the utility of the features through Well-Architected pillar perspectives.

For more information, see the available guides.

Assessment

Microsoft Azure Well-Architected Review is offered at no charge. It's a collection of questionnaires tied to the pillar checklists to evaluate your design choices. Track your score through iterative runs to identify possible areas for enhancement.

For more information, see Azure Well-Architected Review tool.

Adopt a phased learning process and improve quality iteratively

The Well-Architected Framework comprehensively covers best practices that are applicable to any class of workload. This guidance includes not only the foundational principles of good design and trade-offs but also the application of those principles to the components of your architecture, as described in the service guides.

We acknowledge that reading this guidance end-to-end can be overwhelming. To achieve the desired state of architectural excellence, we recommend that you have a solid understanding of the design principles for all pillars and that you prioritize checklist items based on their relevance to your workload and business goals. Prioritization should consider factors such as business criticality, compliance needs, and time to market. As these factors evolve, iteratively incorporate more or less from the framework to enhance the quality of your workload.

Consider adopting a phased approach to consuming this guidance. Classify recommendations by what's easy to achieve or must be achieved initially. Then, as your workload's business requirements change, incrementally evolve production-ready systems. For example, initial alignment with the framework could apply to workloads early in their funding and development process, setting a solid foundation for good design. Mature phases of alignment could apply to production-worthy solutions, with the highest levels reserved for always-on, business-critical solutions.

Related links

Here are some resources to get started with using the Well-Architected Framework documentation:

- Azure Well-Architected Framework
- Introducing the Azure Well-Architected Framework
- Training for the Well-Architected Framework
- Azure Well-Architected

Feedback

Was this page helpful? Yes No

Solution architect's fundamentals

Article • 11/01/2024

Every workload passes through a component and topology design process. This process is most intense at the inception of the workload, which includes designing for initial requirements and long-term success of the workload. Architecture is also designed when the workload changes over time and the organization adds, changes, or removes functionality.

Component and topology design is the primary function of an architect. Architects who focus on cloud-based and hybrid solutions are often called *cloud solution architects*. In some organizations, cloud solution architects exist in a centralized capacity within an enterprise architecture group. They can also focus on a specific workload.

A dedicated role can deliver the function of an architect. In some cases, trusted technical specialists (such as a workload engineering lead) can deliver the function of an architect. Or an organization might distribute the function among a small group of senior engineers who are associated with the workload.

Architects usually have experience in roles beyond system design. They might have:

- Been developers and operations team members.
- Worked with customer support teams.
- Developed an understanding of how a system is tested for quality assurance and user acceptance.
- Been through disaster recovery drills or incident responses.
- Been exposed to both incremental and large functional changes in workloads.
- Interpreted specifications and user-acceptance criteria.

Although the preceding list isn't exhaustive, those perspectives are an important aspect of what an architect brings to design duties. The Azure Well-Architected Framework assumes that these practices are in place for the most effective use of the guidance.

The following sections highlight the guiding principles that architects should follow to be effective in their function.

Have a decision-making framework

A key aspect of design is using a consistent process to make decisions. An architect should approach both initial and incremental design with rigor.

Identify expected decisions. Use learned experiences to help with decision identification. Log all decisions that you plan to make.

Make informed decisions. Consider limitations, constraints, tradeoffs, effort, reversibility, and risk. Include supporting evidence from proofs of concept, along with technology documentation and guidance.

Document decisions in an architecture decision record (ADR). Document the justification along with each decision.

Follow up on implementation. Communicate and implement all decisions. Learn from the implementation to help guide future decisions. Look for areas where a failure to identify decisions introduced risk.

Know cloud design patterns

Cloud design patterns are a fundamental building block of architecture. Cloud-based architecture and application design are often an exercise in pattern recognition.

Evaluate a workload's functional and nonfunctional requirements to recognize patterns. Look for opportunities to map your design to use cases via standardized patterns.

Be forward-thinking

Designing to achieve current requirements is a must, but it's important for an architect to predict the workload's evolution. Incorporating change in an implemented system is more expensive than changing the design before implementation.

To design a system that will last until its planned end of life, you must design the workload with architectural flexibility in mind. Avoid design cliffs when you can identify them.

Growth model. Predict how the workload's usage will grow or shrink over time.

Compliance changes. Take proactive measures if you expect the workload to be under compliance requirements in the future. This approach can reduce rework when following compliance becomes a requirement.

Regional expansion. Consider future expansion of the workload into multiple regions. A design that's limited to a single region will need to be heavily refactored for multiple-region deployment, and that can be a costly change. There's even more complexity if the workload design needs to accommodate multiple geographies with different

compliance requirements. Make sure that your design factors in any reasonable prediction about regional expansion.

Product roadmaps. In your design, don't include components that are on the path to deprecation. Likewise, be careful when you include features in your design that are currently in a preview state. They might be released, but they might also be canceled. Being ahead of the curve by using preview features can be highly advantageous. Soon after the feature is released, the workload is prepared to go to production. But include preview features in your design only after you do a careful risk analysis. Ship only features that have a tolerated risk profile.

For more information on cloud design patterns, see:

- Cloud design patterns that support reliability
- Cloud design patterns that support security
- Cloud design patterns that support cost optimization
- Cloud design patterns that support operational excellence
- Cloud design patterns that support performance efficiency

Design for supportability

Design workloads with three key support perspectives:

Cloud provider support. The workload should operate within the supported configuration of your cloud provider to avoid disruptions when you're engaging platform support channels.

Operational visibility. The design should provide execution visibility for the workload operations team to prevent confusion during incident response.

Customer support capabilities. The design should meet user needs but also facilitate customer support functions. A design that hinders the support team's ability to investigate or to assist customers is inadequate.

Maintain and enhance your skills

An architect's expertise is often rooted in practical experience. It's important to invest in expanding your skill set to keep up with the evolving cloud ecosystem.

Education. Seek opportunities for training and certification that technology providers offer for architects.

Community participation. Engage with peers through online and local architecture communities.

Exploratory exercises. Participate in organizationally sponsored hackathons or similar events to develop skills in unfamiliar areas.

Collaborate for success

An architect should take advantage of the expertise of the cloud provider or implementation partner. Most providers want your workload to succeed on their platform, and they often provide services such as architecture design review sessions or consultative sessions with their cloud solution architects. Seek opportunities for review and assistance within your vendor relationships.

Be methodical in your design approach

Architecture frameworks support an architect by offering workload perspectives and methodological approaches. The Well-Architected Framework provides a comprehensive workload viewpoint. Architects can combine the Well-Architected Framework with other architecture frameworks, such as The Open Group Architecture Framework (TOGAF).

Use the principles, checklists, assessments, and reference materials in architecture frameworks to establish a process that fits the workload. Combine frameworks with personal techniques, such as mind mapping.

Architecture is about communication as much as it's about the end product. Make sure that you're optimizing for intentional decision-making, tradeoff acknowledgment, and clear communication in your established processes.

Next steps

Architect's checklist

Feedback

Was this page helpful? Yes No

Solution architect's checklist

Article • 11/01/2024

The responsibility of an architect is to **deliver designs and plans**. Keep in mind that an architect isn't the implementor of a workload. The architect **translates functional and nonfunctional requirements into cloud design patterns** and fit-for-purpose components. The architect also designs a workload that's flexible enough to adapt when needed, but durable enough to weather the planned life of the functionality.

Also included in the design are the operational aspects of the workload, including observability and supportability, and accounting for undesirable situations such as disaster recovery. Finally, the design must be constrained by all business, financial, compliance, and organizational requirements.

Architecture frameworks, such as the Azure Well-Architected Framework, help to give architects a holistic perspective on system design. The Well-Architected Framework artifacts include elements like **design principles**, **checklists**, and **recommendations**. To support the requirements of a workload, these artifacts should be combined with other resources, such as **decision trees**, **reference architectures**, and **assessments**, to make informed decisions.

Checklist

Expand table

	Deliverable tasks
☐	Develop an architecture design specification that's accompanied by diagrams as a structured packet. The specification must meet the workload's functional and nonfunctional requirements and include provisions for routine, ad hoc, and emergency operations.
☐	Create architecture design diagrams that illustrate all aspects of system design, from a broad overview to detailed dimensions such as network and identity.
☐	Maintain an architecture decision record (ADR) that contains justifications for architectural decisions that are made during the design process.
☐	Collaborate with the workload and platform team during implementation to provide clarity and recommendations about the implementation sequence. This collaboration helps you maximize learning and make improvements from the outset. Also renegotiate requirements with stakeholders, if needed.
☐	Support modeling exercises that provide contextualized information about workload concerns. The contextualized information can cover costs, application health, and other areas.

	Deliverable tasks
☐	**Provide optimization recommendations** that are based on observations of usage patterns and changes in workload functionality or cloud provider changes.
☐	**Participate in audit, compliance, and confidence reviews** to provide a valuable perspective to external parties who have the authority to conduct reviews.
☐	**Be a consultant during change reviews** to provide insight into the estimated cost of change and its feasibility.

Next steps

Get started with the Well-Architected Framework pillars, and familiarize yourself with their key concepts.

Azure Well-Architected Framework pillars

Feedback

Was this page helpful? Yes No

Workload architecture design specification

Article • 07/24/2024

A workload architecture design specification is **a detailed specification that describes design choices and is accompanied by diagrams**. The design choices must meet functional and nonfunctional requirements and include provisions for routine, ad hoc, and emergency operations.

For information about diagrams, see Architecture design diagrams.

Workload architecture design, typically expansive, starts with application design and progresses to cloud service selection. These phases mutually inform each other. The combined application and infrastructure design must fulfill all requirements.

Achieving a solution that satisfies all requirements is a **collaborative effort among stakeholders, developers, testers, operations teams, and product owners**. The design process should involve refining requirements with clarity and negotiation. The process is iterative and often requires multiple reviews.

We recommend that you align your design with the Azure Well-Architected Framework fundamental guidance, which includes design principles and recommendation guides, and acknowledge the tradeoffs.

Ultimately, the workload architecture design specification is implemented by the workload development team, so it must be clear and unambiguous. The specification should be readily available and stored with the workload's documentation.

Functional specification

The functional specification for a workload details the *what* and the *why* of the system or feature under development, but not the implementation. This document must explain the current problems that exist and how this feature or system is going to improve that experience. This document captures most of the business requirements.

An architect can help shape this document, but primarily it's a function of product ownership. An architect should help design the data that is captured in this specification. This involvement ensures the functional specification drives effective and efficient technical design.

Here are a few example topics that should be covered in this specification.

- In addition to detailing what is *in scope* for this design, also be explicit about adjacent concerns that are *out of scope*. Setting clear scopes reduces scope creep by defining boundaries around the functionality.
- It's helpful to include the details on how this change is going to be measured. What measurements need to be collected and what business goals those measurements support.
- User flows should be clearly described. Low-fedelity mockups can be helpful as well. If error handling situations are important for these flows, ensure the expected behavior described.
- Always include any specific requirements for accessibility, compliance, performance, privacy, or security.
- Include any planned rollout strategy. For example, "This feature is going to be available for our beta users for two months before deciding on a full release."

Avoid specific technical implementation details in this specification. These functional specifications will drive technical specifications created by the architect.

Technical specification

The technical specification describes the *how* based on the scope and goals described in the functional specification. This specification is designed for the engineering team to use as a plan-of-record during implementation.

In this specification include items such as:

- Technology decisions, including as: buy, build, reuse, extend, or decommission.
- API and data contracts (schemas), including backwards compatibility implementation strategy
- Rollout and rollback implementation details
- Unique secure development lifecycle (SDL) and privacy implementation
- The test plan
- Key monitoring and alert signal sources
- Alternative designs that were considered

The technical specification will drive engineering efforts. Engineering work items are primarily created from the contents this specification. Implementation teams refer to the work items, the technical specification, and the functional specification to ensure the final result is meeting both the functional and nonfunctional requirements.

Disaster recovery plans

In order to meet the reliability requirements for the workload, an architect needs to design a system that can recover within the target recovery time objective (RTO) and recovery point objective (RPO) goals. The architecture design specification must include the recovery plan. This plan must cover the involved architecture components, failover mechanisms and impact to user and data flow, and operational recommendations. It should describe which recovery targets are met by the design and how.

Although the initial plan is expected to evolve based on insights from drills and post-incident reviews, it's the architect's responsibility to deliver the initial plan for all new architecture.

Security and compliance documentation

An architect is responsible for designing a solution that adheres to pertinent security and compliance constraints. It's important for the design artifacts to highlight the affordances incorporated in the design to support these requirements, and to identify any necessary compensating controls when requirements can't be met directly.

Consistency

Use a template to document your workload's various specifications. Consistency helps stakeholders, responsible parties, and implementation teams when the specification is being read.

Ensure specs have key metadata fields such as:

- **State**: A status as *Draft*, *In review*, and *Approved*.
- **Work item link**: A link to the primary work item in the team's backlog.
- **Key cross links**: Links to related specifications or other documentation that supports the specification.
- **Key individuals**: A place to list the names of key decision makers involved. This list might include roles like: business analyst, business partner, technical lead, and the product owner or lead that signed off on the specification.

Next steps

Architect's checklist

Feedback

Was this page helpful? Yes No

Architecture design diagrams

Article • 07/18/2024

Architects often communicate through diagrams. Diagrams are powerful communication tools that help implementers and stakeholders see the broad vision or dive deep into highly sensitive or nuanced areas of a system. To communicate with intention, an architect must select which diagram is useful in each situation.

The list of diagrams in this article isn't exhaustive. Diagrams are often a composite of multiple types.

Ultimately, the choice of architecture diagram depends on what you're trying to convey and the audience profile. An architect uses multiple types of diagrams throughout activities for design, refinement of requirements, and communication.

Diagramming practices

Diagrams present substantial information without the need for textual explanation. Avoid ambiguity in diagrams. Here are some recommendations:

Use standard notations. Use widely recognized symbols, icons, and presentation conventions for good readability and interpretation of a diagram.

Avoid ambiguous lines. Diagrams often show relationships between entities represented as lines. Be consistent in how you use the lines.

Avoid lines without arrows. It's hard to know what the relationship is without direction, so use arrows. Label all lines without arrows to denote the relationships.

Avoid lines with double arrows. Double arrows imply a bidirectional dependency. Prefer using a single-ended arrow to represent the flow from the dependent (client) to the dependency (server).

Label everything. Provide clear, accurate, and meaningful labels for each icon. Label lines when the relationships aren't clear.

Maintain consistency. Use standardized colors, casing, icons, icon sizes, line types, arrow heads, and other representations for similar elements throughout a diagram. Stay consistent across all diagrams created for your workload's design and documentation. Draw from existing data or taxonomies.

Be accurate. Diagrams are abstractions, but don't sacrifice accuracy in the process. For example, don't represent a service in a virtual network if it's not present in that virtual

network. A diagram is a communication tool, so you need to avoid miscommunication from inaccuracies.

Include metadata. Ensure that a diagram contains metadata that provides essential information about the purpose of the diagram. Metadata also gives context to help viewers understand the diagram's scope and significance. Include items such as title, description, last updated date, author, and external references.

Use official icons and service names. When you're representing a specific technology, use the latest official icons from your technology provider. If identifying the technology is important, use the official name for the service.

For example, here are the icons for Microsoft services:

- Azure architecture icons
- Microsoft 365 icons
- Microsoft Dynamics 365 icons
- Microsoft Entra ID architecture icons
- Microsoft Power Platform icons

Types of design diagrams

Workload architecture is complex and multidimensional. Each dimension type focuses on a specific aspect of the system by providing a level of detail that's specific to that dimension. For instance, flowcharts illustrate process flow. Entity-relationship diagrams depict relationships between system components.

Having different types of diagrams allows for a comprehensive understanding of dimensions. It helps encourage effective communication, problem-solving, and decision-making among stakeholders.

High-level system diagram

A high-level system diagram serves as a broad overview of a whole workload or of a subsection within a workload. It includes the main components, their relationships to each other, and the rough order in which data flows through the system. Arrows show the direction of interaction.

These diagrams are good for reaching a common understanding so that you can start deeper discussions or for stakeholder communication.

Block diagram

A block diagram breaks down a workload into its major functional blocks. The blocks are usually technology agnostic. They refer to the functionality that's being performed instead of a specific component.

For example, a block diagram might reference a "messaging bus" instead of a specific message-bus technology. This type of diagram can help explain a system's structure, data flow, and processing flow without distracting the audience with fine details.

Component diagram

A component diagram works like a block diagram but replaces generic functionality blocks with specific technologies. It presents a detailed view with the goal of communicating the system's individual technology components and their relationships, such as client/server. These diagrams are a sort of visual bill of materials for the scope of the diagram.

Deployment diagram

A deployment diagram focuses on the deployment of infrastructure, commercial off-the-shelf (COTS) software, and custom code across the workload. It shows how the software and code are distributed across the hosting infrastructure.

Data-flow diagram

A data-flow diagram (DFD) illustrates how data moves through a system, which is useful when you're modeling data-centric systems. In a diagram like this, it's a good idea to note if data is moved in batches or in real time to remove ambiguity.

Sequence diagram

A sequence diagram depicts the communication exchanges between workload components over time. It illustrates client/server relationships and their synchronous or asynchronous nature. It also highlights dependencies in these exchanges and evaluates fault scenarios within them.

User-flow diagram

A user-flow diagram focuses on a scoped interaction between workloads, users, or actors and the workload. It's helpful for clarifying and visualizing functional requirements across various ways that a user and the user's data interact with the system.

Entity-relationship diagram

An entity-relationship diagram (ERD) is a modeling diagram that represents the structure of a database or another storage system. It shows the relationship between entities (such as tables) through industry-standard attributes and association symbology.

Network diagram

A network diagram illustrates the solution from the perspective of the network that it runs on or interacts with. These diagrams are useful in visualizing the workload's network segmentation, network points of failure, and key network transitions such as internet egress and ingress points.

Network diagrams usually have a life past implementation. Network diagrams are often used in audits and incident response.

State diagram

A state diagram is a specialized visualization. It shows the state that a flow (or an individual component) is in. It also shows how the flow transitions between states in response to conditions or events.

Flowchart

Although it's not an architecture diagram specifically, a flowchart is another way to bring clarity to a design. Flowcharts are often useful when they represent complex workflows or logic. You can use them to help refine requirements and to help drive implementation choices.

Flowcharts are useful to include in workload incident response plans to help highlight key decision points and their associated actions or notification channels.

Next steps

Architect's checklist

Feedback

Was this page helpful? Yes No

Architecture decision record

Article • 10/10/2024

An architecture decision record (ADR) is one of the most important deliverables of a solution architect. This record documents architectural decisions that you make throughout the design process. It also provides context-specific justifications and implications for each decision.

The ADR documents all key decisions, including alternatives that you ruled out, for architecturally significant requirements. The log incorporates requirements and constraints into the documented effects of a decision.

Implement an ADR

Start the ADR at the onset of a workload and maintain it throughout the workload's lifespan. An ADR should be started for brownfield workloads, and if the data is available, it should be retroactively generated based on known past decisions. The ADR serves as an append-only log. This record extends beyond the initial design to include design aspects against future functional and nonfunctional requirements.

Suggested characteristics of an individual record

- Be consistent with the anatomy of every record. Design a record template and consistently follow it.

  A record should include consistent elements such as:
  - Problem statement with context
  - Options considered
  - Decision outcome
    - Include important tradeoffs made with this decision
    - Record the confidence level of the decision. Sometimes an architecturally significant decision is made with relatively low confidence. Documenting that low confidence status could prove useful for future reconsideration decisions.

- Break one decision into multiple if an architectural decision is going to result in multiple phases, such as short-term, mid-term, long-term approaches. Log each phase as its own decision record.

- Avoid hiding consequences of decisions intentionally or accidentally.

- Keep records pithy, assertive, on-topic, and factual.

- Avoid making decision records design guides. If more justification or design ideation is available, provide a link to a document as supplemental material, but the decision must be clear and stand alone without that material.

Workload documentation repository

This log should be readily available and stored openly with the workload's documentation.

An architect can help a workload team achieve recommendations of Operational Excellence by helping to establish and maintain a document and asset repository. Architects help teams place all of their assets into the repository, such as this decision log. They also encourage the teams' unified adoption of a single source of truth to be used for reference, audits, and incident response.

Next steps

Architect's checklist

More resources

For template examples, visit the Architectural Decision Records GitHub organization.

Feedback

Was this page helpful? Yes No

Collaboration with the workload team

Article • 11/01/2024

Delivering architecture specifications isn't a one-off task. An architect should expect to engage with the workload team throughout the implementation.

Continuous collaboration tasks

- **Provide clarity**. Architects should be readily available to provide clarity on any delivered specifications to ensure that implementation teams remain unblocked. To address potential blockers, architects should actively participate in iteration planning exercises and team meetings.
- **Make implementation sequencing suggestions**. Architects understand that the journey from design to a production-ready product is iterative. They can recommend which parts of the application to implement first. This approach enables the workload team to learn from that experience and apply the knowledge that they gain to the remaining parts of the workload.
- **Set implementation review checkpoints**. Workload teams should establish regular review checkpoints for comparing the implementation with the architectural specification. This practice helps ensure that the design is implemented according to the specification and that the specification meets the predicted requirements. This feedback loop can rectify any design or implementation errors.
- **Communicate with stakeholders**. Architects have an established relationship with stakeholders and the business and have an intimate understanding of the workload. As a result, they're often in a good position to relay implementation team concerns or requests for negotiating changes to requirements.
- **Make environment recommendations**. Workload design often extends beyond designing for production and its disaster recovery. Production is just one of many environments a workload implementation team might need. Architects can also assist workload teams in right-sizing preproduction environments.
- **Use a proof of concept (POC)**. Architects frequently use POCs in their designs to inform decisions about the design specifications for the workload architecture. These POCs can also provide insight into the feasibility of the actual workload implementation. If a POC doesn't exist, an architect should create one before the implementation team starts development.

Collaboration with platform teams

Some organizations split responsibilities between workload teams and platform teams, such as suggested in Azure landing zones. For workloads that will be partnering with platform teams to co-deliver solutions and value, it's important to collaborate with those teams. If you don't collaborate with platform teams, your design might miss out on taking cost advantage of available platform-provided offerings or you might design a solution that violates platform-mandated constraints on workloads.

Next steps

Architect's checklist

Feedback

Was this page helpful? Yes No

Support the workload in a consultative role

Article • 11/01/2024

Architects should seek ways to stay involved with the workload as it changes over time. Their role doesn't end with design handoff or consultation during the initial implementation. Architects bring perspective that can be used in other activities that are related to the product's evolution.

Support modeling exercises

Teams can model workloads in multiple dimensions and for multiple purposes. For example, workloads can abstract health signals out of implementation details and into business constructs. Or they can model system growth over time or the licensing process to evaluate alternative billing models.

Whether the model is an abstraction or evaluates hypotheticals to inform a future business decision, architects contribute to that process. They use their insight into the workload design, its known or predicted limitations, and its scaling characteristics, to validate or tune the assumptions in the model and approximate the system more accurately. For example, architects review the health model for a critical flow by evaluating the characteristics of dependencies, such as service-level objectives (SLOs).

Share potential improvements

Architects stay current with fundamentals like cloud provider offerings and industry design patterns. Features that were state of the art when a workload was designed might no longer be. Or the expected usage patterns of the application might not manifest in the way that they were predicted. In cases like these, there's an opportunity for you to present a recommendation to further optimize or refine the current design based on this new knowledge.

As an architect, you should follow up with the workload team periodically after the workload is live. Continued communication helps you expand your knowledge for future design work by seeing how the design was implemented and how it's performing with actual use. It also allows you to offer optimization recommendations based on the actual implementation and use.

Assist in reviews

When a workload is under review, such as by an official audit or a compliance review, the system architect's involvement can be a boon to the process. They bring the workload's architecture decision record to help answer questions on implementation choices. They also provide updated diagrams to visualize the system during conversations and provide subject-matter expertise.

Architects have authoritative knowledge that builds confidence in the product during select customer or funding engagements. They can learn about unique demands that customers have for the product and consider those needs in the design of the system.

Review proposed changes

Every workload has a backlog of work that ranges from broad, directional-level work to specific tasks. Architects should be involved in gathering requirements, scoping, and building acceptance criteria for work items.

The implementation team is busy delivering on current work items, so architects can use their time to review, validate, and refine future work items. They can help detect when a new feature requires a redesign of a component in the system, provide cost analysis on a proposed change, or propose an approach to incrementally introduce new changes. Ultimately, involving an architect early in the process for a proposed change that involves new functionality or an expanded user base minimizes rework and helps the team discover cliffs in the design.

Next steps

Architect's checklist

Feedback

Was this page helpful? Yes No

Microsoft Azure Well-Architected Framework pillars

Article • 01/22/2025

The Azure Well-Architected Framework pillars drive architectural excellence at the fundamental level of a workload.

Use this matrix to familiarize yourself with the key concepts:

Expand table

Pillar	Workload concern	Apply the principles	Strike a balance
Reliability	Resiliency, availability, recovery	Design for business requirements, resilience, recovery, and operations, while keeping it simple. Design principles	Tradeoffs
Security	Data protection, threat detection, and mitigation	Protect confidentiality, integrity, and availability. Design principles	Tradeoffs
Cost Optimization	Cost modeling, budgets, reduce waste	Optimize on usage and rate utilization while keeping a cost-efficient mindset. Design principles	Tradeoffs
Operational Excellence	Holistic observability, DevOps practices	Streamline operations with standards, comprehensive monitoring, and safe deployment practices. Design principles	Tradeoffs
Performance Efficiency	Scalability, load testing	Scale horizontally, test early and often, and monitor the health of the solution. Design principles	Tradeoffs

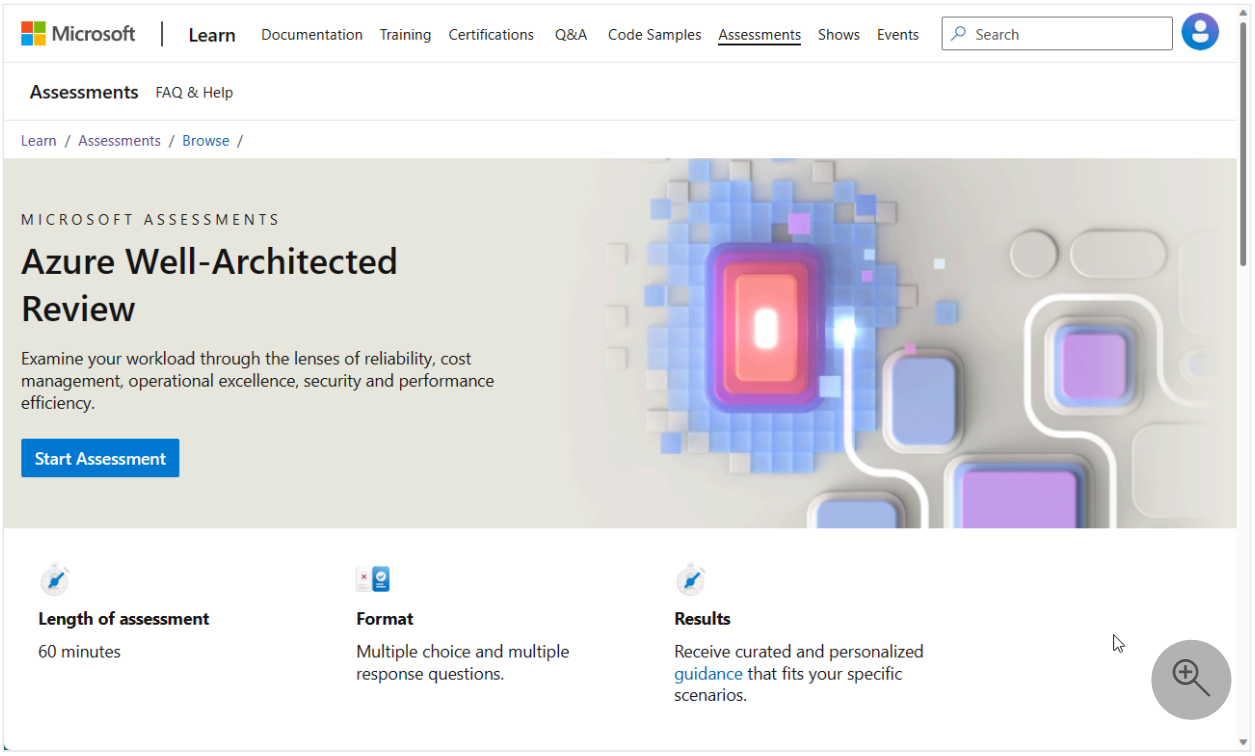
Important

The reference architectures available in the Azure Architecture Center are designed with the design principles in mind. The architecture articles describe a prescriptive path for applying the design principles and provide a holistic view on the Ten design principles for Azure applications.

Assessment review tool

Assess your workload by using the core pillars to **identify and prioritize opportunities** for improving the posture of your workloads.

Start your assessment with the Azure Well-Architected Review.



Integrate with Azure recommendation services

- Azure Advisor uses the **core pillars** as a basis for analyzing your resource configuration and usage telemetry and provides appropriate recommendations.
- Azure Advisor score is a core feature of Azure Advisor that aggregates Advisor recommendations into a simple, actionable score. You can categorize the overall score into the core pillars of the Well-Architected Framework. Use the score to prioritize the actions that yield the biggest improvement.

Feedback

Was this page helpful? Yes No

Reliability quick links

Apply reliability guidance to your architecture to make your workload resilient to malfunction and to ensure that it returns to a fully functioning state after a failure occurs.

Learn key points

QUICKSTART

Design principles

Checklist

Tradeoffs

Reliability patterns

Azure Well-Architected Review assessment

TRAINING

Reliability

VIDEO

Inside Azure datacenter architecture with Mark Russinovich

Review design principles

CONCEPT

Design for business requirements

Design for resilience

Design for recovery

Design for operations

Keep it simple

Set and measure reliability targets

HOW-TO GUIDE

Identify flows

Perform failure mode analysis

Set reliability targets

Design a monitoring and alerting strategy

Achieve reliability targets

HOW-TO GUIDE

Design for redundancy

Use availability zones and regions

Implement highly available multi-region design

Partition data

Reliably scale

Design for self-healing and preservation

Design with simplicity

Test and conduct drills

HOW-TO GUIDE

Design a testing strategy

Design a recovery strategy

Explore related resources

REFERENCE

Azure Advisor: Reliability recommendations

Azure reliability documentation

Site reliability engineering documentation

Reliability design principles

Article • 11/14/2023

Outages and malfunctions are serious concerns for all workloads. A reliable workload must survive those events and **continue to consistently provide its intended functionality**. It must be **resilient** so that it can detect, withstand, and recover from failures within an acceptable time period. It must also be **available** so that users can access the workload during the promised time period at the promised quality level.

It's not realistic to assume failures won't occur, especially when the workload is built to run on distributed systems. Some components might fail while others continue to operate. At some point, the user experience might be affected, which compromises business goals.

Workload architectures should have **reliability assurances in application code, infrastructure, and operations**. Design choices shouldn't change the intent that's specified by business requirements. Such changes should be considered significant tradeoffs.

The **design principles** are intended to provide guidance for aspects of reliability that you should consider throughout the development lifecycle. Start with the recommended approaches and **justify the benefits for a set of requirements**. After you set your strategy, drive actions by using the Reliability checklist.

If you don't apply these principles to your design, the workload most likely won't be prepared to **anticipate or handle problems in production**. The outcome might be service disruptions that lead to financial loss. In the case of critical workloads, failing to apply these principles could jeopardize safety.

Design for business requirements



Gather business requirements with a focus on the intended utility of the workload.

Requirements must cover user experience, data, workflows, and characteristics that are unique to the workload. The outcome of *the requirements process must clearly state the expectations*. The goals must be achievable and negotiated with the team, given a specified investment. They must be documented to drive technological choices, implementations, and operations.

Approach	Benefit
Quantify success by setting targets on indicators for individual components, system flows, and the system as a whole. **Do those targets make user flows more reliable?**	Metrics quantify expectations. They enable you to **understand complexities** and determine whether the downstream costs of those complexities are within the investment limit. The target values indicate an ideal state. You can use the values as test thresholds that help you **detect deviations** from that state and how long it takes to return to the target state. Compliance requirements must also have predictable outcomes for in-scope flows. Prioritizing these flows **bring attention to areas that are the most sensitive**.
Understand platform commitments. **Consider the limits, quotas, and capacity constraints** for services.	Service-level agreements (SLAs) vary by service. Not all services and features are covered equally. Having a good **understanding of coverage and limits** can help you detect drift and build resiliency and recovery mechanisms.
Determine dependencies and their effect on resiliency.	Keeping track of dependent infrastructure, services, APIs, and functions developed by other teams or third parties helps you determine **whether the workload can operate in absence of those dependencies**. It also helps you **understand cascading failures** and **improve downstream operations**. Developers can **implement resilient design patterns** to handle potential failures when you use external services that might be susceptible to failures.

Design for resilience

 **The workload must continue to operate with full or reduced functionality.**

You should expect that component malfunctions, platform outages, performance degradations, and other faults will occur. Build resiliency in the system so that it's *fault-tolerant and can degrade gracefully*.

Approach	Benefit
Distinguish components that are on the critical path from those	Not all components of the workload need to be equally reliable. Determining criticality helps you design according to the **criticality of each component**. You **won't**

Approach	Benefit
that can function in a degraded state.	**overengineer resiliency for components** that could slightly deteriorate the user experience, as opposed to components that can cause end-to-end problems if they fail. The design can be efficient in allocating resources to critical components. You can also implement fault isolation strategies so that if a noncritical component fails or enters a degraded state, it can be isolated to prevent cascading failures.
Identify potential failure points in the system, especially for the critical components, and determine the effect on user flows.	You can **analyze the failure cases, blast radius, and intensity of fault**: full or partial outage. This analysis influences the design of error handling capabilities at the component level.
Build self-preservation capabilities by using design patterns correctly and modularizing the design to isolate faults.	The system will be able to **prevent a problem from affecting downstream components**. The system will be able to mitigate transient and permanent failures, performance bottlenecks, and other problems that might affect reliability. You'll also be able to **minimize the blast radius**.
Add the capability to scale out the critical components (application and infrastructure) by considering the capacity constraints of services in the supported regions.	The workload will be able to **handle variable capacity spikes and fluctuations**. This capability is crucial when there's an unexpected load on the system, like a surge in valid usage.
Build redundancy in layers and resiliency on various application tiers. Aim for redundancy in physical utilities and immediate data replication. Also aim for redundancy in the functional layer that covers services, operations, and personnel.	Redundancy helps **minimize single points of failure**. For example, if there's a component, zonal, or regional outage, redundant deployment (in active-active or active-passive) allows you to meet uptime targets. Adding intermediaries prevents direct dependency between components and improves buffering. Both of these benefits harden the resiliency of the system.
Overprovision to immediately mitigate individual failure of redundant instances and to buffer against runaway resource consumption.	Higher investment in overprovisioning **increases resiliency**. The system will continue to operate at full utility during an active failure even before scaling operations can start to **remediate the failure**. Likewise, you can reduce the risk of unexpected runaway resource consumption claiming your planned buffer, gaining critical triage time, before system faults or aggressive scaling occurs.

Design for recovery

The workload must be able to anticipate and recover from most failures, of all magnitudes, with minimal disruption to the user experience and business objectives.

Even highly resilient systems need *disaster preparedness approaches*, in both architecture design and workload operations. On the data layer, you should have strategies that can repair workload state in case of corruption.

Approach	Benefit
Have structured, tested, and documented recovery plans that are aligned with the negotiated recovery targets. Plans must cover all components in addition to the system as a whole.	A well-defined process leads to a **quick recovery** that can prevent negative impact on the finances and reputation of your business. Conducting regular recovery drills tests the process of recovering system components, data, and failover and failback steps to **avoid confusion when time and data integrity are key** measures of success.
Ensure that you can **repair data** of all stateful components within your recovery targets.	Backups are essential to getting the **system back to a working state** by using a trusted recovery point, like the last-known good state. Immutable and transactionally consistent backups ensure that data can't be altered, and that the restored data isn't corrupted.
Implement **automated self-healing capabilities** in the design.	This automation **reduces risks from external factors**, like human intervention, and shortens the break-fix cycle.
Replace stateless components with **immutable ephemeral units**.	Building ephemeral units that you can spin up and destroy on demand provides **repeatability and consistency**. Use side-by-side deployment models to make the transition to the new units incremental, minimizing disruptions.

Design for operations

Shift left in operations to anticipate failure conditions.

Test failures early and often in the development lifecycle, and determine the impact of performance on reliability. For the sake of root cause analysis and postmortems, you need to have shared visibility, across teams, of dependency status and ongoing failures. *Insights, diagnostics, and alerts from observable systems are fundamental* to effective incident management and continuous improvement.

Approach	Benefit
Build observable systems that can correlate telemetry.	Monitoring and diagnostics are crucial operations. If something fails, you need to know that it failed, **when it failed**, and **why it failed**. Observability at the component level is fundamental, but aggregated observability of components and correlated user flows provides a holistic view of health status. This data is required to enable site-reliability engineers to prioritize their efforts for remediation.
Predict potential malfunctions and anomalous behavior. Make active reliability failures visible by using prioritized and actionable alerts. Invest in reliable processes and infrastructure that leads to quicker triage.	Site reliability engineers can be notified immediately so that they can **mitigate ongoing live site incidents** and proactively mitigate potential failures identified by predictive alerts before they become live incidents.
Simulate failures and run tests in production and pre-production environments.	It's beneficial to experience failures in production so you can set **realistic expectations for recovery**. This allows you to make design choices that gracefully respond to failures. Also, it enables you to test the thresholds you set for business metrics.
Build components with **automation in mind**, and automate as much as you can.	Automation **minimizes the potential for human error**, bringing **consistency** to testing, deployment, and operations.
Factor in **routine operations and their impact** on the stability of the system.	The workload might be subject to ongoing operations, like application revisions, security and compliance audits, component upgrades, and backup processes. Scrutinizing those changes ensures the **stability of the system**.
Continuously **learn from incidents in production**.	Based on the incidents, you can determine the impact and oversights in design and operations that might go unnoticed in preproduction. Ultimately, you'll be able to **drive improvements** based on real-life incidents.

Keep it simple

 **Avoid overengineering the architecture design, application code, and operations.**

It's often what you remove rather than what you add that leads to the most reliable solutions. *Simplicity reduces the surface area for control*, minimizing inefficiencies and

potential misconfigurations or unexpected interactions. On the other hand, oversimplification can introduce single points of failure. Maintain a balanced approach.

Approach	Benefit
Add components to your architecture only if they help you achieve target business values. **Keep the critical path lean**.	Designing for business requirements can lead to a straightforward solution that's **easy to implement and manage**. Avoid having too many critical components, because each one is a significant point of failure.
Establish standards in code implementation, deployment, and processes, and document them. Identify opportunities to enforce those standards by using automated validations.	Standards provide **consistency and minimize human errors**. Approaches like standard naming conventions and code style guides can help you maintain quality and make assets easy to identify during troubleshooting.
Evaluate whether theoretical approaches translate to **pragmatic design** that applies to your use cases.	Application code that's too granular can lead to unnecessary interdependence, extra operations, and **difficult maintenance**.
Develop just enough code.	You'll be able to prevent problems that are the result of **inefficient implementations**, like unexpected resource consumption, user or dataflow failures, and code bugs. Conversely, reliability problems should lead to code reviews to ensure that code is resilient enough to handle the problems.
Take advantage of platform-provided features and prebuilt assets that can help you effectively meet business targets.	This approach **minimizes development time**. It also enables you to **rely on tried and tested practices** that have been used with similar workloads.

Next steps

Reliability checklist

Design review checklist for Reliability

Article • 02/27/2025

This checklist presents a set of recommendations for you to use to evaluate the reliability, resiliency, and failure recovery strategies in your architecture design. To ensure reliability, identify the best infrastructure and application design for your workload. Make these decisions based on your business requirements that are mapped to availability and recoverability target metrics.

To implement a reliable design, thoroughly consider decision points in your design and be aware of how those decisions affect your workload. This checklist and the accompanying guides provide resources to help you make those decisions. Make workload reliability a central consideration throughout the workload design, development, and operation lifecycle.

Checklist

Approach your design with a focus on reliability to help ensure that you design a workload that's resilient, manageable, and repeatable. If you don't include reliability practices and consider the tradeoffs, your design is potentially at risk. Carefully consider all the points covered in the checklist to instill confidence in your system's success.

Expand table

	Code	Recommendation
☐	RE:01	**Focus your workload design on simplicity and efficiency.** Use a practical approach to avoid unnecessary complexity while meeting your business goals and requirements.
☐	RE:02	**Identify and rate user and system flows.** Use a criticality scale based on your business requirements to prioritize the flows.
☐	RE:03	**Use failure mode analysis (FMA) to identify potential failures in your workload.** Identify dependencies and failure points, and develop mitigation strategies for those failures.
☐	RE:04	**Define reliability and recovery targets** for your workload. Use the targets to inform your design and as the foundation of your health model.
☐	RE:05 RE:05 RE:05	**Add redundancy at different levels, especially for critical flows**, to help meet your reliability targets. Consider redundant infrastructure components such as compute and network, and multiple instances of your solution.

	Code	Recommendation
☐	RE:06 RE:06	**Implement a timely and reliable scaling strategy at the application, data, and infrastructure levels**. Base the scaling strategy on actual or predicted usage patterns and minimize manual intervention.
☐	RE:07	**Strengthen the resiliency of your workload by implementing self-preservation and self-healing measures**. Use built-in features and well-established cloud patterns to help your workload remain functional during and recover from incidents.
☐	RE:08	**Test for resiliency and availability scenarios by applying the principles of chaos engineering**. Ensure that your graceful degradation implementation and scaling strategies are effective by performing active malfunction and simulated load testing.
☐	RE:09	**Implement structured, tested, and documented business continuity and disaster recovery (BCDR) plans** that align with the recovery targets. Plans must cover all components and the system as a whole.
☐	RE:10	**Measure and model the solution's health signals**. Continuously capture uptime and other reliability data from across the workload and also from individual components and key flows.

Next steps

We recommend that you review the Reliability tradeoffs to explore other concepts.

Reliability tradeoffs

Feedback

Was this page helpful? Yes No

Reliability tradeoffs

Article • 10/10/2024

A reliable workload consistently meets its defined reliability objectives. It should reach established resiliency targets, ideally by circumventing events that affect reliability. Realistically, however, a workload must tolerate and control the impact of such events and maintain operations at a predetermined level during active malfunction. Even during a disaster, a reliable workload must recover to a specific state within a given period of time, both of which are agreed upon among the stakeholders. An incident response plan that enables you to achieve rapid detection and recovery is vital.

During the design phase of a workload, you need to consider how decisions based on the Reliability design principles and the recommendations in the Design review checklist for Reliability might influence the goals and optimizations of other pillars. Certain decisions might benefit some pillars but constitute a tradeoff for others. This article describes example tradeoffs that a workload team might encounter when designing workload architecture and operations for reliability.

Reliability tradeoffs with Security

>  **Tradeoff: Increased workload surface area.** The Security pillar prioritizes a reduced and contained surface area to minimize attack vectors and reduce the management of security controls.

- Reliability is often obtained through replication. Replication can occur at the component level, at the data level, or even at a geographic level. Replicas, by design, increase the surface area of a workload. From a security perspective, a reduced and contained surface area is preferred to minimize potential attack vectors and streamline the management of security controls.

- Similarly, disaster recovery solutions, like backups, increase a workload's surface area. However, they're often isolated from the workload's runtime. These solutions require the implementation of additional security controls, which might be specific to the disaster recovery approach.

- For the sake of reliability goals, additional components might be needed for the architecture, which increases the surface area. For example, a message bus might be added to make requests resilient through decoupling. This increased complexity increases the surface area of the workload by adding new components that need

to be secured, possibly in ways that aren't already used in the system. Typically, these components are accompanied by additional code and libraries to support their use or general reliability patterns, which also increases application surface area.

>  **Tradeoff: Security control bypass.** The Security pillar recommends that all controls remain active in both normal and stressed systems.

- When a workload is experiencing a reliability event that's being addressed under active incident response, urgency might create pressure for workload teams to bypass security controls that are optimized for routine access.
- Troubleshooting activities can cause the team to temporary disable security protocols, leaving an already stressed system potentially exposed to additional security risks. There's also a risk that the security protocols won't be reestablished promptly.
- Granular implementations of security controls, like custom role-based access control assignments or narrow firewall rules, introduce configuration complexity and sensitivity, increasing the chance for misconfiguration. Mitigating this potential reliability impact by using broad rules erodes all three Zero Trust architecture principles.

>  **Tradeoff: Old software versions.** The Security pillar encourages a "get current, stay current" approach to vendor security patches.

- Applying security patches or software updates can potentially disrupt the target component, causing unavailability during the software change. Delaying or avoiding patching might avoid the potential reliability risks, but it leaves the system unprotected against evolving threats.
- The preceding consideration also applies to the workload's code. For example, it applies to application code that uses old libraries and containers that use old base images. If updating and deploying application code is viewed as an unmitigated reliability risk, the application is exposed to additional security risks over time.

Reliability tradeoffs with Cost Optimization

>  **Tradeoff: Increased implementation redundancy or waste.** A cost-optimized workload minimizes underutilized resources and avoids over-provisioning resources.

- Replication is a key strategy for reliability. Specifically, the strategy is to have enough replication to handle a given number of concurrent node failures. The tolerance for more concurrent node failures requires a higher replica count, which leads to increased costs.
- Over-provisioning is another technique for absorbing unexpected load on a system, such as during a failover event, that could otherwise lead to a reliability issue. Any excess capacity that's not utilized is considered wasteful.
- If a workload uses a disaster recovery solution that excessively satisfies the workload's recovery point and time objectives, the excess leads to higher costs because of waste.
- Workload deployments themselves are a potential source for reliability impact, and that impact is often mitigated by redundancy at deployment time via a deployment strategy like blue/green. This transient duplication of resources during safe deployment typically increases the overall cost of the workload during those periods. Costs increase with frequency of deployments.

>  **Tradeoff: Increased investment in operations that aren't aligned with functional requirements.** One approach to cost optimization is evaluating the value that's provided by any deployed solution.

- To achieve reliability, a system requires observability. Monitoring systems require observability data transfer and collection. As monitoring capabilities increase, the frequency and volume of data increase, leading to additional costs.
- Reliability affordances in workloads necessitate testing and drills. Designing and running tests takes time and potentially specialized tooling, which incurs costs.
- Workloads with high reliability targets often have a rapid response process that requires technical team members to be part of a formal on-call rotation. This process incurs additional personnel costs and lost opportunity costs because of attention that could be directed elsewhere. It also incurs potential tooling costs for management of the process.
- Support contracts with technology providers are a key component of a reliable workload. Support contracts that aren't utilized because the level of support is over-provisioned incur waste.

Reliability tradeoffs with Operational Excellence

>  **Tradeoff: Increased operational complexity.** Operational Excellence, like Reliability itself, prioritizes simplicity.

- Reliability usually increases the complexity of a workload. As the complexity of a workload increases, the operational elements of the workload can also increase to support the added components and processes in terms of deployment coordination and configuration surface area.
- Having a comprehensive monitoring strategy for a workload is a key part of operational excellence. Introducing additional components into an architecture to implement reliability design patterns results in more data sources to manage, increasing the complexity of implementing distributed tracing and observability.
- Using multiple regions to overcome single region resource capacity constraints and/or implement an active/active architecture increases the complexity of the workload's operational management. This complexity is introduced by the need to manage multiple regions and the need to manage the data replication between them.

>  **Tradeoff: Increased effort to generate team knowledge and awareness.** The Operational Excellence pillar recommends keeping and maintaining a documentation repository for procedures and topologies.

- As a workload becomes more robust through the addition of reliability components and patterns, it takes more time to maintain operational procedures and artifact documentation.
- Training becomes more complex as the number of components in the workload increases. This complexity affects the time required for onboarding. The complexity also increases the knowledge that's needed to track product roadmaps and the latest service-level guidance.

Reliability tradeoffs with Performance Efficiency

>  **Tradeoff: Increased latency.** Performance Efficiency requires a system to achieve performance targets for user and data flows.

- Reliability patterns often incorporate data replication to survive replica malfunction. Replication introduces additional latency for reliable data-write operations, which consumes a part of the performance budget for a specific user or data flow.
- Reliability sometimes employs various forms of resource balancing to distribute or redistribute load to healthy replicas. A dedicated component that's used for balancing usually affects the performance of the request or process that's being balanced.
- Distributing components across geographical boundaries or availability zones to survive a scoped impact introduces network latency in the communication between components that span those availability boundaries.
- Extensive processes are used to observe the health of a workload. Although monitoring is critical for reliability, instrumentation can affect system performance. As observability increases, performance might decrease.

>  **Tradeoff: Increased over-provisioning.** The Performance Efficiency pillar discourages over-provisioning, instead recommending the use of just enough resources to satisfy demand.

- Automatic scaling operations aren't instantaneous and therefore can't reliably handle a sudden and dramatic spike in demand that can't be shaped or smoothed. Therefore, over-provisioning via either larger instances or more instances is a critical reliability tactic to account for the lag between demand signal and supply creation to help absorb bursts. Unused capacity counters the goals of performance efficiency.
- Sometimes a component can't be scaled in reaction to demand, and that demand isn't fully predictable. Using large instances to cover the worst case leads to over-provisioning waste in situations that are outside that use case.

Related links

Explore the tradeoffs for the other pillars:

- Security tradeoffs
- Cost Optimization tradeoffs
- Operational Excellence tradeoffs
- Performance Efficiency tradeoffs

Feedback

Was this page helpful? Yes No

Cloud design patterns that support reliability

Article • 10/10/2024

When you design workload architectures, you should use industry patterns that address common challenges. Patterns can help you make intentional tradeoffs within workloads and optimize for your desired outcome. They can also help mitigate risks that originate from specific problems, which can impact security, performance, cost, and operations. If not mitigated, those risks will eventually cause reliability issues. These patterns are backed by real-world experience, are designed for cloud scale and operating models, and are inherently vendor agnostic. Using well-known patterns as a way to standardize your workload design is a component of operational excellence.

Many design patterns directly support one or more architecture pillars. Design patterns that support the Reliability pillar prioritize workload availability, self-preservation, recovery, data and processing integrity, and containment of malfunctions.

Design patterns for reliability

The following table summarizes cloud design patterns that support the goals of reliability.

Expand table

Pattern	Summary
Ambassador	Encapsulates and manages network communications by offloading cross-cutting tasks that are related to network communication. The resulting helper services initiate communication on behalf of the client. This mediation point provides an opportunity to add reliability patterns to network communication, such as retry or buffering.
Backends for Frontends	Individualizes the service layer of a workload by creating separate services that are exclusive to a specific frontend interface. Because of this separation, a malfunction in the service layer that supports one client might not affect the availability of another client's access. When you treat various clients differently, you can prioritize reliability efforts based on expected client access patterns.
Bulkhead	Introduces intentional and complete segmentation between components to isolate the blast radius of malfunctions. This failure isolation strategy attempts to contain faults to just the bulkhead that's experiencing the problem, preventing impact to other bulkheads.

Pattern	Summary
Cache-Aside	Optimizes access to frequently read data by introducing a cache that's populated on demand. The cache is then used on subsequent requests for the same data. Caching creates data replication and, in limited ways, can be used to preserve the availability of frequently accessed data if the origin data store is temporarily unavailable. Additionally, if there's a malfunction in the cache, the workload can fall back to the origin data store.
Circuit Breaker	Prevents continuous requests to a malfunctioning or unavailable dependency. By doing so, this pattern prevents overloading a faulting dependency. You can also use this pattern to trigger graceful degradation in the workload. Circuit breakers are often coupled with automatic recovery to provide both self-preservation and self-healing.
Claim Check	Separates data from the messaging flow, providing a way to separately retrieve the data related to a message. Message buses don't provide the same reliability and disaster recovery that are often present in dedicated data stores, so separating the data from the message can provide increased reliability for the underlying data. This separation also allows for a message queue recovery approach after a disaster.
Compensating Transaction	Provides a mechanism to recover from failures by reversing the effects of previously applied actions. This pattern addresses malfunctions in critical workload paths by using compensation actions, which can involve processes like directly rolling back data changes, breaking transaction locks, or even executing native system behavior to reverse the effect.
Competing Consumers	Applies distributed and concurrent processing to efficiently handle items in a queue. This model builds redundancy in queue processing by treating consumers as replicas, so an instance failure doesn't prevent other consumers from processing queue messages.
Event Sourcing	Treats state change as series of events, capturing them in an immutable, append-only log. You can use this pattern when a reliable history of changes is crucial in a complex business process. It also facilitates state reconstruction if you need to recover state stores.
Federated Identity	Delegates trust to an identity provider that's external to the workload for managing users and providing authentication for your application. Offloading user management and authentication shifts reliability for those components to the identity provider, which usually has a high SLA. Additionally, during workload disaster recovery, authentication components probably don't need to be addressed as part of the workload recovery plan.
Gateway Aggregation	Simplifies client interactions with your workload by aggregating calls to multiple backend services in a single request. This topology enables you

Pattern	Summary
	to shift transient fault handling from a distributed implementation across clients to a centralized implementation.
Gateway Offloading	Offloads request processing to a gateway device before and after forwarding the request to a backend node. Offloading this responsibility to a gateway reduces the complexity of application code on backend nodes. In some cases, offloading completely replaces functionality with a reliable platform-provided feature.
Gateway Routing	Routes incoming network requests to various backend systems based on request intents, business logic, and backend availability. Gateway routing enables you to route traffic to only healthy nodes in your system.
Geode	Deploys systems that operate in active-active availability modes across multiple geographies. This pattern uses data replication to support the ideal that any client can connect to any geographical instance. It can help your workload withstand one or more regional outages.
Health Endpoint Monitoring	Provides a way to monitor the health or status of a system by exposing an endpoint that's specifically designed for that purpose. You can use this endpoint to manage your workload's health and for alerting and dashboarding. You can also use it as a signal for self-healing remediation.
Index Table	Optimizes data retrieval in distributed data stores by enabling clients to look up metadata so that data can be directly retrieved, avoiding the need to do full data store scans. Because clients are pointed to their shard, partition, or endpoint through a lookup process, you can use this pattern to facilitate a failover approach for data access.
Leader Election	Establishes a *leader* of instances of a distributed application. The leader coordinates responsibilities that are related to accomplishing a goal. This pattern mitigates the effect of node malfunctions by reliably redirecting work. It also implements failover via consensus algorithms when a leader malfunctions.
Pipes and Filters	Breaks down complex data processing into a series of independent stages to achieve a specific outcome. The single responsibility of each stage enables focused attention and avoids the distraction of commingled data processing.
Priority Queue	Ensures that higher-priority items are processed and completed before lower-priority items. Separating items based on business priority enables you to focus reliability efforts on the most critical work.
Publisher/Subscriber	Decouples components of an architecture by replacing direct client-to-service or client-to-services communication with communication via an intermediate message broker or event bus.

Pattern	Summary
Queue-Based Load Leveling	Controls the level of incoming requests or tasks by buffering them in a queue and letting the queue processor handle them at a controlled pace. This approach can provide resilience against sudden spikes in demand by decoupling the arrival of tasks from their processing. It can also isolate malfunctions in queue processing so that they don't affect intake.
Rate Limiting	Controls the rate of client requests to reduce throttling errors and avoid unbounded retry-on-error scenarios. This tactic protects the client by acknowledging the limitations and costs of communicating with a service when the service is designed to avoid reaching specified limits. It works by controlling the number and/or size of operations that are sent to the service during a specific time period.
Retry	Addresses failures that might be transient or intermittent by retrying certain operations, in a controlled way. Mitigating transient faults in a distributed system is a key technique for improving a workload's resilience.
Saga distributed transactions	Coordinates long-running and potentially complex transactions by decomposing the work into sequences of smaller, independent transactions. Each transaction must also have compensating actions to reverse failures in execution and maintain integrity. Because monolithic transactions across multiple distributed systems are usually impossible, this pattern provides consistency and reliability by implementing atomicity and compensation.
Scheduler Agent Supervisor	Efficiently distributes and redistributes tasks across a system based on factors that are observable in the system. This pattern uses health metrics to detect failures and reroute tasks to a healthy agent in order to mitigate the effects of a malfunction.
Sequential Convoy	Maintains concurrent messaging ingress while also supporting processing in a defined order. This pattern can eliminate race conditions that are hard to troubleshoot, contentious message handling, or other workarounds for addressing incorrectly ordered messages that can lead to malfunctions.
Sharding	Directs load to a specific logical destination to handle the specific request, enabling colocation for optimization. Because the data or processing is isolated to the shard, a malfunction in one shard remains isolated to that shard.
Strangler Fig	Provides an approach for systematically replacing the components of a running system with new components, often during a migration or modernization of the system. This pattern's incremental approach can help mitigate risks during a transition.
Throttling	Imposes limits on the rate or throughput of incoming requests to a resource or component. You can design the limits to help prevent

Pattern	Summary
	resource exhaustion that might lead to malfunctions. You can also use this pattern as a control mechanism in a graceful degradation plan.

Next steps

Review the cloud design patterns that support the other Azure Well-Architected Framework pillars:

- Cloud design patterns that support security
- Cloud design patterns that support cost optimization
- Cloud design patterns that support operational excellence
- Cloud design patterns that support performance efficiency

Feedback

Was this page helpful? Yes No

Recommendations for designing for simplicity and efficiency

Article • 12/01/2023

Applies to this Azure Well-Architected Framework Reliability checklist recommendation:

Expand table

RE:01	**Design your workload to align with business objectives and avoid unnecessary complexity or overhead. Use a practical and balanced approach to make design decisions that deliver the desired results. Contain your design to the necessities to reduce inefficiencies and potential problems.**

This guide describes the recommendations for minimizing unnecessary complexity and overhead to keep your workloads simple and efficient. Choose the best components to perform the necessary workload tasks to optimize the reliability of your workload. To lessen your development and management burdens, take advantage of efficiencies that platform-provided services offer. This design helps you create a workload architecture that's resilient, repeatable, scalable, and manageable.

Definitions

Expand table

Term	Definition
Workload	A discrete capability or computing task that you can logically separate from other tasks.

Key design strategies

A key tenet of designing for reliability is to keep things simple and efficient. Focus your workload design on meeting business requirements to reduce the risk of unnecessary complexity or excess overhead. Consider the recommendations in this article to help you make decisions about your design to create a lean, efficient, and reliable workload. Different workloads might have different requirements for availability, scalability, data consistency, and disaster recovery.

You must justify every design decision with a business requirement. This design principle might seem obvious, but it's crucial for workload design. Does your application support

millions of users, or a few thousand? Are there large traffic bursts, or a steady workload? What level of application outage is acceptable? Business requirements drive these design considerations.

> **Tradeoff**: A complex solution can offer more features and flexibility, but it might affect the reliability of the workload because it requires more coordination, communication, and management of components. Alternatively, a simpler solution might not fully meet user expectations, or it might have a negative effect on scalability and extensibility as the workload evolves.

Collaborative design exercises

Work with stakeholders to:

- **Define and assign a criticality level to your workload's user flows and system flows**. Focus your design on critical flows to help you determine the required components and the best approach to achieve the required resiliency level.
- **Define functional and nonfunctional requirements**. Consider functional requirements to determine whether an application performs a task. Consider nonfunctional requirements to determine how well the application performs a task. Ensure that you understand nonfunctional requirements like scalability, availability, and latency. These requirements influence design decisions and technology choices.
- **Decompose workloads into components**. Prioritize simplicity, efficiency, and reliability in your design. Determine the components that you need to support your flows. Some components support multiple flows. Identify which challenge a component conceptually addresses, and consider removing a component from individual flows to simplify the overall design while still providing full functionality. For more information, see Recommendations for performing failure mode analysis.
- **Use failure mode analysis** to identify single points of failure and potential risks. Consider whether you need to account for unlikely situations, for example a geographic area that experiences a major natural disaster that affects all the availability zones in the region. It's expensive and involves significant tradeoffs to mitigate these uncommon risks. Clearly understand your business's tolerance for risk. For more information, see Recommendations for performing failure mode analysis.
- **Define availability and recovery targets** for your flows to inform your workload's architecture. Business metrics include service-level objectives (SLOs), service-level

agreements (SLAs), mean time to recover (MTTR), mean time between failure (MTBF), recovery time objectives (RTOs), and recovery point objectives (RPOs). Define target values for these metrics. This exercise might require compromise and mutual understanding between technology and business teams to ensure that each team's goals meet business objectives and are realistic. For more information, see Recommendations for defining reliability targets.

Additional design recommendations

You can perform the following recommendations without stakeholder engagement:

- **Strive for simplicity and clarity** in your design. Use the appropriate level of abstraction and granularity for your components and services. Avoid overengineering or under-engineering your solution. For example, if you break down your code into multiple small functions, it's hard to understand, test, and maintain.

- **Concede that all successful applications change over time**, whether to fix bugs, implement new features or technologies, or make existing systems more scalable and resilient.

- **Use platform as a service (PaaS) options** instead of infrastructure as a service (IaaS) when possible. IaaS is like having a box of parts. You can build anything, but you have to assemble it yourself. PaaS options are easier to configure and administer. You don't need to set up virtual machines (VMs) or virtual networks. You also don't have to perform maintenance tasks, such as installing patches and updates.

- **Use asynchronous messaging** to decouple the message producer from the consumer.

- **Abstract infrastructure away from domain logic**. Ensure that domain logic doesn't interfere with infrastructure-related functionality, such as messaging or persistence.

- **Offload cross-cutting concerns to a separate service**. Minimize the need to duplicate code across different functions, prefer reusing services with well-defined interfaces that can be easily consumed by different components. For example, if several services need to authenticate requests, you can move this functionality into its own service. Then you can evolve the authentication service. For example, you can add a new authentication flow without touching any of the services that use it.

- **Evaluate the suitability of common patterns and practices** for your needs. Avoid following trends or recommendations that might not be best for your context or

requirements. For example, microservices aren't the best option for every application because they can introduce complexity, overhead, and dependency issues.

Develop just enough code

The principles of simplicity, efficiency, and reliability also apply to your development practices. In a loosely coupled, componentized workload, determine the functionality that a component provides. Develop your flows to take advantage of that functionality. Consider these recommendations for your development practices:

- Use platform capabilities when they meet your business requirements. For example, to offload development and management, use low-code, no-code, or serverless solutions that your cloud provider offers.
- Use libraries and frameworks.
- Introduce pair programming or dedicated code review sessions as a development practice.
- Implement an approach to identify *dead code*. Be skeptical of the code that your automated tests don't cover.

Use the best data store for your data

In the past, many organizations stored all their data in large relational SQL databases. Relational databases provide atomic, consistent, isolated, and durable (ACID) guarantees for relational data transactions. But these databases come with disadvantages:

- Queries can require expensive joins.
- You need to normalize the data and restructure it for schema on write.
- Lock contention can affect performance.

Alternatives to relational databases

In a large solution, a single data store technology likely doesn't meet all your needs. Alternatives to relational databases include:

- Key-value stores
- Document databases

- Search engine databases
- Time series databases
- Column family databases
- Graph databases

Each option has pros and cons. Different data types are better suited for different data store types. Pick the storage technology that's the best fit for your data and how you use it.

For example, you might store a product catalog in a document database, such as Azure Cosmos DB, which supports a flexible schema. Each product description is a self-contained document. For queries over the entire catalog, you might index the catalog and store the index in Azure Cognitive Search. Product inventory might go into a SQL database because that data requires ACID guarantees.

Recommendations

- Consider other data stores. Relational databases aren't always appropriate. For more information, see Understand data store models.
- Remember that data includes more than just persisted application data. It also includes application logs, events, messages, and caches.
- Embrace polyglot persistence or solutions that use a combination of data store technologies.
- Consider the type of data that you have. For example, store:
  - Transactional data in a SQL database.
  - JSON documents in a document database.
  - Telemetry in a time series database.
  - Application logs in Azure Cognitive Search.
  - Blobs in Azure Blob Storage.
- Prioritize availability over consistency. The CAP theorem implies that you have to make tradeoffs between availability and consistency in a distributed system. You can't completely avoid network partitions, which is the other component of the CAP theorem. But you can adopt an eventual consistency model to achieve higher availability.

- Consider the skill set of your development team. There are advantages to using polyglot persistence, but it's possible to go overboard. It requires new skill sets to adopt a new data storage technology. To get the most out of the technology, your development team must:
    - Optimize queries.
    - Tune for performance.
    - Work with the appropriate usage patterns.

Consider these factors when you choose a storage technology:

- Use compensating transactions. With polyglot persistence, a single transaction might write data to multiple stores. If there's a failure, use compensating transactions to undo any steps that have finished.
- Consider bounded contexts, which is a domain-driven design concept. A bounded context is an explicit boundary around a domain model. A bounded context defines which parts of the domain that the model applies to. Ideally, a bounded context maps to a subdomain of the business domain. Consider polyglot persistence for bounded contexts in your system. For example, products might appear in the product catalog subdomain and the product inventory subdomain. But most likely, these two subdomains have different requirements for storing, updating, and querying products.

Azure facilitation

Azure offers the following services:

- Azure Functions is a serverless compute service that you can use to build orchestration with minimal code.
- Azure Logic Apps is a serverless workflow integration platform that you can use to build orchestration with a GUI or by editing a configuration file.
- Azure Event Grid is a highly scalable, fully managed publish-subscribe message distribution service that offers flexible message consumption patterns that use the MQTT and HTTP protocols. With Event Grid, you can build data pipelines with device data, build event-driven serverless architectures, and integrate applications.

For more information, see:

- Choose an Azure compute service

- Choose a compute option for microservices
- Review your data options

Example

For an example workload that determines components and their features based on requirements, see Reliable Web App pattern.

Related links

- Azure serverless
- Cloud-native applications
- Types of databases on Azure

Reliability checklist

Refer to the complete set of recommendations.

Reliability checklist

Recommendations for identifying and rating flows

Article • 01/25/2024

Applies to this Azure Well-Architected Framework Reliability checklist recommendation:

Expand table

RE:02	Identify and rate user and system flows. **Use a criticality scale based on your business requirements to prioritize the flows.**

This guide describes the recommendations for identifying and prioritizing workload flows. Identifying and prioritizing workload flows involves mapping user flows and system flows to determine their criticality to the organization. This practice ensures you identify and prioritize the most critical workload functionality to reduce the risk of damaging failures. Failure to identify and prioritize workload flows can lead to system breakdowns and compromised workload reliability.

Definitions

Expand table

Term	Definition
User flow	The paths or sequences of actions that users take within an application or system.
System flow	The flow of information and processes within a system. The system automatically follows this flow to enable user flows or workload functionality.

Key design strategies

When you design your workload, it's essential to define the user flows and system flows. User flows chart the movement of a user through your application. They focus on the user interface, interactions, decisions, and the steps required to complete a task. User flows provide a user-centric perspective on user experience and interface design. System flows chart the internal workings of your workload. They focus on data movement, input processing, output processing, and interactions among workload components, backend services, and external APIs. System flows indicate the intricate details of how the workload operates internally.

You should identify and define flows early in the design phase of your workload. It gives you a clearer understanding of what affects the reliability of your workload. It aligns your architectural decisions closely with the reliability goals of your workload.

Identify all user and system flows

The output of identifying all user and system flows is a catalog of all the flows in your workload. This identification process requires you to map out every user interaction and process within a system from beginning to end. This mapping is a prerequisite for identifying critical flows. Here are recommendations for identifying all user and system flows in a workload:

- *Interview stakeholders.* Stakeholders can provide valuable information to identify flows, and they can even help you map and prioritize flows. You can also interview users, business analysts, and technical teams to gather insights about user interactions and dependencies within the workload.

- *Review documentation.* In the design phase, you might not have documentation to review. However, if documentation exists, you should use it. Ask for system architecture diagrams, user manuals, and process descriptions. These documents can help you understand the intended functionality of the workload and its individual flows.

- *Observe the workload.* Monitor the workload in operation, noting how users interact with it and how different components speak with each other. You should analyze system logs, performance metrics, and user activity logs to identify patterns, frequent tasks, and system responses.

- *List identified flows.* The interviews, documentation, and observation should enable you to identify all the flows in the workload. Compile a list of all the flows you identify and categorize them into user flows (focusing on user interactions) and system flows (focusing on backend processes and data movement).

- *Define flow start and end points.* For each identified flow, clearly define where the flow starts and where it ends. For user flows, document each user interaction and its expected outcome. Focus on the user experience and interface design. For system flows, you need to identify its underlying triggers and expected outcomes.

- *Break down each flow.* Break down each flow into individual steps, describing the actions, decisions, or processes that occur at each point. Note how each step interacts with other parts of the system, including dependencies on other flows or external systems. You should be able to pinpoint how flows integrate with and

affect the workload and user experience. This dual approach provides a holistic view of your entire workload.

- *Document unique outputs.* Identify any alternative paths or exceptions within each flow, such as error handling or conditional branching. If a flow has multiple possible outcomes, you should add it to the catalog as distinct entries. For user flows, you should identify the intended behavior of the interaction. For system flows, you should identify the intended behavior of the process.

- *Visualize with diagrams.* Create flowcharts or diagrams to visually represent the flow and its steps. You can use tools like Microsoft Visio, UML sequence diagrams, use-case diagrams, simple drawing tools, or a descriptive list in text format (*see Example flow catalog*).

- *Update flow mapping iteratively.* Flow mapping is an iterative process. Flows can change, split, or combine, especially in the design phase. As the workload flows become more clearly defined, you should update the catalog of flows to match. Validate and refine your flow diagrams with feedback from stakeholders to ensure accuracy and completeness.

Identify business processes for each flow

Business processes are a series of tasks to achieve an output, such as order fulfillment, customer service management, or inventory control. The identification of business processes for each flow involves mapping flows to one or more business processes. This mapping helps you understand the importance of each flow to the business.

You might have existing documentation or business plans that provide a mapping of flows to business processes. Sometimes user manuals, training materials, or system specifications can provide insights into the intended use and purpose of the workload and its flows. If not, you need to map flows to the business processes they support. Here are recommendations to identify business processes for each flow:

- *Use workload outputs.* You can use the workload outputs and flow breakdown to correlate flows with the business processes they support. First, review the outputs the workload generates. The output could be sales reports, data files, or completed tasks.

- *Conduct interviews.* Speak with team members and stakeholders who interact with the workload. You should ask specific questions about their daily tasks, how they use the workload, and what objectives they achieve with it. Technical teams often have a deeper understanding the workload structure and can provide insights into the business processes it supports.

- *Monitor workload usage*. For existing workloads, monitor the workload and look for patterns in usage that indicate underlying business processes, such as data entry, order processing, or customer interaction.
- *Connect the output to a business process*. Connect the dots from the flow outputs to the overall business process they support. For example, if a flow step involves processing customer orders, then it directly supports the business process of order fulfillment. Order fulfillment contributes to the business objective of maintaining customer satisfaction and generating revenue. Finally, use the flow breakdown to help determine which flow created the sales report.

Identify process owners and stakeholders for each flow

The process owner for a flow is the individual that's responsible for the successful execution of a given process. They're responsible for that process and the flows that support it. You should identify the process owner for each workload flow. You should also identify the stakeholders for each flow. Stakeholders can be involved in the workload, have dependencies on a flow, or manage a dependency that the flow has.

You might have a responsibility assignment matrix (RAM) or RACI matrix that already identifies process owners and stakeholders. Typically, process owners are responsible or accountable for a process, and you consult or inform stakeholders.

Identify escalation paths for each flow

The identification of escalation paths is about determining channels for escalating issues related to a flow. Issues that need escalation could be urgent updates, security concerns, degradations, or technical incidents. The goal of identifying an escalation path is to ensure timely and effective resolution of issues.

The escalation path you map out should start with the person or group most likely to resolve a particular issue. If this person or group can't resolve the issue, the escalation path should identify the next point of contact. The next point of contact has broader responsibilities and is able to coordinate mitigation strategies with more parts of the organization. The number of people on an escalation path varies by flow and organization. Too many people on an escalation path can slow the resolution efforts.

Identify business impact of each flow

The identification of the business impact of each flow is essential for understanding how each flow contributes to key business objectives. Business impact could include revenue

generation, customer satisfaction, or operational efficiency. By understanding both the positive and negative impact of each flow, you can prioritize efforts to ensure the reliability of the flows that matter the most to your business. It's important to consider the direct impact of flow failure and its indirect effect on other interconnected processes. Here are steps to identify the business impact of each flow:

- *Identify positive impact*. Determine the expected benefits when a flow runs as intended. The expected benefits could include improved efficiency, increased revenue, enhanced customer satisfaction, or any other positive effect on the business.

- *Identify negative impact*. Assess the potential negative impacts if a process fails or doesn't work as expected. Consider quantifying specific losses, such as revenue drops. Include subjective effects like damage to reputation, erosion of customer trust, or adverse effects on other related business processes.

- *Define capacity and availability assumptions*. Establish assumptions about the expected capacity and availability of each process. Consider factors like throughput per unit of time, expected business hours, and target percentage uptime. If there are expectations for recovery time objective (RTO) or recovery point objective (RPO), you should include these expectations. These assumptions help in understanding reliability requirements of each flow.

By systematically evaluating these aspects, you can gain a comprehensive view of how each flow impacts the business and make strategic decisions about reliability optimization.

Assign a criticality rating to each flow

A detailed evaluation of flow importance relative to the overall business impacts allows you to assign a criticality rating to each flow. You can use quantitative or qualitative criticality ratings. The purpose is to sort the flows by priority and assign a label that allows you to identify the critical flows. This process is a logical continuation of identifying, mapping, and aligning with business processes and impact. Use the following criticality descriptions to assign your critical ratings:

- *High criticality*: High criticality flows are integral to core business functions. They directly affect critical aspects of a business such as customer experience, financial transactions, security protocols, human health, and safety. The failure or disruption of these flows could lead to significant immediate or long-term negative effects. Examples of negatives effects include loss of revenue, breach of trust, and legal

issues. Prioritizing these flows ensures that the most crucial aspects of the workload are robust and resilient.

- *Medium criticality*: Medium criticality flows are important for the complete functionality of the system but don't directly interface with the customer or critical business operations. For example, if an issue disrupts an internal data processing flow, you can retry the data processing without immediate external effects. These flows are essential for smooth operations but offer a buffer in terms of immediate customer or financial effect, allowing for managed responses to issues.

- *Low criticality*: Low criticality flows don't have a direct or significant effect on the core business functions or customer experience. Examples include ancillary processes like nightly log transfers or optional user features such as feedback surveys. While these flows contribute to the overall system, their disruption is unlikely to cause significant immediate business or operational issues.

By following this structured approach to assigning criticality, you can effectively prioritize resources and focus on maintaining and enhancing the reliability and effectiveness of your most critical flows.

>  **Tradeoff**: Higher expectations for reliability sometimes coincide with higher setup costs, operational costs, and management burden for operators. Ensure that stakeholders understand the potential cost increases of improving the reliability of critical flows.

Organizational alignment

Cloud Adoption Framework provides guidance for workloads that require business criticality classification.

For more information, see business criticality in cloud management.

Recommendations for performing failure mode analysis

Article • 10/08/2024

Applies to this Azure Well-Architected Framework Reliability checklist recommendation:

Expand table

RE:03	**Use failure mode analysis (FMA) to identify and prioritize potential failures in your solution components. Perform FMA to help you assess the risk and effect of each failure mode. Determine how the workload responds and recovers.**

This guide describes the best practices for performing failure mode analysis (FMA) for your workload. FMA is the practice of identifying potential points of failure within your workload and the associated flows and planning mitigation actions accordingly. At each step of the flow, you identify the blast radius of multiple failure types, which helps you design a new workload or refactor an existing workload to minimize the widespread effect of failures.

A key tenet of FMA is that failures happen no matter how many layers of resiliency you apply. More complex environments are exposed to more types of failures. Given this reality, FMA allows you to design your workload to withstand most types of failures and recover gracefully when a failure occurs.

If you skip FMA altogether or perform an incomplete analysis, your workload is at risk of unpredicted behavior and potential outages caused by suboptimal design.

Definitions

Expand table

Term	Definition
Failure mode	A type of problem that can cause one or more workload components to be degraded or severely affected to the point of being unavailable.
Mitigation	The activities that you have identified to address problems either proactively or reactively.
Detection	Your infrastructure, data, and app monitoring and alerting processes and procedures.

Key design strategies

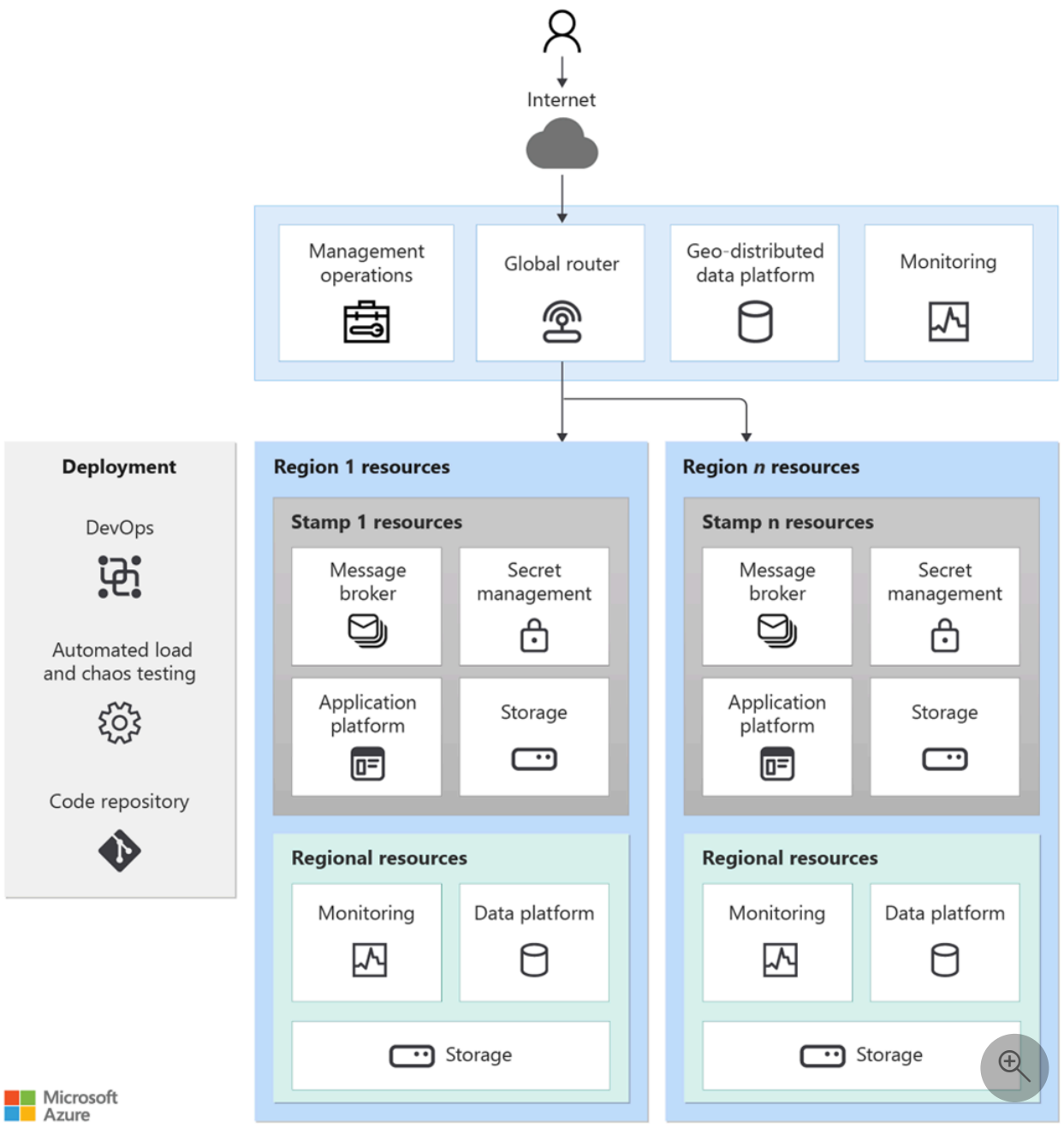
Review and implement the recommendations for identifying flows. It's assumed that you have identified and prioritized user and system flows based on criticality.

The data that you have gathered and the artifacts that you have created in your work provide you with a concrete description of your data paths involved throughout the flows. To be successful in your FMA work, accuracy and thoroughness in your artifacts is critical.

After you determine the critical flows, you can plan their required components. Next, follow each flow step by step to identify dependencies, including third-party services and potential points of failure, and plan your mitigation strategies.

Decompose the workload

As you move from ideation to design, you need to identify the component types that are required to support your workload. Your workload determines the necessary components that you must plan for. Typically, you need to plan for ingress control, networking, compute, data, storage, supporting services (like authentication, messaging, and secret or key management), and egress control. At this stage in your design work, you might not know the specific technologies that you'll deploy, so your design might look like the following example.



After you create your initial architecture design, you can overlay your flows to identify the discrete components that are used in those flows and create lists or workflow diagrams that describe the flows and their components. To understand the criticality of the components, use the criticality definitions that you have assigned to the flows. Consider the effect of a component malfunction on your flows.

Identify dependencies

Identify your workload dependencies to perform your single point-of-failure analysis. Decomposing your workload and overlaying flows provides insight into dependencies that are internal and external to the workload.

Internal dependencies are components in the workload scope that are required for the workload to function. Typical internal dependencies include APIs or secret/key

management solutions like Azure Key Vault. For these dependencies, capture the reliability data, like availability SLAs and scaling limits. External dependencies are required components outside the scope of the workload, such as another application or third-party service. Typical external dependencies include authentication solutions, like Microsoft Entra ID, and cloud connectivity solutions, like Azure ExpressRoute.

Identify and document the dependencies in your workload, and include them in your flow documentation artifacts.

Evaluate failure points

In your workload's critical flows, consider each component and determine how that component, and its dependencies, might be affected by a failure mode. Remember that there are many failure modes to consider when planning for resiliency and recovery. Any one component can be affected by more than one failure mode at any given time. These failure modes include:

- Regional outage. An entire Azure region is unavailable.
- Availability zone outage. An Azure availability zone is unavailable.
- Service outage. One or more Azure services are unavailable.
- Distributed denial-of-service (DDoS) or other malicious attack.
- App or component misconfiguration.
- Operator error.
- Planned maintenance outage.
- Component overload.

The analysis should always be in the context of the flow you're attempting to analyze, so be sure to document the effect on the user and expected result of that flow. For example, if you have an e-commerce application and you’re analyzing your customer flow, the effect of a particular failure mode on one or more components might be that all customers are unable to complete the checkout.

Consider the likelihood of each type of failure mode. Some are very unlikely, like multi-zone or multi-region outages, and adding mitigation planning beyond redundancy isn't a good use of resources and time.

Mitigation

Mitigation strategies fall into two broad categories: building more resiliency and designing for degraded performance.

Building more resiliency includes adding redundancy to your components, like infrastructure, data, and networking, and ensuring that your application design follows best practices for durability, for example breaking up monolithic applications into isolated apps and microservices. For more information, see Recommendations for redundancy and Recommendations for self-preservation.

To design for degraded performance, identify potential failure points that might disable one or more components of your flow but don't fully disable that flow. To maintain the functionality of the end-to-end flow, you might need to reroute one or more steps to other components or accept that a failed component runs a function, so the function is no longer available in the user experience. To return to the e-commerce application example, a failed component like a microservice might cause your recommendation engine to be unavailable, but the customers can still search for products and complete their transaction.

You also need to plan mitigation around dependencies. Strong dependencies play a critical role in application function and availability. If they're absent or experiencing a malfunction, there might be significant effect. The absence of weak dependencies might only affect specific features and not affect overall availability. This distinction reflects the cost to maintain the high availability relationship between the service and its dependencies. Classify dependencies as either strong or weak to help you identify which components are essential to the application.

If the application has strong dependencies that it can't operate without, the availability and recovery targets of these dependencies should align with the targets of the application itself. Minimize dependencies to achieve control over application reliability. For more information, see Minimize coordination between application services to achieve scalability.

If the application lifecycle is closely coupled with the lifecycle of its dependencies, the operational agility of the application might be limited, particularly for new releases.

Detection

Failure detection is essential to ensure that you have correctly identified failure points in your analysis and properly planned your mitigation strategies. Detection in this context means the monitoring of your infrastructure, data and application, and alerting when issues arise. Automate detection as much as possible, and build redundancy into your operations processes to ensure that alerts are always caught and are responded to

quickly enough to meet your business requirements. For more information, see the Recommendations for monitoring.

Outcome

For the outcome of your analysis, create a set of documents that effectively communicate your findings, the decisions that you have made relative to the flow components and mitigation, and the effect of the failure on your workload.

In your analysis, prioritize the failure modes and mitigation strategies that you have identified based on severity and likelihood. Use this prioritization to focus your documentation on those failure modes that are common and severe enough to warrant spending the time, effort, and resources on designing mitigation strategies around. For example, there might be some failure modes that are very rare in occurrence or detection. Designing mitigation strategies around them isn't worth the cost.

Refer to the following example table for a documentation starting point.

During your initial FMA exercise, the documents you produce will be mostly theoretical planning. The FMA documents should be reviewed and updated regularly to ensure that they stay up-to-date with your workload. Chaos testing and real-world experiences will help you refine your analyses over time.

Azure facilitation

Use Azure Monitor and Log Analytics to detect issues in your workload. For further insight into issues related to your infrastructure, apps, and databases, use tools like Application Insights, Container Insights, Network Insights, VM Insights, and SQL Insights.

Azure Chaos Studio is a managed service that uses chaos engineering to help you measure, understand, and improve your cloud application and service resilience.

For information about applying FMA principles to common Azure services, see Failure mode analysis for Azure applications.

Example

The following table shows an FMA example for an e-commerce website that's hosted on Azure App Service instances with Azure SQL databases and is fronted by Azure Front Door.

User flow: User sign in, product search, and shopping cart interaction

Expand table

Component	Risk	Likelihood	Effect/Mitigation/Note	Outage
Microsoft Entra ID	Service outage	Low	Full workload outage. Dependent on Microsoft to remediate.	Full
Microsoft Entra ID	Misconfiguration	Medium	Users unable to sign in. No downstream effect. Help desk reports configuration issue to identity team.	None
Azure Front Door	Service outage	Low	Full outage for external users. Dependent on Microsoft to remediate.	External only
Azure Front Door	Regional outage	Very low	Minimal effect. Azure Front Door is a global service, so global traffic routing directs traffic through non-effected Azure regions.	None
Azure Front Door	Misconfiguration	Medium	Misconfigurations should be caught during deployment. If these happen during a configuration update, administrators must roll back changes. Configuration update causes a brief external outage.	External only
Azure Front Door	DDoS attack	Medium	Potential for disruption. Microsoft manages DDoS (L3 and L4) protection and Azure Web Application Firewall blocks most threats. Potential risk of effect from L7 attacks.	Potential for partial outage
Azure SQL	Service outage	Low	Full workload outage. Dependent on Microsoft to remediate.	Full
Azure SQL	Regional outage	Very low	Auto-failover group fails over to secondary region. Potential outage during failover. Recovery time objectives (RTOs) and recovery point objectives (RPOs) to be determined during reliability testing.	Potential full
Azure SQL	Availability zone outage	Low	No effect	None

Component	Risk	Likelihood	Effect/Mitigation/Note	Outage
Azure SQL	Malicious attack (injection)	Medium	Minimal risk. All Azure SQL instances are virtual network-bound through private endpoints and network security groups (NSGs) add further intra-virtual network protection.	Low risk, potential for partial outage
App Service	Service outage	Low	Full workload outage. Dependent on Microsoft to remediate.	Full
App Service	Regional outage	Very low	Minimal effect. Latency for users in effected regions. Azure Front Door automatically routes traffic to non-effected regions.	None
App Service	Availability zone outage	Low	No effect. App services have been deployed as zone redundant. Without zone redundancy, there's a potential for effect.	None
App Service	DDoS attack	Medium	Minimal effect. Ingress traffic is protected by Azure Front Door and Azure Web Application Firewall.	None

Related links

- Failure mode analysis for Azure applications
- Resiliency and dependencies

Reliability checklist

Refer to the complete set of recommendations.

Reliability checklist

Feedback

Was this page helpful? Yes No

Recommendations for defining reliability targets

Article • 08/20/2024

Applies to this Azure Well-Architected Framework Reliability checklist recommendation:

Expand table

RE:04	**Define reliability and recovery targets for the components, the flows, and the overall solution. Visualize the targets to negotiate, gain consensus, set expectations, and drive actions to achieve the ideal state. Use the defined targets to build the health model. The health model defines what healthy, degraded, and unhealthy states look like.**

This guide describes the recommendations for defining availability and recovery target metrics for critical workloads and flows. You should derive reliability targets from workshop exercises with business stakeholders. Then refine those targets by monitoring and testing your workloads.

Set realistic expectations with your internal stakeholders about workload reliability. Then they can use contractual agreements to communicate those expectations to customers. Realistic expectations also help stakeholders understand and support your architectural design decisions and know that you're designing to optimally meet the targets that you agree on.

Consider using the following metrics to quantify your business requirements.

Expand table

Term	Definition
Service-level objective (SLO)	A measure of the performance and reliability of a workload or application. An SLO is a specific, measurable target that you set for particular customer interactions. It's a target that you set for your workload or application based on the quality of service that your customers expect to receive.
Service-level indicator (SLI)	A quantitative measurement of a particular aspect of a service's performance. You can use an SLI to measure your workload's compliance with an SLO.
Service-level agreement (SLA)	A contractual agreement between the service provider and the service customer. Failure to meet the agreement might have financial consequences for the service provider.

Term	Definition
Mean time to recover (MTTR)	The time taken to restore a component after a failure is detected.
Mean time between failure (MTBF)	The duration for which the workload can perform the expected function without interruption until it fails.
Recovery time objective (RTO)	The maximum acceptable time that an application can be unavailable after an incident.
Recovery point objective (RPO)	The maximum acceptable duration of data loss during an incident.

Key design strategies

Reliability targets represent the desired quality goal of a workload, as promised to its users and the business stakeholders. That goal includes both availability and recoverability of the workload. Keep in mind that reliability targets differ from performance targets, but you should include performance targets in reliability targets. Consider the following reliability targets:

- **Availability targets** define the quality standards for a system to remain accessible and functional. If it doesn't meet these standards, the system is considered unreliable. Use SLOs to help check whether your system meets these standards. Business and technical stakeholders collaborate to set realistic SLOs and consider factors like comparative analysis, user experience, and the workload profile.
- **Correctness targets** ensure that the workload properly performs its functions with consistent quality. To measure correctness, quantify the workload's indicators so that you can combine them into a unified, objective score.
- **Recovery targets** correspond to RTO, RPO, MTTR, and MTBF metrics, which quantify the effectiveness of your plans and testing for business continuity and disaster recovery.

To set reliability targets, business stakeholders define broad requirements. Then, technical experts assess the current state of the workload and work towards achieving and maintaining targets through monitoring and alerts. Both parties must agree on realistic targets.

Identify and score user and system flows based on their importance to your business requirements. Use these scores to guide the design, review, testing, and incident

management of your workload. Set reliability targets for these flows, and understand that failing to meet those targets can significantly affect your business.

>  **Tradeoff**: You might have a gap between the technical limits of your system and its business impact, like throughput versus transactions per second. Bridging this gap can be tough. Aim for a practical and cost-effective solution instead of overengineering.

Set availability objectives

The overall SLO of a workload reflects the **holistic quality, including all its dependencies.** A mature declaration of the SLO should indicate the overall business target for that workload, not just a composite of those dependencies. For example, if customers expect 99.99% availability, the overall SLO should aim for that goal, even if one part only achieves 99.80%.

Stakeholders evaluate the customer experience and consider how downtime affects revenue. They compare this loss to the cost of designing and operating the business flow. Decision makers then decide if they should spend more money on reliability to avoid revenue loss and maintain their reputation.

Workload owners **use financial goals to determine objectives.** Business requirements map to measurable metrics. The goal is to identify a set of factors that influence the quality of the customer experience.

Workload architects **make many technical decisions based on SLOs.** SLOs can:

- Serve as critical input into architectural decisions when you consider other dependencies.
- Provide a near real-time view and shared understanding of the health of a workload to enable objective discussions. They also help the workload team prioritize efforts to improve reliability and develop new features.
- Act as a primary signal for deployment operations, which drives automated rollback if problems occur and provides validation that the changes achieve the expected improvements to the customer experience.
- Speed up remediation and recovery by focusing on objectives, drive automated notification of problems to customers, and build trust between teams in your organization.

> **Important**
>
> You must know the difference between SLAs and SLOs. Although SLAs and SLOs might use or even present similar data, their intent is different. An SLA is a formal contract between an organization and its customers, and it has direct financial and legal implications if the organization fails to deliver on its promise. Organizations use SLOs to evaluate whether the potential downtime is within the tolerable limits.
>
> SLOs and SLAs share a business relationship and should be independently controlled. If the SLA serves as a business tactic, the organization might intentionally set it to a high value based on the business owner's goals. Conversely, SLOs can be higher. Consider mission-critical workloads as an example. This workload class can't afford long downtimes because the effects, including financial loss and even threats to human safety, are significant. Therefore, SLOs typically target 99.999% uptime, commonly referred to as the *five nines*. If SLOs don't meet those targets, organizations must react quickly to mitigate failures and prevent the outcomes of a failed SLA.
>
> The **example** in this article sets a high SLA to support business goals.
>
> Cloud platform and technology providers publish SLAs on their offerings. You should consider the SLAs as part of the SLO calculation, but you shouldn't use them as is without understanding the SLA's scope of coverage. For more information, see **Assess the impact of Microsoft SLAs**.

Consider common SLOs and influencing factors

Every SLO targets a specific quality criteria. Consider these common SLOs for reliability. This list isn't exhaustive. Add SLOs based on your business requirements.

- **Success rate** measures the success of requests and processes relative to those that return an error or fail their task.
- **Latency** measures the time between when a request for an operation is initiated and when the result is available or the process is complete.
- **Capacity** measures concurrent usage, such as the number of throttling-based responses.
- **Availability** measures uptime from the perspective of customers.
- **Throughput** measures the minimum data transfer rate during a certain amount of time. Throughput is measured as a data-rate unit, such as transactions per second

(TPS) or requests per second (RPS).

Understand the scenarios and tolerances for your workload on Azure. Both Azure services and application components affect the workload SLO. Combine responses from the following table to derive the overall SLO. Use these questions as examples to evaluate the workload component's utility.

Expand table

Component characteristics	Customer interaction	Other factors
- Does it expose **request or response APIs?** - Does it have **query APIs**? - Is it a **compute** component? - Is it a **job processing** component?	- **Control plane and management plane access** for public-facing Azure services. - **Data plane access**, for example create, read, update, and delete (CRUD) operations.	- Does your **release process** involve downtime? - What's the likelihood of **introducing bugs**? If the workload integrates with other systems, you might need to consider integration bugs. - How do **routine operations** like patching affect the availability target? Have you factored in partner dependencies? - Is your **staffing** sufficient to support constant emergency and emergency backup on-call rotation? - Does the application have **noisy neighbors** outside of your scope of control that can potentially cause disruptions?

Determine the SLO scope

You can set SLOs at various levels, such as for each application, workload, or a specific flow, in your system. Set level-specific SLOs so that you can customize SLOs based on each component's importance.

In software as a service (SaaS) solutions, measure SLOs per customer to optimize each customer's experience. Customers might have different infrastructure resources in their segments. For such cases, a system-wide SLO that aggregates all resources across customer segments might not make sense. Instead, measure SLOs that align with each customer's specific context. For more information, see Tenancy models for a multitenant solution.

Define composite SLO targets

SLOs must be **measurable** and **measured within an observability window**.

SLOs are often percentages like 99.90%, but they can also be statements. Use both methods to get a numerical value that includes all factors.

An SLO consists of measurable SLIs that define acceptable factors. SLIs are metrics with a set threshold that can be alerted. You can collect them from a platform or an application. Different components emit relevant SLIs. When you choose SLIs, consider factors that influence the SLO.

For example, to calculate the SLO for a flow that uses a response and request API, measure server latency and request-processing time. Throughput and error rates don't apply to continuous compute environments like virtual machines (VMs), scale sets, or Azure Batch.

For control plane access, consider error rates and latency for API responses and long-running operations like resource creation. Data plane access depends on the APIs used, each with their own SLO targets.

A good SLI shows when you might breach an SLO. It's usually measured in percentiles. Here are some commonly used percentiles and the estimated time of noncompliance to the expected availability.

Expand table

Objective	Noncompliance per week	Noncompliance per month	Noncompliance per year
99%	1.68 hours	7.20 hours	3.65 days
99.90%	10.10 minutes	43.20 minutes	8.76 hours
99.95%	5 minutes	21.60 minutes	4.38 hours
99.99%	1.01 minutes	4.32 minutes	52.56 minutes
99.999%	6 seconds	25.90 seconds	5.26 minutes

Important

A composite SLO value is a product distribution of the contributing factors.

An example composite SLO is 99.95% × 99.99999% = ~99.95%.

When you create composite SLOs for different flows, consider their varying criticality and relevance. Flows might have components that you deem as

noncritical and omit from your calculations. You can justify their omission based on whether their brief unavailability affects the customer's experience. In some cases, a component might not be relevant to the use case that you consider for the SLO. You can omit these components from the calculation, too.

The same principle applies to operations. Certain operations might introduce risks or affect the SLO, and others are insignificant. The decision should be explicit and built on consensus.

For an illustrative example of how to define and measure SLOs and SLIs, see the **Example** section.

Assess the impact of Microsoft SLAs

A Microsoft SLA provides insight into the availability of areas that Microsoft commits to. **SLAs don't guarantee an offering as a whole**. When you evaluate SLAs, have a good understanding of the coverage that's provided around the published percentile.

Consider Web Apps, a feature of Azure App Service. The feature is considered available when it returns a *200 OK* status in a given use case. Within that specific context and timeframe, it doesn't cover a financially backed guarantee on the availability of features like Easy Auth or slot switching. You should consider areas that aren't mentioned explicitly in the agreement as available by the platform's best effort.

So, if your workload relies on deployment slots, you can't derive your SLO solely from the App Service SLA. As a workload team, you need to hedge and predict the uptime availability. However, this prediction can be uncertain, which is why closely tying your SLO to the platform's SLA can be problematic.

Consider another example. If Azure Front Door has 99.99% availability, your design must adhere to specific criteria that's published in the agreement. For example, your back end must include storage, you need a `GET` operation that can retrieve a file of at least 50 KB in size, and you need to deploy agents across multiple places in at least five geographically diverse locations. This narrow use case of Azure Front Door doesn't guarantee features like caching, routing rules, or a web application firewall. These aspects fall outside the scope of the SLA.

Implement multiregion targets

From a reliability perspective, multiregion deployment is an implementation of the principle of redundancy. The goal is to mitigate the risk of a regional outage or

degraded performance. This strategy, when properly designed, can improve SLOs because it adds a secondary region for failover purposes.

There are two main use cases:

- High availability pattern, in which you distribute a load across regions for more capacity. High availability doesn't restrict workload users to a region, and the entire system's performance contributes to the SLO.
- Bulkhead pattern, in which you restrict customers to specific regions to segment them. In such cases, treat multiregion deployments as separate deployments, or *stamps*, in each region. Measure the health of each stamp separately, with the SLIs that are appropriate to your workload. Consider your overall workload's SLO based on the health of each stamp. If you can fail over between stamps, then your overall workload SLO is higher because a failure in one stamp is recoverable through a failover to another stamp.

>  **Tradeoff**: Determine whether the risk reduction is worth the added complexity. Multiregion targets also introduce operational complexities, such as coordinating deployments, ensuring data consistency, and handling latency. Those operations are significant during recovery. Your team should weigh these complexities against the increased resiliency.

Pay attention to how much redundancy you need to meet high SLOs. For example, Microsoft guarantees higher SLAs for multiregion deployments of Azure Cosmos DB than it guarantees for single-region deployments.

Define recovery metrics

Definitions for realistic recovery targets, like RTO, RPO, MTTR, and MTBF metrics, rely on your failure mode analysis and your plans and testing for business continuity and disaster recovery. When you define these targets, factor in the platform-provided recovery guarantees. Microsoft publishes RTO and RPO guarantees only for some products, like Azure SQL Database.

Before you finish this work, discuss aspirational targets with stakeholders, and ensure that your architecture design supports the recovery targets to the best of your understanding. Clearly communicate to stakeholders that any flows or entire workloads that aren't thoroughly tested for recovery metrics shouldn't have guaranteed SLAs. Make sure that stakeholders understand that recovery targets can change over time as workloads are updated. The workload can become more complex as you add customers

or as you adopt new technologies to improve the customer experience. These changes can increase or decrease your recovery metrics.

> **Note**
>
> MTBF can be challenging to define and guarantee. Platform as a service (PaaS) or SaaS models can fail and recover without any notification from the cloud provider, and the process can be completely transparent to you or your customers. If you define targets for this metric, cover only components that are under your control.

When you define recovery targets, define thresholds to initiate a recovery. For example, if a web node is unavailable for more than five minutes, automatically add a new node to the pool. Define thresholds for all components, and consider what the recovery for a specific component involves, including the effect on other components and dependencies. Your thresholds should also account for transient faults to ensure that you don't start recovery actions too quickly. Document and share with the stakeholders the potential risks, like data loss or session interruptions for customers, of recovery operations.

Monitor and visualize the targets

Use the data that you gather for your reliability targets to build your health model for each workload and the associated critical flows. A health model defines *healthy*, *degraded*, and *unhealthy* states for the flows and workloads. When the state changes, the model should alert the responsible parties. For detailed design considerations and recommendations, see Health modeling guidance.

To keep your operations teams and workload stakeholders informed, create a visualization that reflects the real-time status and overall trends of the workload health model. Discuss visualization solutions with the stakeholders to ensure that you deliver information that they value and that's easy to consume. They might also want to see generated reports weekly, monthly, or quarterly.

Azure facilitation

Azure SLAs provide the Microsoft commitments for uptime and connectivity. Different services have different SLAs, and sometimes products within a service have different SLAs. For more information, see SLAs for online services.

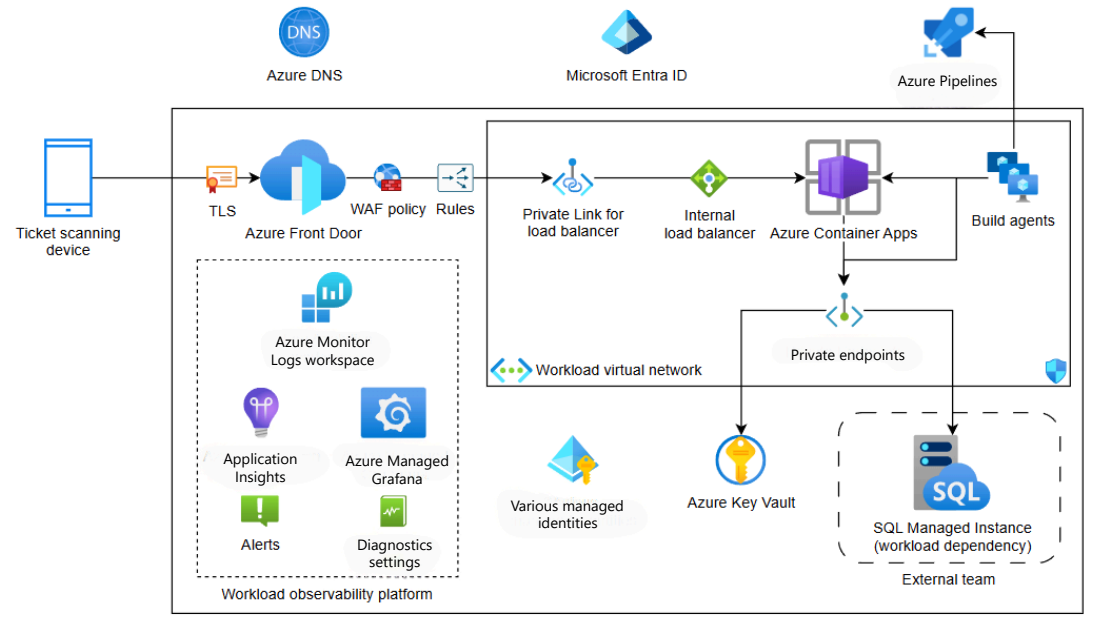
The Azure SLA includes procedures for obtaining a service credit if your workload doesn't meet the SLA, along with definitions of availability for each service. That aspect

of the SLA acts as an enforcement policy.

Explore the Azure Monitor dashboards for your visualization system.

Example

Contoso, Ltd. is designing a new mobile experience for their event ticketing system. Here's the high-level architecture.



Microsoft Azure

The Grafana logo is a trademark of its respective company. No endorsement is implied by the use of this mark.

Components

Here are some components that illustrate the concept of SLO definition. There are components in this architecture that aren't included in the following list. For example, even though Key Vault is part of the critical request flow, it isn't part of the response use case. If Key Vault is unavailable, the application continues to function by using secrets that are loaded during startup. However, if the application needs to scale, Key Vault availability becomes critical because new nodes need to be loaded with secrets. In this example, scaling operations aren't considered. Other components are omitted for brevity.

- **Azure Front Door** is the single point of entry that exposes an API that customers use to send requests.
- **Azure Container Apps** is the environment that the workload team owns and uses to run business logic for the application.
- **SQL Managed Instance** is owned and managed by another team and is a critical dependency of the workload.
- **Azure Private Link** provides private connectivity between Azure Front Door and Container Apps deployments. SQL Managed Instance is also exposed to the application through a private endpoint.

In this architecture, the API team defines an initial SLO target for critical flows in the application. They adopt the strategy that's described in Factors that influence SLOs. They aim to define objectives that cover the core functionality without overly emphasizing supplemental features. They measure the health of three critical user flows, which involve all core cloud functionalities and execute code across deployments. However, these flows don't cover all of the code or data access. Here are the influencing factors.

Calculate a composite SLO

- **Azure availability SLO:** The financial commitment SLA for Azure serves as a proxy to assess platform reliability.

Expand table

Azure component	Applicable SLA coverage	Not covered by SLA	Adjusted SLO
Azure Front Door	99.99% for successful HTTP `GET` operations	Caching, rules engine	99.98%
Container Apps	99.95% based on deployed apps that are reachable by the built-in ingress	Autoscaling, token store capabilities	99.95%
SQL Managed Instance	99.99% based on the connection to the SQL Server instance	Performance, data retention	99.80%
Private Link	99.99% based on whole minutes when the private endpoint network didn't accept traffic or when traffic didn't flow between the endpoint and the Private Link service	Individual failures lasting less than one minute	99.99%

The adjustment is based on several factors that depend on the workload team's promise to their objectives. One factor might be confidence in the platform's capability based on prior experience. For example, for Container Apps and Private Link, the team feels comfortable taking the SLA value as is.

There are also nuanced factors. For example, the team lowers the SLO value for SQL Managed Instance to 99.80% to account for potential failures in their data operations, such as schema changes and taking backups.

The team sets the composite SLO by calculating the impact of individual, adjusted SLO values. This value is 99.72%.

There are other contributing factors. The architecture relies on Azure networking components like virtual networks and network security groups (NSGs) that don't have a published SLA. The workload team decides to consider those factors with 99.99% availability for each component.

> A composite SLO based on predicted platform availability: 99.68% per month.

- **Application code SLO:** The team acknowledges that bugs in their application code or stored procedures can affect system availability, and they allocate one hour of monthly downtime to account for code-related errors.

  They use common downtime percentiles to estimate the SLO for individual factors like code defects, scaling problems, and other code-related considerations.

  > A composite SLO based on code and data availability: 99.86% per month.

- **Resource and application configuration SLO:** The team recognizes that cloud resources and application code must be properly configured. This target includes setting up autoscaling rules, deploying NSG rules, and selecting the correct size of SKUs. To account for configuration errors, they budget 10 minutes of monthly downtime, which is about 99.98% availability.

  > A composite SLO based on configuration availability: 99.95% per month.

- **Operations SLO:** The workload team develops good DevOps culture by following Well-Architected Framework principles for Operational Excellence. They deploy cloud resources, configuration, and code every sprint.

  The team considers deployments to be a risk because they can destabilize a running system. There might be errors as a result of TLS certificate updates, DNS changes, or tool errors. The team also considers potential downtime caused by

emergency fixes. They budget a total of 20 minutes of monthly downtime, which is approximately 99.95% availability.

Maintenance windows are designated time periods during which system maintenance or updates occur. The API is mostly unused for approximately four hours each day. To reduce the risk of unavailability, the team can schedule maintenance tasks during those less-active hours. This approach leads to a higher SLO, but the team decides not to include the maintenance window as part of their SLO.

> A composite SLO based on operations availability: 99.95% per month.

- **External dependencies SLO:** The team considers SQL Managed Instance as the primary dependency, which already has a 99.80% availability factored into the overall platform availability. No other external dependencies are considered.

  > A composite SLO based on external dependencies: Not applicable.

Determine the overall composite SLO result

The overall composite SLO target is set at 99.45%, which is equivalent to approximately four hours of downtime per month.

To meet the SLO target of only four hours of unavailability per month, the workload team establishes an on-call rotation. Both customer support and synthetic transaction monitoring can invoke on-call site reliability engineering (SRE) support to promptly start recovery steps to address SLO problems.

Set the workload SLA

The SLA for the workload is 99.90% availability per month.

The workload team's legal and finance departments set the SLA for the workload at 99.90% availability per month, which exceeds the SLO target of 99.45%. They make this decision after they analyze financial payouts versus projected customer growth based on an attractive SLA. The SLA covers two core user flows and includes performance considerations, not just availability. It's a calculated risk taken by the business team to benefit the business, and the engineering team is aware of the commitment.

Set the correctness SLO

The application's core user flows must be available and usably, or even competitively, responsive. The team sets a response time SLO specifically for the API, excluding client processing time and internet network traversal. They evaluate this SLO only during periods of availability. They choose the 75th percentile as both the SLO target and the performance measurement, which captures the typical customer experience and excludes worst-case scenarios.

Related links

Health modeling and observability of mission-critical workloads on Azure

Reliability checklist

Refer to the complete set of recommendations.

Reliability checklist

Feedback

Was this page helpful? Yes No

Recommendations for highly available multi-region design

Article • 01/15/2024

Applies to this Azure Well-Architected Framework Reliability checklist recommendation:

Expand table

RE:05	**Add redundancy at different levels, especially for critical flows. Apply redundancy to the compute, data, network, and other infrastructure tiers in accordance with the identified reliability targets.**

Related guides: Redundancy | Using availability zones and regions

This guide describes the recommendations for designing a highly available multi-region cloud environment. High availability is a foundational tenet of designing for reliability. A highly available architecture can help you avoid downtime as much as possible and recover efficiently if downtime does occur.

Active-active and *active-passive* are general architecture types that can be applied in different ways, depending on the platform you deploy your environment on. This guide focuses on a multi-region cloud environment design. On Azure, you can also design an active-active or active-passive architecture within a single region by using availability zones. For detailed guidance on designing a highly available architecture by using availability zones, see the Azure Well-Architected Framework guide.

Key design strategies

Active-active and active-passive are the two fundamental approaches to designing a highly available cloud environment. Active-active environments are designed to handle production loads in every region in which you deploy your workload. Active-passive environments are designed to handle production loads only in the primary region but fail over to the secondary (passive) region when necessary. Selecting the best Azure regions for your workload is a key part of designing a highly available multi-region environment. For guidance on selecting Azure regions, see the Select Azure Regions guide.

This section describes design options that you should consider when you evaluate each pattern and refine your architecture to meet your business requirements.

See Deployment Stamps pattern for guidance on architecting your workload in a repeatable, scalable way. This design pattern can help you optimize your high-availability design for efficient management.

The following sections describe the design options of the two patterns.

Active-active

- **Active-active at capacity**: Mirrored deployment stamps in two or more Azure regions, each configured to handle production workloads for the region or regions they serve and scalable to handle loads from other regions in case of a regional outage.
    - Networking: Use latency or weighted global routing to spread traffic among regions.
    - Data replication and consistency: Use a globally distributed data store like Azure Cosmos DB for multi-region read and write capabilities. For relational databases, use readable replicas with read-only connection strings.
    - Advantage of this design: Lower operating costs than an overprovisioned design.
    - Disadvantage of this design: Possible degradation of the user experience when scaling up to meet the demands of a full load if another region experiences an outage.
- **Active-active overprovisioned**: Mirrored deployment stamps in two or more Azure regions, each overprovisioned to handle production workloads for the region or regions they serve and to handle loads from other regions in case of a regional outage.
    - Networking: Use latency or weighted global routing to spread traffic among regions.
    - Data replication and consistency: Use a globally distributed data store like Azure Cosmos DB for multi-region read and write capabilities. For relational databases, use readable replicas with read-only connection strings.
    - Advantage of this design: The most resilient design possible.
    - Disadvantage of this design: Higher operating costs than a scalable design.
- Common advantages of both designs: High resiliency and low risk of full workload outage.

- Common disadvantages of both designs: Higher operating costs and management burden due to various factors, including the necessity of managing the synchronization of application state and data.

Active-passive

- **Warm spare**: One primary region and one or more secondary regions. The secondary region is deployed with the minimum possible compute and data sizing and runs without load. This region is known as a *warm spare* region. Upon failover, the compute and data resources are scaled to handle the load from the primary region.
    - Networking: Use priority global routing.
    - Data replication and consistency: Replicate your database to your passive region and use the automatic failover capabilities of platform as a service (PaaS) solutions like Azure Cosmos DB and Azure SQL Database.
    - Advantage of this design: Shortest recovery time among the active-passive designs.
    - Disadvantage of this design: Highest operating cost among the active-passive designs.
- **Cold spare**: One primary region and one or more secondary regions. The secondary region is scaled to handle full load, but all compute resources are stopped. This region is known as a *cold spare* region. You need to start the resources before failover.
    - Networking: Use priority global routing.
    - Data replication and consistency: Replicate your database to your passive region and use the automatic failover capabilities of PaaS solutions like Azure Cosmos DB and Azure SQL Database.
    - Advantage of this design: Lower operating costs than the warm spare design.
    - Disadvantage of this design: Longer recovery time than the warm spare design.
- **Redeploy on disaster**: One primary region and one or more secondary regions. Only the necessary networking is deployed in the secondary region. Operators must run provisioning scripts in the secondary region to fail over the workloads. This design is known as *redeploy on disaster*.
    - Networking: Use priority global routing.

    - Data replication and consistency: Deploy new database instances and rehydrate the data from backups.
    - Advantage of this design: Lowest operating costs.
    - Disadvantage of this design: Longest recovery time.
- Common advantages of active-passive designs: Lower operating costs and less day-to-day management burden than active-active designs. No need to synchronize application state.
- Common disadvantages of active-passive designs: Longer, more complex recovery process. Higher likelihood of needing manual intervention for a successful failover.

> **Note**
>
> Regardless of your high-availability design, remember to configure redundancy for supporting services like Azure DevOps infrastructure, jump boxes, monitoring, and any other critical service that's necessary to administer the workload.

Azure facilitation

- Azure Front Door combines the global routing functionality of Azure Traffic Manager with a content delivery system and web application firewall to help you manage your high-availability workload.
- Azure Cosmos DB is a globally distributed NoSQL database platform that can help you run an active-active environment and minimize the chance of downtime when a regional outage occurs.

Related links

- Multi-region N-tier application
- Multi-region load balancing

Reliability checklist

Refer to the complete set of recommendations.

Reliability checklist

Recommendations for designing for redundancy

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Reliability checklist recommendation:

RE:05	**Add redundancy at different levels, especially for critical flows. Apply redundancy to the compute, data, network, and other infrastructure tiers in accordance with the identified reliability targets.**

Related guides: Highly available multiregional design | Using availability zones and regions

This guide describes the recommendations for adding redundancy throughout critical flows at different workload layers, which optimizes resiliency. Meet the requirements of your defined reliability targets by applying the proper levels of redundancy to your compute, data, networking, and other infrastructure tiers. Apply this redundancy to give your workload a strong, reliable foundation to build on. When you build your workload without infrastructure redundancy, there's a high risk of extended downtime due to potential failures.

Definitions

Term	Definition
Redundancy	The implementation of multiple identical instances of a workload component.
Polyglot persistence	The concept of using different storage technologies by the same application or solution to take advantage of the best capabilities of each component.
Data consistency	The measure of how in sync or out of sync a given dataset is across multiple stores.
Partitioning	The process of physically dividing data into separate data stores.
Shard	A horizontal database partitioning strategy that supports multiple storage instances with a common schema. Data isn't replicated in all instances.

Key design strategies

In the context of reliability, use redundancy to contain problems that affect a single resource and ensure that those problems don't affect the reliability of the entire system.

Use the information that you identified about your critical flows and reliability targets to make informed decisions that are required for each flow's redundancy.

For example, you might have multiple web server nodes running at once. The criticality of the flow that they support might require that all of them have replicas that are ready to accept traffic if there's a problem that affects the entire pool, for example a regional outage. Alternatively, because large-scale problems are rare and it's costly to deploy an entire set of replicas, you might deploy a limited number of replicas so the flow operates in a degraded state until you resolve the problem.

When you design for redundancy in the context of performance efficiency, distribute the load across multiple redundant nodes to ensure that each node performs optimally. In the context of reliability, build in spare capacity to absorb failures or malfunctions that affect one or more nodes. Ensure that the spare capacity can absorb failures for the entire time that's needed to recover the affected nodes. With this distinction in mind, both strategies need to work together. If you spread traffic across two nodes for performance and they both run at 60 percent utilization and one node fails, your remaining node is at risk of becoming overwhelmed because it can't operate at 120 percent. Spread the load out with another node to ensure that your performance and reliability targets are upheld.

Redundant architecture design

Consider two approaches when you design a redundant architecture: active-active or active-passive. Choose your approach depending on the criticality of the user flow and system flow that the infrastructure components support. In terms of reliability, a multi-region active-active design helps you achieve the highest level of reliability possible, but it's significantly more expensive than an active-passive design. You can also use these design approaches for a single region by using availability zones. For more information, see Recommendations for highly available multi-region design.

Deployment stamps and units of scale

Whether you deploy in an active-active or active-passive model, follow the Deployment Stamps design pattern to ensure that you deploy your workload in a repeatable, scalable way. Deployment stamps are the groupings of resources that are required to deliver your workload to a given subset of your customers. For example, the subset might be a regional subset or a subset with all the same data privacy requirements as your workload. Think of each stamp as a *unit of scale* that you can duplicate to scale your workload horizontally or to perform blue-green deployments. Design your workload

with deployment stamps to optimize your active-active or active-passive implementation for resiliency and management burden.

Availability zones within Azure regions

Whether you deploy an active-active or an active-passive design, take advantage of availability zones within the active regions to fully optimize your resiliency. Many Azure regions provide multiple availability zones, which are separated groups of data centers within a region. Depending on the Azure service, you can take advantage of availability zones by deploying elements of your workload redundantly across zones or pinning elements to specific zones. For more information, see Recommendations for using availability zones and regions.

Infrastructure layer guidance

Compute resources

- Choose the appropriate compute service for your workload. Depending on the type of workload that you design, there might be several options available. Research the available services and understand which types of workloads work best on a given compute service. For example, SAP workloads are typically best suited for infrastructure as a service (IaaS) compute services. For a containerized application, determine the specific functionality you need to have control over to determine whether to use Azure Kubernetes Service (AKS) or a platform as a service (PaaS) solution. Your cloud platform fully manages a PaaS service.

- Use PaaS compute options if your requirements allow it. Azure fully manages PaaS services, which reduces your management burden, and a documented degree of redundancy is built in.

- Use Azure Virtual Machine Scale Sets if you need to deploy virtual machines (VMs). With Virtual Machine Scale Sets, you can automatically spread your compute evenly across availability zones.

- Keep your compute layer *clean of any state* because individual nodes that serve requests might be deleted, faulted, or replaced at any time.

- Use zone-redundant services where possible to provide higher resilience without increasing your operational burden.

- Overprovision critical resources to mitigate failures of redundant instances, even before autoscaling operations begin, so the system continues to operate after a

component failure. Calculate the acceptable effect of a fault when you incorporate overprovisioning into your redundancy design. As with your redundancy decision-making process, your reliability targets and financial tradeoff decisions determine the extent that you add spare capacity with overprovisioning. Overprovisioning specifically refers to *scaling out*, which means adding extra instances of a given compute resource type, rather than increasing the compute capabilities of any single instance. For example, if you change a VM from a lower-tier SKU to a higher-tier SKU.

- Deploy IaaS services manually or via automation in each availability zone or region in which you intend to implement your solution. Some PaaS services have built-in capabilities that are automatically replicated across availability zones and regions.

Data resources

- Determine whether synchronous or asynchronous data replication is necessary for your workload's functionality. To help you make this determination, see Recommendations for using availability zones and regions.
- Consider the growth rate of your data. For capacity planning, plan for data growth, data retention, and archiving to ensure your reliability requirements are met as the amount of data in your solution increases.
- Distribute data geographically, as supported by your service, to minimize the effect of geographically localized failures.
- Replicate data across geographical regions to provide resilience to regional outages and catastrophic failures.
- Automate failover after a database instance failure. You can configure automated failover in multiple Azure PaaS data services. Automated failover isn't required for data stores that support multi-region writes, like Azure Cosmos DB. For more information, see Recommendations for designing a disaster recovery strategy.
- Use the best data store for your data. Embrace polyglot persistence or solutions that use a mix of data store technologies. Data includes more than just persisted application data. It also includes application logs, events, messages, and caches.
- Consider data consistency requirements and use eventual consistency when requirements allow it. When data is distributed, use appropriate coordination to enforce strong consistency guarantees. Coordination can reduce your throughput and make your systems tightly coupled, which can make them more brittle. For

example, if an operation updates two databases, instead of putting it into a single transaction scope, it's better if the system can accommodate eventual consistency.

- Partition data for availability. Database partitioning improves scalability and it can also improve availability. If one shard goes down, the other shards are still reachable. A failure in one shard only disrupts a subset of the total transactions.
- If sharding isn't an option, you can use the Command and Query Responsibility Segregation (CQRS) pattern to separate your read-write and read-only data models. Add more redundant read-only database instances to provide more resilience.
- Understand the built-in replication and redundancy capabilities of the stateful platform services that you use. For specific redundancy capabilities of stateful data services, see Related links.

Networking

- Decide on a reliable and scalable network topology. Use a hub-and-spoke model or an Azure Virtual WAN model to help you organize your cloud infrastructure in logical patterns that make your redundancy design easier to build and scale.
- Select the appropriate network service to balance and redirect requests within or across regions. Use global or zone-redundant load balancing services when possible to meet your reliability targets.
- Ensure that you have allocated sufficient IP address space in your virtual networks and subnets to account for planned usage, including scale-out requirements.
- Ensure that the application can scale within the port limits of the chosen application hosting platform. If an application initiates several outbound TCP or UDP connections, it might exhaust all available ports and cause poor application performance.
- Choose SKUs and configure networking services that can meet your bandwidth and availability requirements. A VPN gateway's throughput varies based on their SKU. Support for zone redundancy is only available for some load balancer SKUs.
- Ensure that your design for handling DNS is built with a focus on resilience and supports redundant infrastructure.

Azure facilitation

The Azure platform helps you optimize the resiliency of your workload and add redundancy by:

- Providing built-in redundancy with many PaaS and software as a service (SaaS) solutions, some of which are configurable.
- Allowing you to design and implement intra-region redundancy by using availability zones and inter-region redundancy.
- Offering replica-aware load balancing services like Azure Application Gateway, Azure Front Door, and Azure Load Balancer.
- Offering easily implemented geo-replication solutions like active geo replication for Azure SQL Database. Implement global distribution and transparent replication by using Azure Cosmos DB. Azure Cosmos DB offers two options for handling conflicting writes. Choose the best option for your workload.
- Offering point-in-time restore capabilities for many PaaS data services.
- Mitigating port exhaustion via Azure NAT Gateway or Azure Firewall.

DNS-specific Azure facilitation

- For internal name resolution scenarios, use Azure DNS private zones, which have built-in zone redundancy and geo redundancy. For more information, see Azure DNS private zone resiliency.
- For external name resolution scenarios, use Azure DNS public zones, which have built-in zone redundancy and geo redundancy.
- The public and private Azure DNS services are global services that are resilient to regional outages because zone data is globally available.
- For hybrid name resolution scenarios between on-premises and cloud environments, use Azure DNS Private Resolver. This service supports zone redundancy if your workload is located in a region that supports availability zones. A zone-wide outage requires no action during zone recovery. The service automatically self-heals and rebalances to take advantage of the healthy zone. For more information, see Resiliency in Azure DNS Private Resolver.
- To eliminate a single point of failure and achieve a more resilient hybrid name resolution across regions, deploy two or more Azure DNS private resolvers across different regions. DNS failover, in a conditional forwarding scenario, is achieved by assigning a resolver as your primary DNS server. Assign the other resolver in a

different region as a secondary DNS server. For more information, see Set up DNS failover by using private resolvers.

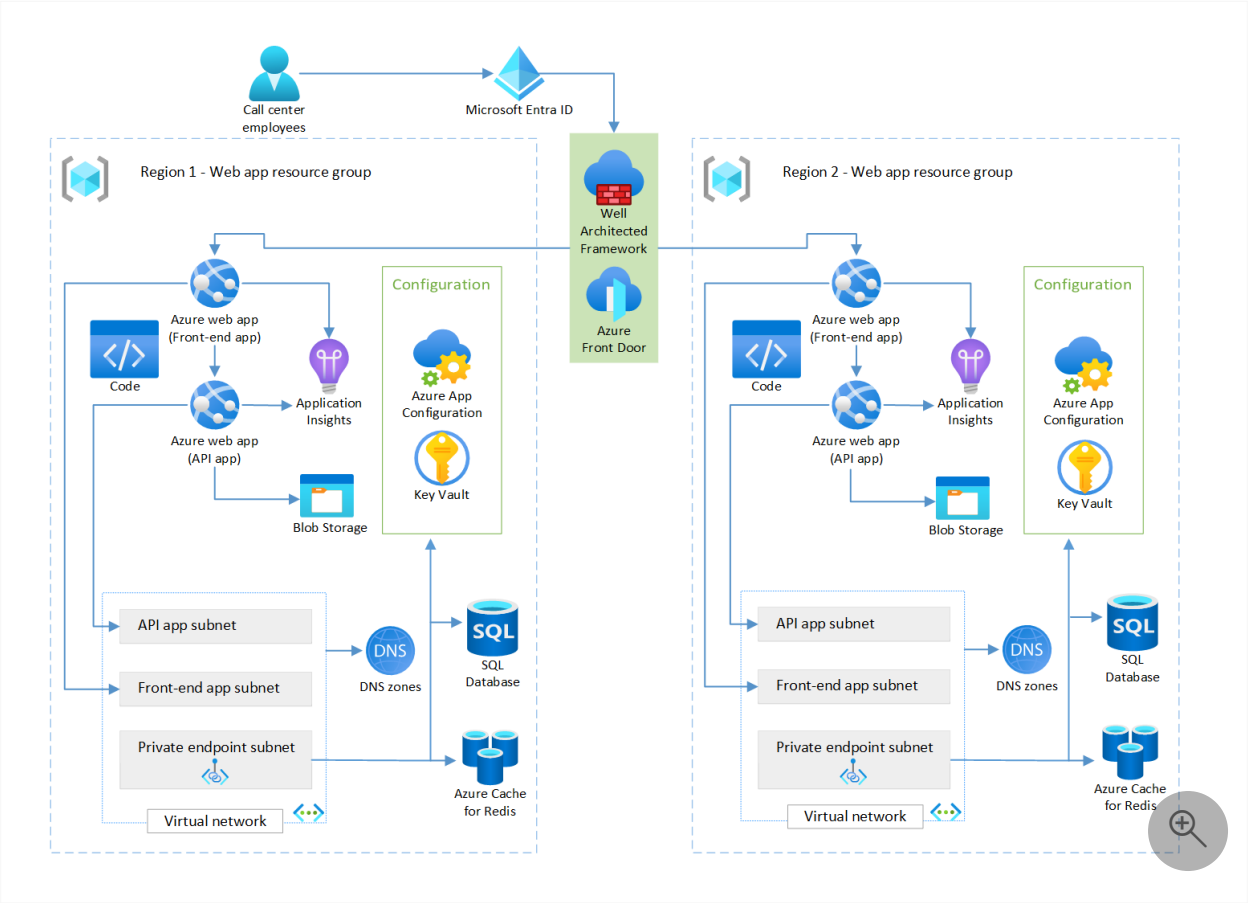
Tradeoffs

- More workload redundancy equates to more costs. Carefully consider adding redundancy and regularly review your architecture to ensure that you're managing costs, especially when you use overprovisioning. When you use overprovisioning as a resiliency strategy, balance it with a well-defined scaling strategy to minimize cost inefficiencies.
- There can be performance tradeoffs when you build in a high degree of redundancy. For example, resources that spread across availability zones or regions can affect performance because you have to send traffic over high-latency connections between redundant resources, like web servers or database instances.
- Different flows within the same workload might have different reliability requirements. Flow-specific redundancy designs can potentially introduce complexity into the overall design.

Example

For an example of a multi-region redundant deployment, see Baseline highly available zone-redundant web application.

The following diagram shows another example:



Related links

To learn more about stateful data service redundancy, see the following resources:

- Azure Storage redundancy
- Zone-redundant storage
- Azure SQL Database active geo-replication
- Configure replication between two managed instances

Reliability checklist

Refer to the complete set of recommendations.

Reliability checklist

Recommendations for using availability zones and regions

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Reliability checklist recommendation:

RE:05	**Add redundancy at different levels, especially for critical flows. Apply redundancy to the compute, data, network, and other infrastructure tiers in accordance with the identified reliability targets.**

Related guides: Highly available multiregional design | Redundancy

This guide describes the recommendations for determining when to deploy workloads across availability zones or regions.

When you design a solution for Azure, you need to decide whether you'll deploy across multiple availability zones in a region or deploy into multiple regions. This decision affects your solution's reliability, cost, and performance, and your team's ability to operate the solution. This guide provides information about the key business requirements that influence your decision, the approaches you can consider, the tradeoffs involved in each approach, and the effect of each approach on the core pillars of the Azure Well-Architected Framework.

Your choice of how you use regions and availability zones affects several of the pillars of the Well-Architected Framework:

- **Reliability**: Your choice of deployment approach can help you to mitigate various types of risks. In general, by spreading your workload across a more geographically distributed area, you can achieve higher resiliency.
- **Cost Optimization**: Some architectural approaches require deploying more resources than others, which can increase your resource costs. Other approaches involve sending data across geographically separated availability zones or regions, which might incur network traffic charges. It's also important to consider the ongoing cost of managing your resources, which is usually higher when you have a more complex architecture.
- **Performance Efficiency**: Most workloads aren't highly sensitive to network latency, but occasionally they can be. If latency is an issue, you need to physically locate the components close together to minimize latency when they communicate, which typically means deploying them into a single availability zone.

- **Operational Excellence**: A complex architecture takes more effort to deploy, configure, and manage. Additionally, for a highly available solution, you might need to plan how to fail over to a secondary set of resources. Failover, failback, and transparently redirecting your traffic can be complex, especially when manual steps are required.

However you design your solution, the Security pillar applies. Usually, decisions about whether and how you use availability zones and regions doesn't change your security posture. Azure applies the same security rigor to every region and availability zone.

> **Tip**
>
> For many production workloads, a **zone-redundant deployment** provides the best balance of tradeoffs. You can extend this approach with **asynchronous data backup to another region**. If you aren't sure which approach to select, start with this type of deployment.
>
> Consider other workload approaches when you need the specific benefits that those approaches provide, but be aware of the tradeoffs involved.

Definitions

Term	Definition
Active-active	An architecture in which multiple instances of a solution actively process requests at the same time.
Active-passive	An architecture in which one instance of a solution is designated as the *primary* and processes traffic, and one or more *secondary* instances are deployed to serve traffic if the primary is unavailable.
Asynchronous replication	A data replication approach in which data is written and committed to one location. At a later time, the changes are replicated to another location.
Availability zone	A separated group of datacenters within a region. Each availability zone is independent of the others, with its own power, cooling, and networking infrastructure. Many regions support availability zones.
Datacenter	A facility that contains servers, networking equipment, and other hardware to support Azure resources and workloads.
Locally redundant deployment	A deployment model in which a resource is deployed into a single region without reference to an availability zone. In a region that supports availability zones, the resource might be deployed in any of the region's availability zones.

Term	Definition
Multi-region deployment	A deployment model in which resources are deployed into multiple Azure regions.
Paired regions	A relationship between two Azure regions. Some Azure regions are connected to another defined region to enable specific types of multi-region solutions. Newer Azure regions aren't paired.
Region	A geographic perimeter that contains a set of datacenters.
Region architecture	The specific configuration of the Azure region, including the number of availability zones and whether the region is paired with another region.
Synchronous replication	A data replication approach in which data is written and committed to multiple locations. Each location must acknowledge completion of the write operation before the overall write operation is considered complete.
Zonal (pinned) deployment	A deployment model in which a resource is deployed into a specific availability zone.
Zone-redundant deployment	A deployment model in which a resource is deployed across multiple availability zones. Microsoft manages data synchronization, traffic distribution, and failover if a zone experiences an outage.

Key design strategies

Azure has a large global footprint. An Azure *region* is a geographic perimeter that contains a set of datacenters. You need to have a complete understanding of Azure regions and availability zones.

Azure regions have a variety of configurations, which are also called *region architectures*.

Many Azure regions provide *availability zones*, which are separated groups of datacenters. Within a region, availability zones are close enough to have low-latency connections to other availability zones, but they're far enough apart to reduce the likelihood that more than one will be affected by local outages or weather. Availability zones have independent power, cooling, and networking infrastructure. They're designed so that if one zone experiences an outage, then regional services, capacity, and high availability are supported by the remaining zones.

The following diagram shows several example Azure regions. Regions 1 and 2 support availability zones.



If you deploy into an Azure region that contains availability zones, you can use multiple availability zones together. By using multiple availability zones, you can keep separate copies of your application and data within separate physical datacenters in a large metropolitan area.

There are two ways to use availability zones in a solution:

- **Zonal** resources are pinned to a specific availability zone. You can combine multiple zonal deployments across different zones to meet high reliability requirements. You're responsible for managing data replication and distributing requests across zones. If an outage occurs in a single availability zone, you're responsible for failover to another availability zone.
- **Zone-redundant** resources are spread across multiple availability zones. Microsoft manages spreading requests across zones and the replication of data across zones. If an outage occurs in a single availability zone, Microsoft manages failover automatically.

Azure services support one or both of these approaches. Platform as a service (PaaS) services typically support zone-redundant deployments. Infrastructure as a service (IaaS) services typically support zonal deployments. For more information about how Azure services work with availability zones, see Azure services with availability zone support.

Microsoft aims to deploy updates to Azure services to a single availability zone at a time. This approach reduces the impact that updates might have on an active workload, because the workload can continue to run in other zones while the update is in process. For more information about how Azure deploys updates, see Advancing safe deployment practices.

Many regions also have a *paired region*. Paired regions support certain types of multi-region deployment approaches. Some newer regions have multiple availability zones and don't have a paired region. You can still deploy multi-region solutions into these regions, but the approaches you use might be different.

For more information about how Azure uses regions and availability zones, see What are Azure regions and availability zones?

Understand shared responsibilities

The shared responsibility principle describes how responsibilities are divided between the cloud provider (Microsoft) and you. Depending on the type of services you use, you might take on more or less responsibility for operating the service.

Microsoft provides availability zones and regions to give you flexibility in how you design your solution to meet your requirements. When you use managed services, Microsoft takes on more of the management responsibilities for your resources, which might even include data replication, failover, failback, and other tasks related to operating a distributed system.

Regardless of the approach you use, your own code needs to follow recommended practices for handling transient failures. These practices are even more important in a multi-zone or multi-region solution, because failover between zones or regions usually requires that your application retry connections to services.

Identify key business and workload requirements

To make an informed decision about how to use availability zones and regions in your solution, you need to understand your requirements. These requirements should be driven by discussions between solution designers and business stakeholders.

Risk tolerance

Different organizations have different degrees of risk tolerance. Even within an organization, risk tolerance is often different for each workload. Most workloads don't need extremely high availability. However, some workloads are so important that it's even worth mitigating risks that are unlikely to occur, like major natural disasters that affect a wide geographic area.

This table lists a few of the common risks that you should consider in a cloud environment:

Risk	Examples	Likelihood
Hardware outage	Problem with hard disk or networking equipment.	High. Any resiliency strategy should account for these risks.

Risk	Examples	Likelihood
	Host reboots.	
Datacenter outage	Power, cooling, or network failure across an entire datacenter. Natural disaster in one part of a metropolitan area.	Medium
Region outage	Major natural disaster that affects a wide geographical area. Network or service problem that makes one or more Azure services unavailable in an entire region.	Low

It would be ideal to mitigate every possible risk for every workload, but it's not practical or cost effective to do so. It's important to have an open discussion with business stakeholders so you can make informed decisions about the risks that you should mitigate.

> **Tip**
>
> Generally, it's only worth mitigating the low-likelihood risks for mission-critical workloads. For example, banks, governments, and healthcare workloads often need to remain operational in all situations. For other workloads, the organization's risk tolerance is usually higher.

Resiliency requirements

It's important to understand the resiliency requirements for your workload, including the recovery time objective (RTO) and recovery point objective (RPO). These objectives help you decide which approaches to rule out. If you don't have clear requirements, you can't make an informed decision about which approach to follow. For more information, see Target functional and nonfunctional requirements.

Service-level agreements

You should understand your solution's expected uptime service-level agreement (SLA). For example, you might have a business requirement that your solution needs to meet 99.9% uptime.

Azure provides SLAs for each service. An SLA indicates the level of uptime you should expect from the service and the conditions you need to meet to achieve that expected SLA.

Your architectural decisions affect your solution's composite SLA. In general, the more redundancy you build into your solution, the higher your SLA is likely to be. Many Azure services provide higher SLAs when you deploy services in a zone-redundant or multi-region configuration. Review the SLAs for each of the Azure services you use to ensure that you understand how to maximize the resiliency and SLA of the service.

Data residency

Some organizations place restrictions on the physical locations into which their data can be stored and processed. Sometimes these requirements are based on legal or regulatory standards. In other situations, an organization might decide to adopt a data residency policy to avoid customer concerns. If you have strict data residency requirements, you might need to use a single-region deployment or use a subset of Azure regions and services.

> **Note**
>
> Avoid making unfounded assumptions about your data residency requirements. If you need to comply with specific regulatory standards, verify what those standards actually specify.

User location

If your users are geographically dispersed, it might make sense to deploy your workload across multiple regions. If your users are in one area, a single-region deployment can simplify your operational requirements and reduce your costs.

Even if your users are in different geographical areas, you might not need a multi-region deployment. Consider whether you can achieve your requirements within a single region by using global traffic acceleration, like the acceleration provided by Azure Front Door.

Budget

If you operate under a constrained budget, it's important to consider the costs involved in deploying redundant workload components. These costs can include additional resource charges, networking costs, and the operational costs of managing more resources and a more complex environment.

Complexity

It's a good practice to avoid unnecessary complexity in your solution architecture. The more complexity you introduce, the harder it becomes to make decisions about your architecture. Complex architectures are harder to operate, harder to secure, and often less performant.

Azure facilitation

By providing regions and availability zones, Azure enables you to select a deployment approach that fits your needs. There are many approaches that you can choose from, each of which provides benefits and incurs costs.

To illustrate the deployment approaches that you can use, consider an example scenario. Suppose you're thinking about deploying a new solution that includes an application that writes data to some sort of storage:



This example isn't specific to any particular Azure services. It's intended to provide a simple example for illustrating fundamental concepts.

There are multiple ways to deploy this solution. Each approach provides a different set of benefits and incurs different costs. At a high level, you can consider a *locally redundant*, *zonal (pinned)*, *zone-redundant*, or *multi-region* deployment. This table summarizes some of the pillar concerns:

Pillar	**Locally redundant**	**Zonal (pinned)**	**Zone-redundant**	**Multi-region**
Reliability	Low reliability	Depends on approach	High or very high reliability	High or very high reliability
Cost Optimization	Low cost	Depends on approach	Moderate cost	High cost
Performance Efficiency	Acceptable performance (for most workloads)	High performance	Acceptable performance (for most workloads)	Depends on approach
Operational Excellence	Low operational requirements	High operational requirements	Low operational requirements	High operational requirements

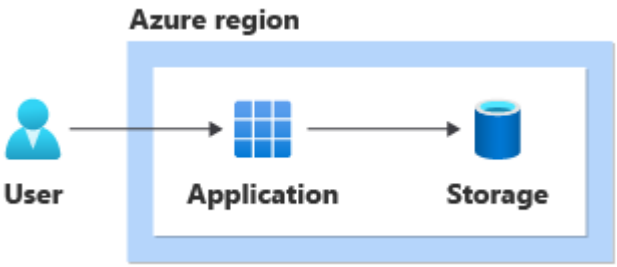
This table summarizes some of the approaches you can use and how they affect your architecture:

Architectural concern	Locally redundant	Zonal (pinned)	Zone-redundant	Multi-region
Compliance with data residency	High	High	High	Depends on region
Regional availability	All regions	Regions with availability zones	Regions with availability zones	Depends on region

The rest of this article describes each of the approaches listed in the preceding table. The list of approaches isn't exhaustive. You might decide to combine multiple approaches or use different approaches in different parts of your solution.

Deployment approach 1: Locally redundant deployments

If you don't specify multiple availability zones or regions when you deploy your resources, Azure doesn't make any guarantees about whether the resources are deployed into a single datacenter or split across multiple datacenters in the region. In some situations, Azure might also move your resource between availability zones.



Most Azure resources are highly available by default, with high SLAs and built-in redundancy within a datacenter that's managed by the platform. However, from a reliability perspective, if any part of the region experiences an outage, there's a chance that your workload might be affected. If it is, your solution might be unavailable, or your data could be lost.

For highly latency-sensitive workloads, this approach might also result in lower performance. Your workload components might not be colocated in the same datacenter, so you might observe some latency for intra-region traffic. Azure might also transparently move your service instances between availability zones, which might slightly affect performance. However, this isn't a concern for most workloads.

This table summarizes some of the pillar concerns:

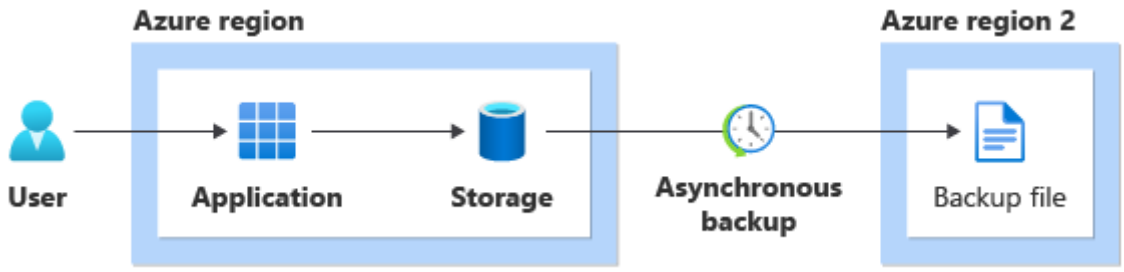
Pillar	Impact
Reliability	**Low reliability.** Services are subject to outages if a datacenter fails. You can, however, build an application to be resilient to other types of failures.
Cost Optimization	**Lowest cost.** You only need to have a single instance of each resource, and you don't incur any inter-zone or inter-region bandwidth costs.
Performance Efficiency	*For most workloads:* **Acceptable performance.** This approach is likely to provide satisfactory performance. *For highly latency-sensitive workloads:* **Low performance.** Components aren't guaranteed to be located in the same availability zone, so you might see lower performance for highly latency-sensitive components.
Operational Excellence	**High operational efficiency.** You only need to deploy and manage a single instance of each resource.

This table summarizes some of the concerns from an architectural perspective:

Architectural concern	Impact
Compliance with data residency	**High.** When you deploy a solution that uses this approach, data is stored in the Azure region that you select.
Regional availability	**High.** This approach can be used in any Azure region.

Locally redundant deployments with backup across regions

You can extend a locally redundant deployment by performing regular backups of your infrastructure and data to a secondary region. This approach adds an extra layer of protection to mitigate against an outage in your primary region. Here's what it looks like:



When you implement this approach, you need to carefully consider your RTO and RPO:

- **Recovery time**: If a regional outage occurs, you might need to rebuild your solution in another Azure region, which affects your recovery time. Consider building your solution by using an infrastructure-as-code (IaC) approach so that

you can quickly redeploy into another region if a major disaster occurs. Ensure that your deployment tools and processes are just as resilient as your applications so that you can use them to redeploy your solution even if there's an outage. Plan for and rehearse the steps that are required to restore your solution back to a working state.

- **Recovery point**: Your backup frequency determines the amount of data loss that you might experience (your recovery point). You can typically control the frequency of backups so that you can meet your RPO.

This table summarizes some of the pillar concerns:

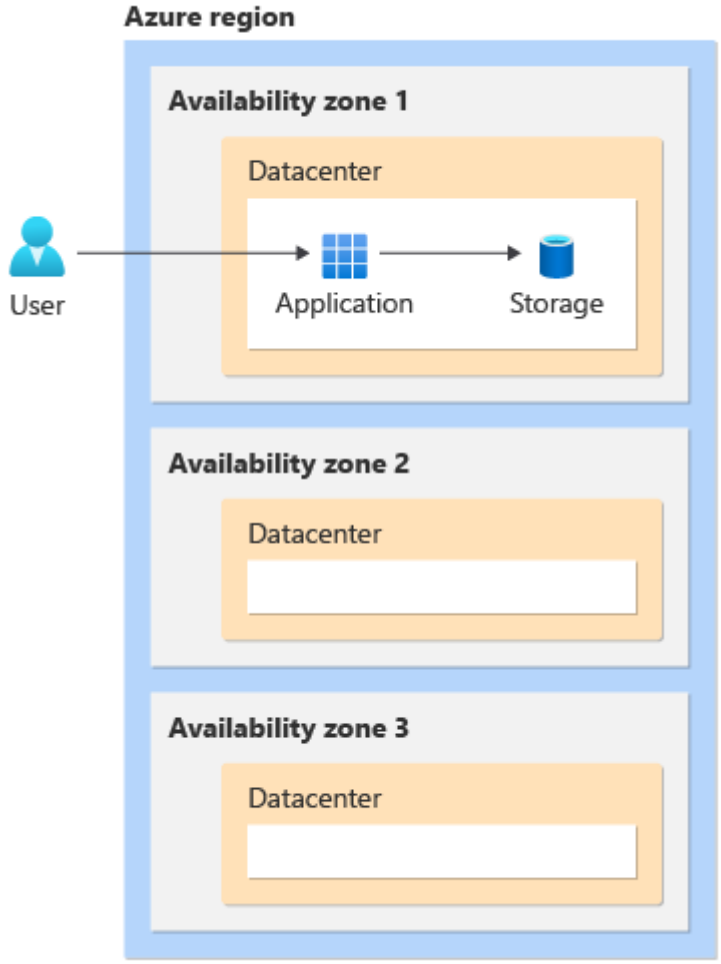
Pillar	Impact
Reliability	**Moderate reliability.** Services are subject to outages if a datacenter fails. Data is backed up asynchronously to a geographically separated region, which reduces the effect of a full region outage by minimizing data loss. In a full region outage, you can manually restore operations into another region. However, recovery processes can be complex, and it can take time to manually restore into the other region.
Cost Optimization	**Low cost.** Typically, adding a backup to another region costs only slightly more than deploying a locally redundant resource.
Performance Efficiency	*For most workloads:* **Acceptable performance.** This approach is likely to provide satisfactory performance. *For highly latency-sensitive workloads:* **Low performance.** Components aren't guaranteed to be located in the same availability zone, so you might see lower performance for highly latency-sensitive components.
Operational Excellence	*During any outage within a region:* **Low operational efficiency.** Failover is your responsibility and might require manual operations and redeployments.

This table summarizes some of the concerns from an architectural perspective:

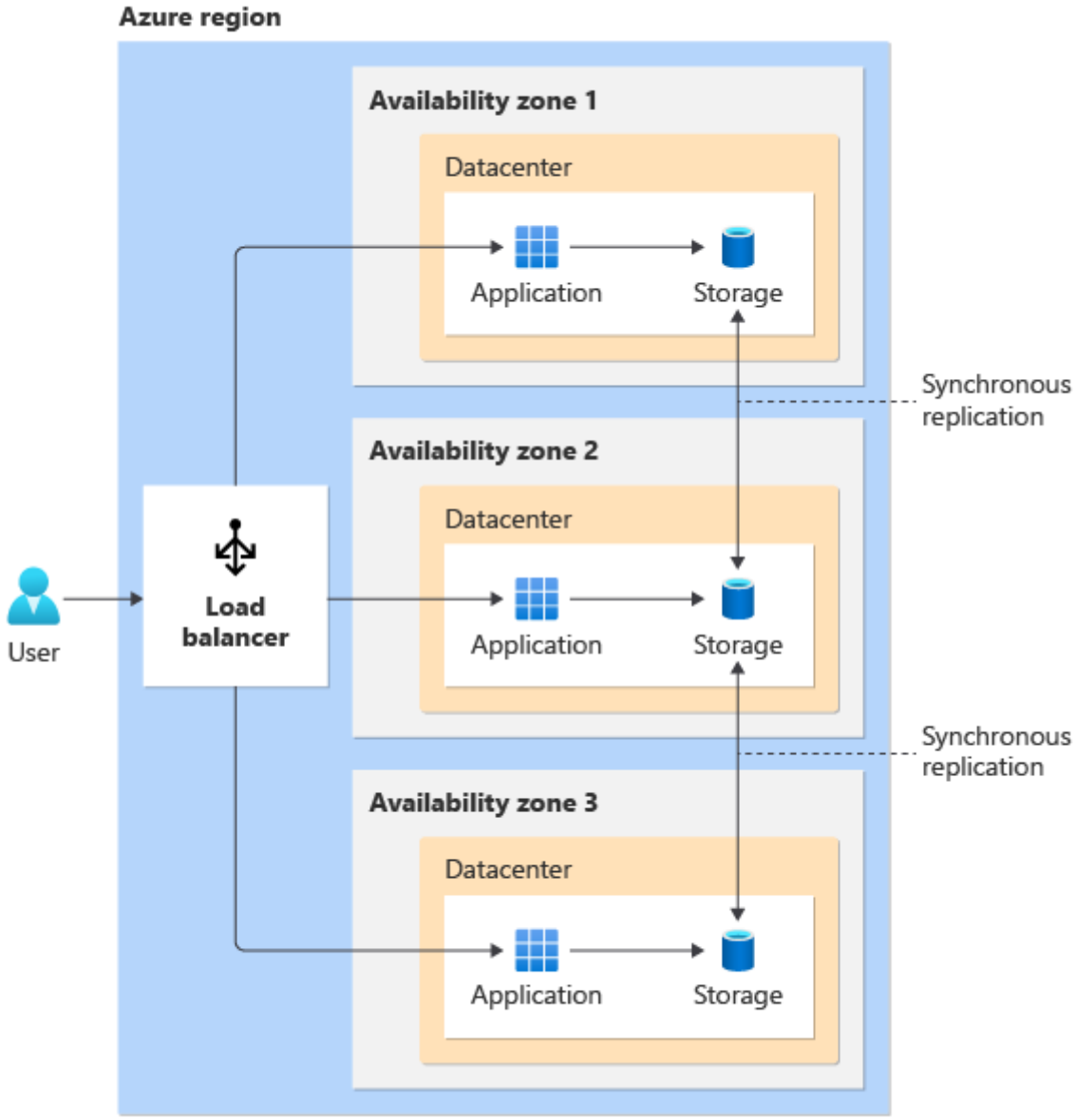
Architectural concern	Impact
Compliance with data residency	**Depends on region selection.** Data is primarily stored in the Azure region that you specify. However, you need to select another region to store your backups, so it's important that you select a region that's compatible with your data residency requirements.
Regional availability	**High.** You can use this approach in any Azure region.

Deployment approach 2: Zonal (pinned) deployments

In a *zonal* deployment, you specify that your resources should be deployed to a specific availability zone. This approach is sometimes called a *zone-pinned* deployment.



A zonal approach reduces the latency between your components. However, by itself, it doesn't increase the resiliency of your solution. To increase your resiliency, you need to deploy multiple instances of your components into multiple availability zones and decide how to route traffic between each instance. This example shows an *active-passive* traffic routing approach:



In the previous example, a load balancer is deployed across multiple availability zones. It's important to consider how you route traffic between instances in different availability zones, because a zone outage might also affect the networking resources deployed into that zone. You might also consider using a global load balancer, like Azure Front Door or Azure Traffic Manager, which isn't deployed into a region at all.

When you use a zonal deployment model, you take on many responsibilities:

- You need to deploy resources to each availability zone, and configure and manage those resources individually.
- You need to decide how and when to replicate data between the availability zones, and then configure and manage the replication.
- You're responsible for distributing the requests to the correct resources, by using, for example, a load balancer. You need to ensure that the load balancer meets your resiliency requirements. You also need to decide whether to use an active-passive or an active-active request distribution model.

- If an availability zone experiences an outage, you need to handle the failover to send traffic to resources in another availability zone.

An active-passive deployment across multiple availability zones is sometimes called *in-region DR* or *Metro DR*.

This table summarizes some of the pillar concerns:

Pillar	Impact
Reliability	*When deployed in a single availability zone:* **Low reliability.** A zonal deployment doesn't provide any resiliency to an outage in a datacenter or availability zone. You must deploy redundant resources across multiple availability zones to achieve high resiliency. *When deployed in multiple availability zones:* **High reliability.** Services can be made resilient to a datacenter or availability zone outage.
Cost Optimization	*When deployed in a single availability zone:* **Low cost.** A single-zone deployment requires only a single instance of each resource. *When deployed in multiple availability zones:* **High cost.** You deploy multiple instances of the resources, each of which are billed separately. You also need to pay for inter-zone traffic for data replication.
Performance Efficiency	**High performance.** Latency can be very low when the components that serve a request are located in the same availability zone.
Operational Excellence	**Low operational efficiency.** You need to configure and manage multiple instances of your service. You need to replicate data between availability zones. During an availability zone outage, failover is your responsibility.

This table summarizes some of the concerns from an architectural perspective:

Architectural concern	Impact
Compliance with data residency	**High.** When you deploy a solution that uses this approach, data is stored in the Azure region that you select.
Regional availability	**Regions with availability zones.** This approach is available in any region that supports availability zones.

This approach is typically used for workloads that are based on virtual machines. For a complete list of services that support zonal deployments, see Availability zone service and regional support.

When you plan a zonal deployment, verify that the Azure services you use are supported in the availability zones you plan to use. For example, to list which virtual machine SKUs

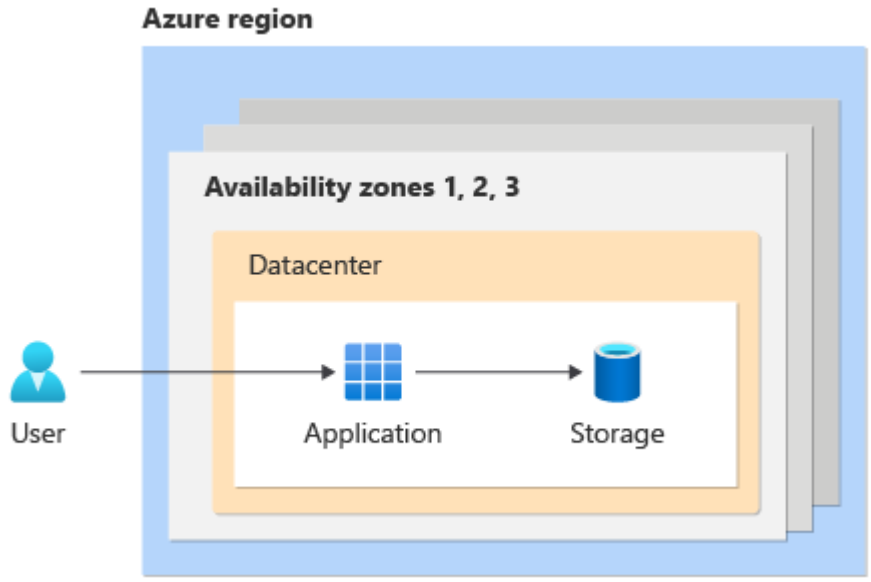
are available in each availability zone, see Check VM SKU availability.

> **Tip**
>
> When you deploy a resource into a specific availability zone, you select the zone number. The sequence of zone numbers is different for each Azure subscription. If you deploy resources across multiple Azure subscriptions, verify the zone numbers that you should use in each subscription. For more information, see **What are Azure regions and availability zones?**.

Deployment approach 3: Zone-redundant deployments

When you use this approach, your application tier is spread across multiple availability zones. When requests arrive, a load balancer that's built into the service (which itself spans availability zones) disperses the requests across the instances in each availability zone. If an availability zone experiences an outage, the load balancer directs traffic to instances in the healthy availability zones.

Your storage tier is also spread across multiple availability zones. Copies of your application's data are distributed across multiple availability zones via *synchronous replication*. When the application makes a change to data, the storage service writes the change to multiple availability zones, and the transaction is considered complete only when all of these changes are complete. This process ensures that each availability zone always has an up-to-date copy of the data. If an availability zone experiences an outage, another availability zone can be used to access the same data.



A zone-redundant approach increases your solution's resiliency to issues like datacenter outages. Because data is replicated synchronously, however, your application has to wait

for the data to be written across multiple separate locations that might be in different parts of a metropolitan area. For most applications, the latency involved in inter-zone communication is negligible. However, for some highly latency-sensitive workloads, synchronous replication across availability zones might affect the application's performance.

This table summarizes some of the pillar concerns:

Pillar	Impact
Reliability	**High reliability.** Services are resilient to an outage of a datacenter or availability zone. Data is synchronously replicated across availability zones and with no delay.
Cost Optimization	**Moderate cost.** Depending on the services you use, you might incur some costs for higher service tiers to enable zone redundancy, or some inter-zone networking costs.
Performance Efficiency	*For most workloads:* **Acceptable performance.** This approach is likely to provide satisfactory performance. *For highly latency-sensitive workloads:* **Low performance.** Some components might be sensitive to latency due to inter-zone traffic or data replication time.
Operational Excellence	**High operational efficiency.** You typically need to manage only a single logical instance of each resource. For most services, during an availability zone outage, failover is the responsibility of Microsoft and happens automatically.

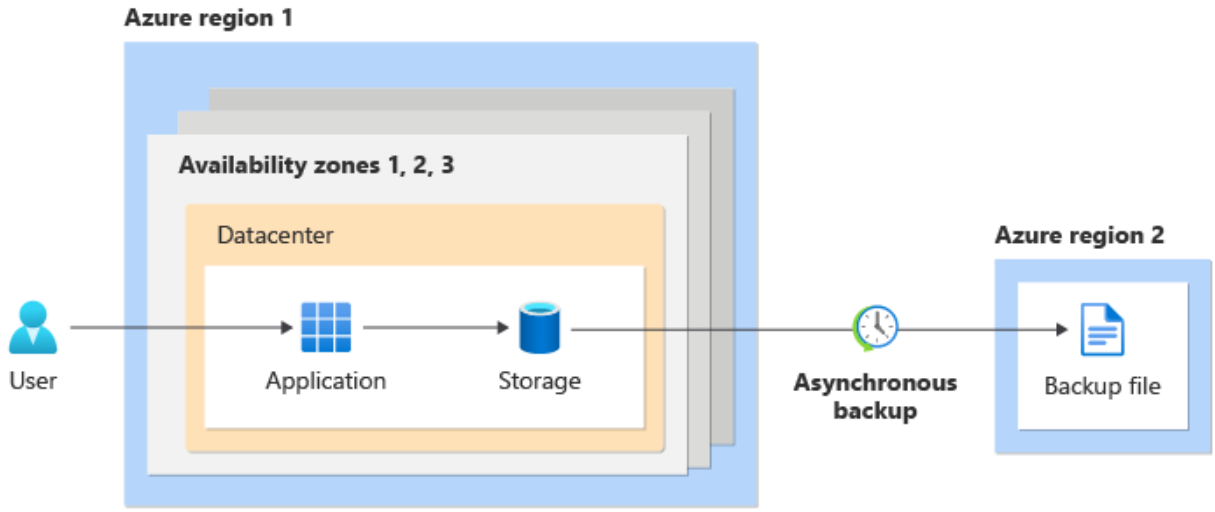
This table summarizes some of the concerns from an architectural perspective:

Architectural concern	Impact
Compliance with data residency	**High.** When you deploy a solution that uses this approach, data is stored in the Azure region that you select.
Regional availability	**Regions with availability zones.** This approach is available in any region that supports availability zones.

This approach is possible with many Azure services, including Azure Virtual Machine Scale Sets, Azure App Service, Azure Functions, Azure Kubernetes Service, Azure Storage, Azure SQL, Azure Service Bus, and many others. For a complete list of services that support zone redundancy, see Availability zone service and regional support.

Zone-redundant deployments with backup across regions

You can extend a zone-redundant deployment by performing regular backups of your infrastructure and data to a secondary region. This approach gives you the benefits of a zone-redundant approach and adds a layer of protection to mitigate the extremely unlikely event of a full region outage.



When you implement this approach, you need to carefully consider your RTO and RPO:

- **Recovery time**: If a regional outage does occur, you might need to rebuild your solution in another Azure region, which affects your recovery time. Consider building your solution by using an IaC approach so that you can quickly redeploy into another region during a major disaster. Ensure that your deployment tools and processes are just as resilient as your applications so that you can use them to redeploy your solution even if an outage occurs. Plan for and rehearse the steps required to restore your solution back to a working state.
- **Recovery point**: Your backup frequency determines the amount of data loss that you might experience (your recovery point). You can typically control the frequency of backups to meet your RPO.

> **Tip**
>
> This approach often provides a good balance for all architectural concerns. If you aren't sure which approach to use, start with this type of deployment.

This table summarizes some of the pillar concerns:

Pillar	Impact
Reliability	**Very high reliability.** Services are resilient to an outage of a datacenter or availability zone. For most services, data is replicated across zones automatically and with no delay. Data is backed up asynchronously to a geographically

Pillar	Impact
	separated region. This backup reduces the effect of a full region outage by minimizing data loss. After a full region outage, you can manually restore operations into another region. However, recovery processes can be complex, and it can take time to manually restore into the other region.
Cost Optimization	**Moderate cost.** Typically, adding a backup to another region costs only slightly more than implementing zone redundancy.
Performance Efficiency	*For most workloads:* **Acceptable performance.** This approach is likely to provide satisfactory performance. *For highly latency-sensitive workloads:* **Low performance.** Some components might be sensitive to latency due to inter-zone traffic or data replication time.
Operational Excellence	*During an availability zone outage:* **High operational efficiency.** Failover is the responsibility of Microsoft and happens automatically. *During a regional outage:* **Low operational efficiency.** Failover is your responsibility and might require manual operations and redeployments.

This table summarizes some of the concerns from an architectural perspective:

Architectural concern	Impact
Compliance with data residency	**Depends on region selection.** Data is stored primarily in the Azure region that you specify, but you need to select another region to store your backups. Select a region that's compatible with your data residency requirements.
Regional availability	**Regions with availability zones.** This approach is available in any region that supports availability zones.

Deployment approach 4: Multi-region deployments

You can use multiple Azure regions together to distribute your solution across a wide geographical area. You can use this multi-region approach to improve your solution's reliability or to support geographically distributed users. If data residency is an important concern for your solution, carefully consider which regions you use to ensure that your data is stored only in locations that meet your requirements.

Active and passive regions

Multi-region architectures are complex, and there are many ways to design a multi-region solution. For some workloads, it makes sense to have multiple regions actively

processing requests simultaneously (active-active deployments). For other workloads, it's better to designate one *primary region* and use one or more *secondary regions* for failover (active-passive deployments). This section focuses on the second scenario, in which one region is active and another is passive. For information about active-active multi-region solutions, see Deployment Stamps pattern and Geode pattern.

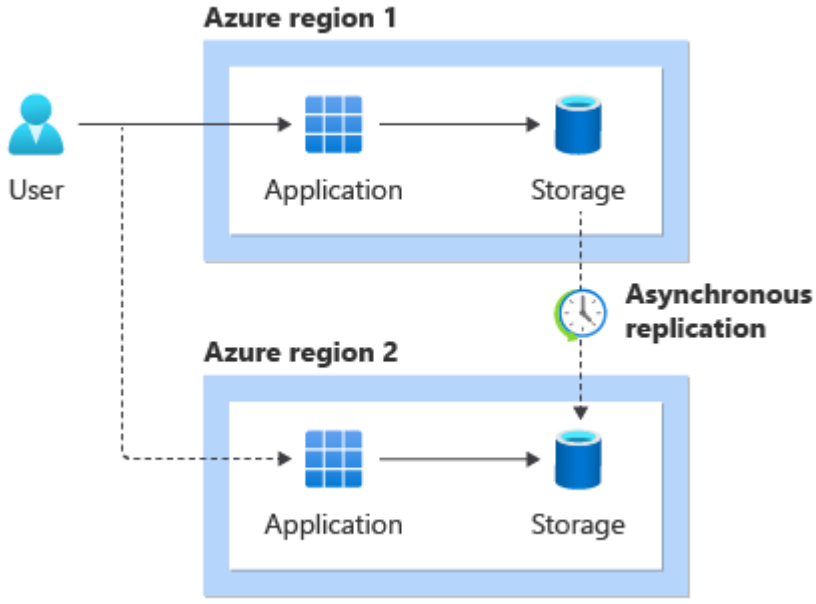
Data replication

Communicating across regions is much slower than communicating within a region. In general, a larger geographic distance between two regions incurs more network latency. See Azure network round-trip latency statistics for the expected network latency when you connect between two regions.

Cross-region network latency can significantly affect your solution design because you need to carefully consider how the extra latency affects data replication and other transactions. For many solutions, a cross-region architecture requires *asynchronous* replication to minimize the effect of cross-region traffic on performance.

Asynchronous data replication

When you implement asynchronous replication across regions, your application doesn't wait for all regions to acknowledge a change. After the change is committed in the primary region, the transaction is considered complete. The change is replicated to the secondary regions at a later time. This approach ensures that inter-region connection latency doesn't directly affect application performance. However, because of the delay in replication, a region-wide outage might result in some data loss. This data loss can occur because a region might experience an outage after a write is completed in the primary region but before the change could be replicated to another region.



This table summarizes some of the pillar concerns:

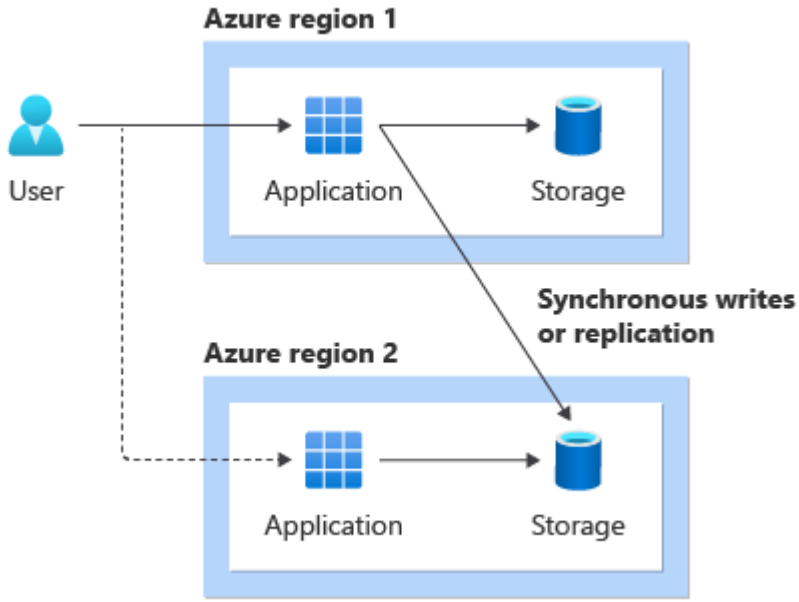
Pillar	Impact
Reliability	**High reliability.** The solution is resilient to an outage of a datacenter, an availability zone, or an entire region. Data is replicated but might not be synchronous, so some data loss is possible in a failover scenario.
Cost Optimization	**High cost.** You need to deploy separate resources in each region, and each resource incurs deployment and maintainenance costs. Data replication across regions might also incur significant costs.
Performance Efficiency	**High performance.** Application requests don't require cross-region traffic, so traffic is typically low latency.
Operational Excellence	**Low operational efficiency.** You need to deploy and manage resources across multiple regions. You're also responsible for failover between regions during a regional outage.

This table summarizes some of the concerns from an architectural perspective:

Architectural concern	Impact
Compliance with data residency	**Depends on region selection.** This approach requires you to select multiple regions for your workload to run in. Choose regions that are compatible with your data residency requirements.
Regional availability	Many Azure regions are paired. Some Azure services use paired regions to replicate data automatically. If you run your workload in a region that doesn't have a pair, you might need to use a different approach to replicate your data.

Synchronous data replication

If you implement a synchronous multi-region solution, your application needs to wait for write operations to complete in each Azure region before the transaction is considered complete. The latency incurred by waiting for write operations depends on the distance between the regions. For many workloads, inter-region latency can make synchronous replication too slow to be useful.



This table summarizes some of the pillar concerns:

Pillar	Impact
Reliability	**Very high reliability.** The solution is resilient to an outage of a datacenter, an availability zone, or an entire region. Data is always in sync across regions, so, even if a complete region loss occurs, your data is available and complete in another region.
Cost Optimization	**High cost.** You need to deploy separate resources in each region, and each resource incurs deployment and maintainenance costs. Data replication across regions might also incur significant costs.
Performance Efficiency	**Low performance.** Application requests require cross-region traffic. Depending on the distance between the regions, synchronous replication might add significant latency to requests.
Operational Excellence	**Low operational efficiency.** You need to deploy and manage resources across multiple regions. You're also responsible for failover between regions during a regional outage.

This table summarizes some of the concerns from an architectural perspective:

Architectural concern	Impact
Compliance with data residency	**Depends on region selection.** This approach requires you to select multiple regions for your workload to run in. Select regions that are compatible with your data residency requirements.
Regional availability	Many Azure regions are paired. Some Azure services use paired regions to replicate data automatically. If you run your workload in a region that

Architectural concern	Impact
	doesn't have a pair, you might need to use a different approach to replicate your data.

Region architectures

When you design a multi-region solution, consider whether the Azure regions you plan to use are paired.

You can create a multi-region solution even when the regions aren't paired. However, the approaches that you use to implement a multi-region architecture might be different. For example, in Azure Storage, you can use geo-redundant storage (GRS) with paired regions. If GRS isn't available, consider using features like Azure Storage object replication, or design your application to write to multiple regions.

Combine multi-zone and multi-region approaches

You can also combine multi-zone and multi-region approaches. For example, you might deploy zone-redundant components into each region and also configure replication between the regions. Configuring this type of approach can be complicated, and this approach can be expensive, but for some solutions it provides a very high degree of reliability.

> **Important**
>
> Mission-critical workloads should use both multiple availability zones *and* multiple regions. For more information about the considerations that you should give when designing mission-critical workloads, see **Mission-critical workload documentation**.

How Azure services support deployment approaches

It's important to understand the specific details of the Azure services that you use. For example, some Azure services require that you configure their availability zone configuration when you first deploy the resource, while others support changing the deployment approach later. Similarly, some service features might not be available with every deployment approach.

To learn more about the specific deployment options and approaches to consider for each Azure service, visit the Reliability hub.

Examples

This section describes some common use cases and the key requirements that you typically need to consider for each workload. For each workload, a suggested deployment approach is provided, based on the requirements and approaches described in this article.

Line-of-business application for an enterprise

Contoso, Ltd., is a large manufacturing company. The company is implementing a line-of-business application to manage some components of its financial processes.

Business requirements: The information that the system manages is difficult to replace, so data needs to be persisted reliably. The architects say that the system needs to incur as little downtime and as little data loss as possible. Contoso's employees use the system throughout the workday, so high performance is important to avoid keeping team members waiting. Cost is also a concern, because the finance team has to pay for the solution.

Suggested approach: Zone-redundant deployment or zone-redundant deployment with backup across regions.

Internal application

Fourth Coffee is a small business. The company is developing a new internal application that employees can use to submit timesheets.

Business requirements: For this workload, cost efficiency is a primary concern. Fourth Coffee evaluated the business impact of downtime and decided that the application doesn't need to prioritize resiliency or performance. The company accepts the risk that an outage in an Azure availability zone or region might make the application temporarily unavailable.

Suggested approach: Locally redundant deployment.

Legacy application migration

Fabrikam, Inc., is migrating a legacy application from an on-premises datacenter to Azure. The implementation will use an IaaS approach that's based on virtual machines. The application wasn't designed for a cloud environment, and communication between the application tier and the database is very chatty.

Business requirements: Performance is a priority for this application. Resiliency is also important, and the application must continue to work even if an Azure datacenter experiences an outage.

Suggested approach: Zonal (pinned) deployment, with passive deployments across multiple availability zones (in-region DR).

Healthcare application

Lamna Healthcare Company is implementing a new electronic health record system on Azure.

Business requirements: Because of the nature of the data that this solution stores, data residency is critically important. Lamna operates under a strict regulatory framework that mandates that data must remain in a specific location.

Suggested approach: Lamna might consider a zone-redundant deployment or a zone-redundant deployment with backup across regions. The company could also consider a multi-region deployment if there are multiple regions that fit Lamna's data residency requirements.

Banking system

Woodgrove Bank runs its core banking operations from a large solution that's deployed to Azure.

Business requirements: This is a mission-critical system. Any outages can cause major financial impact for customers. As a result, Woodgrove Bank has very low risk tolerance. The system needs the highest level of reliability possible, and the architecture needs to mitigate the risk of any failures that can be mitigated.

Suggested approach: Multi-region deployment. The architecture must use regions that fit the company's data residency requirements.

Software as a service (SaaS)

Proseware, Inc., builds software that's used by companies across the world. The company's user base is widely distributed geographically.

Business requirements: Proseware needs to enable each of its customers to choose a deployment region that's close to the customer. Enabling this choice is important for latency and for the customers' data residency requirements.

Suggested approach: Multi-region deployment. Alternatively, Proseware could consider using a single-region deployment with a global traffic acceleration solution, like Azure Front Door.

Related links

Following are some reference architectures and example scenarios for multi-zone and multi-region solutions:

- Baseline highly available zone-redundant web application
- Highly available multi-region web application
- Multi-region N-tier application
- Multi-tier web application built for HA/DR

Many Azure services provide guidance about how to use multiple availability zones, including the following examples:

- Azure Site Recovery: Enable Azure VM disaster recovery between availability zones
- Azure NetApp Files: Understand cross-zone replication of Azure NetApp Files
- Azure Storage redundancy

Reliability checklist

Refer to the complete set of recommendations.

Reliability checklist

Recommendations for designing a reliable scaling strategy

05/29/2025

Applies to this Azure Well-Architected Framework Reliability checklist recommendation:

Expand table

RE:06	**Implement a timely and reliable scaling strategy at the application, data, and infrastructure levels. Base the scaling strategy on actual or predicted usage patterns and minimize manual intervention.**

This guide describes the recommendations for designing a reliable scaling strategy.

Definitions

Expand table

Term	Definition
Vertical scaling	A scaling approach that adds compute capacity to existing resources.
Horizontal scaling	A scaling approach that adds instances of a given type of resource.
Autoscaling	A scaling approach that automatically adds or removes resources when a set of conditions is met.

> **Note**
>
> Scaling operations can be static (regularly scheduled daily scaling to accommodate normal load patterns), automatic (an automated process in response to conditions being met), or manual (an operator performs a one-time scaling operation in reaction to an unanticipated load change). Both vertical and horizontal scaling can be performed via any of these methods. However, automatic vertical scaling requires additional custom automation development and can cause downtime depending on the resources being scaled.

The system must be designed to be horizontally scalable. Avoid making assumptions about instance affinity. Don't design solutions that require that the code is always running in a specific instance of a process. When scaling a cloud service or website horizontally, don't assume that a series of requests from the same source are always routed to the same instance.

For the same reason, design services to be stateless to avoid requiring a series of requests from an application to always be routed to the same instance of a service. When designing a service that reads messages from a queue and processes them, don't make any assumptions about which instance of the service handles a specific message. Autoscaling could start more instances of a service as the queue length grows. The Competing Consumers pattern describes how to handle this scenario.

To use critical time in autoscaling decisions, it's helpful to have a library automatically add the relevant information to the headers of messages while they're sent and processed. One library that provides this functionality is NServiceBus.

Key design strategies

Design according to load patterns

To design a reliable scaling strategy for your workloads, focus on identifying load patterns for the user and system flows for each workload that leads to a scaling operation. Here are examples of different load patterns and their corresponding scaling strategies:

- *Static:* Every night by 11 PM EST, the number of active users of your application falls below 100, and the CPU utilization for the app servers drops by 90% across all nodes. To handle this, you can schedule the scale down of your compute nodes to the minimum count (2) between 11 PM and 6 AM EST.

- *Dynamic, Regular, and Predictable:* Every Monday morning, 1000 employees across multiple regions sign in to the ERP system. To manage this, you can schedule the scale-out of your compute nodes to the normal daily capacity before the first region starts work.

- *Dynamic, Irregular, and Predictable:* A product launch happens on the first day of the month, and there's historical data from previous launches on how the traffic increases in these situations. To address this, you can define a one-time scheduled scale-up of your compute and database instances on the morning of a product launch, and scale back down after one week.

- *Dynamic, Irregular, and Unpredictable:* A large-scale event causes a spike in demand for a product. For example, companies manufacturing and selling dehumidifiers can experience a sudden surge in traffic after a hurricane or other flooding event when people in affected areas need to dry rooms in their homes. To handle this, you can set autoscale thresholds to account for unplanned traffic spikes.

Adapt scaling strategies to fit individual components or flows

There's no one-size-fits-all scaling strategy. Different cloud services have different degrees of support of scaling and different approaches to scaling. For that reason, it's important to understand how scaling is supported and implemented across all of your workload components to design your overall scaling strategy. You might apply scaling strategies at the individual component level or at the flow level, depending on your architectural design. When determining how you'll implement scaling across your workload, consider these factors:

- **Those components that can't be scaled out**. An example would be large, relational databases that don't have sharding enabled and can't be refactored without significant impact. Document the resource limits published by your cloud provider and monitor those resources closely. Include those specific resources in your future planning for migrating to scalable services.

- **The relationship of the flow's components in terms of order of scale operations**. Ensure that you don't inadvertently overload a downstream component by scaling an upstream component first.

- **Any stateful application elements that might be interrupted by a scaling operation and any session affinity (or session stickiness) that's implemented**. Stickiness can limit your scaling ability and introduces single points of failure. Design your workload to be stateless to the extent practical.

- **Potential bottlenecks**. Scaling out doesn't fix every performance issue. For example, if your backend database is the bottleneck, it doesn't help to add more web servers. Identify and resolve the bottlenecks in the system first before just adding more instances. Stateful parts of the system are the most likely cause of bottlenecks.

- **Handling long-running tasks**. Design a long-running task to support both scaling out and scaling in. Without due care, such a task could prevent an instance of a process from being shut down cleanly when the system scales in. Or, it could lose data if the process forcibly terminates. Ideally, refactor a long-running task and break up the processing that it performs into smaller, discrete chunks. The Pipes and Filters pattern provides an example of how you can achieve this solution.

Choose the right technology

Making well-informed technology choices with scaling in mind will help you ensure that your workload can meet your reliability targets as your workload evolves. Research the scaling abilities offered for different resources that offer similar functionality and choose the best combination for your future growth plans. For example, you might have several options for

data stores that can host the particular kind of databases you'll use. However, one choice might have better scaling functionality out-of-the-box than others, which could make it a better choice for your workload.

- **Take advantage of services that automatically scale.** When practical, use SaaS services that automatically scale without any configuration or input. Global services like Microsoft Entra ID offer this functionality. Serverless solutions also offer automatic scaling and can be good choices for many use cases.
- **Take advantage of services that offer out-of-the-box scaling.** Many PaaS services offer integrated, easy-to-use scaling features that you can configure to meet your reliability requirements. For example, you can configure throughput scaling for Cosmos DB to meet your particular requirements.

Automate scaling

Automate the scaling operations for your workload components to extent practical. When using resources that have configurable autoscaling functionality, build the configuration logic into your infrastructure-as-code (IaC) deployment code. When using resources that don't have offer autoscaling out-of-the-box, build automation to perform scaling operations using native automation tooling and include the automation code in your IaC code.

If you base your autoscaling strategy on counters that measure business processes, such as the number of orders placed per hour or the average running time of a complex transaction, ensure that you fully understand the relationship between the results from these types of counters and the actual compute capacity requirements. It might be necessary to scale more than one component or compute unit in response to changes in business process counters.

Keep in mind that autoscaling might not be the most appropriate mechanism to handle a sudden burst in a workload. It takes time to provision and start new instances of a service or add resources to a system, and the peak demand might pass by the time these extra resources are available. In this scenario, it might be better to throttle the service. For more information, see the Throttling pattern.

Conversely, if you need the capacity to process all requests when the volume fluctuates rapidly, and cost isn't a major contributing factor, consider using an aggressive autoscaling strategy that starts more instances more quickly. You can also use a scheduled policy that starts a sufficient number of instances to meet the maximum load before that load is expected.

Choose appropriate scale units

Base your scaling strategy on scale units, which are the logical grouping of components to scale together and the scale increments to use (like moving from one VM SKU to another). Options to consider are:

- **Scaling resources individually**: You might need to only scale individual VMs or databases.
- **Scaling a full component at the same time:** For example, you might have a microservice API that is composed of an App Service, database, and queues that need to scale simultaneously.
- **Scaling the full solution**: For complex or mission critical workloads, scaling the entire solution as a deployment stamp can simplify your scaling strategy. Rather than managing the scaling schedules and autoscaling thresholds of many distinct resources, you can apply a limited set of scaling definitions to a deployment stamp and then mirror that across stamps as needed.

> ⓘ **Important**
>
> Set a maximum limit on the number of scale units that can be automatically allocated to avoid excess costs.

Optimize the scale unit initialization time

When designing your scaling strategy, keep in mind that different services scale at different timescales. There are some services that scale near-instantaneously and others that scale much slower. For example, API Management instances can take up to 45 minutes to finish their scaling operations. To account for the scaling operation's timescale, properly plan to perform the scaling operation before the expected increased load hits your workload. Other recommendations to consider include:

- Pre-initialize nodes that will be deployed to reduce the time required for initialization.
- Allow for a buffer time around configuration changes before you make further changes or use the system in an uncommon way. For example, you might deallocate an App Gateway backend instance via a rule change. You'll need to wait for connections to drain from that instance before it can be safely removed.
- Over-provision resources to handle increased load while scaling takes place. You might ensure that VMs normally run at 75% utilization capacity to ensure that they can handle increased load while horizontal scaling takes place.

- Fine-tune your scaling thresholds with monitoring. Use your capacity monitoring to ensure that your scaling thresholds to trigger scaling operations.

Scale data stores using sharding and partitioning

Optimize the reliability of your data estate by including it in your scaling strategy. Partitioning data spreads a database across logical or physical storage resources, removing single points of failure. Choose the best partitioning strategy for your use case.

- **Horizontal partitioning (sharding)**: Partitions (shards) are placed in separate data stores, but all partitions have the same schema. Each shard holds a subset of the database. This is a good approach to optimize reliability as it helps facilitate load balancing and minimizes the burden on operations when dealing with issues. Shards can be replicated for higher reliability.

- **Vertical partitioning**: Each partition holds a subset of the fields for items in the data store. The fields are divided according to their usage pattern.

- **Functional partitioning**: Data is aggregated according to how each bounded context in the system uses the data.

Consider combining these strategies when you design a partitioning scheme. For example, you might divide data into shards and then use vertical partitioning to further subdivide the data in each shard.

Optimize your partition strategy for scalability. Analyze data access patterns to determine which operations require the most processing resources and balance your partitions to ensure that each one has enough resources to handle scalability requirements.

For detailed guidance on partitioning and sharding, see the design guide

Monitor your scaling operations

The autoscaling mechanism should monitor the autoscaling process and log the details of each autoscaling event (what triggered it, what resources were added or removed, and when). If you create a custom autoscaling mechanism, ensure that it incorporates this capability. Proactively analyze the information to help measure the effectiveness of the autoscaling strategy, and tune it if necessary.

Azure facilitation

An autoscaling feature is available in many Azure services. It lets you easily configure conditions to automatically scale instances horizontally. Some services have limited or no built-in functionality to automatically scale in, so be sure to document these cases and define processes to deal with scaling in.

Many Azure services offer APIs that you can use to design custom automatic scaling operations using Azure Automation, such as the code samples at Autoscale your Azure IoT Hub. You can use tools like KEDA for event-driven autoscaling, which is available in Azure Kubernetes Service and Azure Container Apps.

Azure Monitor autoscale provides a common set of autoscaling functionality for Azure Virtual Machine Scale Sets, Azure App Service, Azure Spring Apps and more. Scaling can be performed on a schedule or based on a runtime metric, such as CPU or memory usage. See the Azure Monitor guide for best practices.

Tradeoffs

> **Tradeoff**: Scaling up has cost implications, so optimize your strategy to scale down as soon as possible to help keep costs under control. Ensure that the automation you're employing to scale up also has triggers to scale down.

Example

Refer to the AKS baseline reference architecture scaling guidance.

Related links

- Performance efficiency scaling
- Best practices for autoscale

Reliability checklist

Refer to the complete set of recommendations.

Reliability checklist

Recommendations for self-healing and self-preservation

Article • 02/28/2025

Applies to this Azure Well-Architected Framework Reliability checklist recommendation:

Expand table

RE:07	**Strengthen the resiliency of your workload by implementing self-preservation and self-healing measures. Use built-in features and well-established cloud patterns to help your workload remain functional during and recover from incidents.**

This guide describes the recommendations for building self-preservation and self-healing capabilities into your application architecture to optimize reliability.

Self-preservation capabilities add resilience to your workload. They reduce the likelihood of a full outage and allow your workload to operate normally, or in a degraded state, when failures occur. Self-healing capabilities help you avoid downtime by building in failure detection and automatic corrective actions to respond to failures.

Definitions

Expand table

Term	Definition
Self-healing	The ability of your workload to automatically resolve issues by recovering affected components and if needed, failing over to redundant infrastructure.
Self-preservation	The ability of your workload to be resilient against potential problems.

Key design strategies

Design for redundancy

One of the most effective strategies to protect your workload from malfunctions is to build redundancy into all of its components and avoid single points of failure. Being able to fail components or the entire workload over to redundant resources provides an efficient way to handle most faults in your system.

Build redundancy at different levels, consider redundant infrastructure components such as compute, network, and storage; and consider deploying multiple instances of your solution. Depending on your business requirements, you can build redundancy within a single region or across regions. You can also decide whether you need an active-active or an active-passive design to meet your recovery requirements. See the redundancy, regions and availability zones, and highly available multi-region design Reliability articles for in-depth guidance on this strategy.

Design for self-preservation

To design your workload for self-preservation, follow infrastructure and application architecture design patterns to optimize your workload's resiliency. To minimize the chance of experiencing a full application outage, increase the resiliency of your solution by eliminating single points of failure and minimizing the blast radius of failures. The design approaches in this article provide several options to strengthen the resilience of your workload and meet your workload's defined reliability targets.

Infrastructure design guidance and patterns

At the infrastructure level, a redundant architecture design should support your critical flows, with resources deployed across availability zones or regions. Implement autoscaling when possible. Autoscaling helps protect your workload against unanticipated bursts in activity, further reinforcing your infrastructure.

Use the Deployment Stamps pattern or the Bulkhead pattern to minimize the blast radius when problems arise. These patterns help to keep your workload available if an individual component is unavailable. Use the following application design patterns in combination with your autoscaling strategy.

- **Deployment Stamps pattern**: Provision, manage, and monitor a varied group of resources to host and operate multiple workloads or tenants. Each individual copy is called a *stamp*, or sometimes a *service unit*, *scale unit*, or *cell*.
- **Bulkhead pattern**: Partition service instances into different groups, known as *pools*, based on the consumer load and availability requirements. This design helps to isolate failures and allows you to sustain service functionality for some consumers, even during a failure.

Application design guidance and patterns

Avoid building monolithic applications in your application design. Use loosely coupled services or microservices that communicate with each other via well-defined standards to reduce the risk of extensive problems when malfunctions happen to a single component. For example, you may standardize the use of a service bus to handle all asynchronous communication. Standardizing communication protocols ensures that applications design is consistent and simplified, which makes the workload more reliable and easier to troubleshoot when malfunctions happen. When practical, prefer asynchronous communication between components over synchronous communication to minimize timeout issues, like dead-lettering.

Use industry-proven patterns to help you develop your design standards and simplify aspects of the architecture. Design patterns that can help support reliability can be found in the Reliability patterns article.

Design for self-healing

To design your workload for self-healing, implement failure detection so automatic responses are triggered and critical flows gracefully recover. Enable logging to provide operational insights about the nature of the failure and the success of the recovery. The approaches that you take to achieve self-healing for a critical flow depend on the reliability targets that are defined for that flow and the flow's components and dependencies.

Infrastructure design guidance

At the infrastructure level, your critical flows should be supported by a redundant architecture design, with automated failover enabled for components that support it. You can enable automated failover for the following types of services:

- **Compute resources**: Azure Virtual Machine Scale Sets and most platform as a service (PaaS) compute services can be configured for automatic failover.
- **Databases**: Relational databases can be configured for automatic failover with solutions like Azure SQL failover clusters, Always On availability groups, or built-in capabilities with PaaS services. NoSQL databases have similar clustering capabilities and built-in capabilities for PaaS services.
- **Storage**: Use redundant storage options with automatic failover.

Application design guidance

In addition to using design patterns that support reliability, other strategies that can help you develop self-healing mechanisms include:

- **Use checkpoints for long-running transactions**: Checkpoints can provide resiliency if a long-running operation fails. When the operation restarts, for example if it's picked up by another virtual machine, it can resume from the last checkpoint. Consider implementing a mechanism that records state information about the task at regular intervals. Save this state in durable storage that can be accessed by any instance of the process running the task. If the process is shut down, the work that it was performing can be resumed from the last checkpoint by using another instance. There are libraries that provide this functionality, such as NServiceBus and MassTransit. They transparently persist state, where the intervals are aligned with the processing of messages from queues in Azure Service Bus.

- **Implement automated self-healing actions:** Use automated actions that are triggered by your monitoring solution when pre-determined health status changes are detected. For example, if your monitoring detects that a web app isn't responding to requests, you can build automation through a PowerShell script to restart the app service. Depending on your team's skill set and preferred development technologies, use a webhook or function to build more complex automation actions. See the Event-based cloud automation reference architecture for an example of using a function to respond to database throttling. Using automated actions can help you recover quickly and minimize the necessity of human intervention.

Implement a graceful degradation mode

Despite your self-preservation and self-healing mechanisms, you may still encounter situations where one or more components malfunction to the extent that they become unavailable for some amount of time. In these cases, ideally, your workload can maintain enough functionality for business to continue in a degraded state. To ensure that this is possible, design and implement a graceful degradation mode. This is a distinct workflow that is enabled in reaction to failed components. Considerations for the design and implementation include:

- **Failure detection and automated initiation:** Your monitoring and alerting systems should detect degraded and failed components, so use those signals to build a workflow that determines when switching to your graceful degradation mode is necessary. The workflow should then automatically reroute calls to and from affected components to alternative components, or other similar actions.

- **Implement a degraded user experience:** Include a notification mechanism for users in your graceful degradation mode to ensure that they know what functionality remains and what has changed. This typically is reflected in messages tied to different functions of the workload, like a pop-up when adding items to a cart, for example.
- **Build alternatives paths to complete your workload's essential functions:** Reflect on your workload's critical flows and determine how you can maintain those flows when core components are unavailable. For example, if a database is down, the application might switch to a read-only mode using cached data. To further illustrate this example, if a payment gateway is down, using cached data might allow users to save their cart and complete the purchase later.

Implement mechanisms for handling transient faults

Transient faults, like network timeouts, are a common issue for cloud workloads, so having mechanisms in place to handle them can minimize downtime and troubleshooting efforts as you operate your workload in production. Since most operations that fail due to a transient fault will succeed if sufficient time is allowed before retrying the operation, using a retry mechanism is the most common approach for dealing with transient faults. When designing your retry strategy, consider the following:

Refer to the Transient faults design guide for a full review of recommendations and considerations.

Implement background jobs

Background jobs are an effective way to enhance the reliability of a system by decoupling tasks from the user interface (UI). Implement a task as a background job if it doesn't require user input or feedback and if it doesn't affect UI responsiveness.

Common examples of background jobs are:

- CPU-intensive jobs, such as performing complex calculations or analyzing structural models.
- I/O-intensive jobs, such as running multiple storage operations or indexing large files.
- Batch jobs, such as updating data regularly or processing tasks at a specific time.
- Long-running workflows, such as completing an order or provisioning services and systems.

Refer to the background jobs design guide for detailed guidance for a full review of recommendations and considerations.

Azure facilitation

Most Azure services and client SDKs include a retry mechanism. But they differ because each service has different characteristics and requirements, so each retry mechanism is tuned to a specific service. For more information, see Recommendations for transient fault handling.

Use Azure Monitor action groups for notifications, like email, voice or SMS, and to trigger automated actions. When you're notified of a failure, trigger an Azure Automation runbook, Azure Event Hubs, an Azure function, a logic app, or a webhook to perform an automated healing action.

Example

For example use cases of some patterns, see the reliable web app pattern for .NET. Follow these steps to deploy a reference implementation.

Related links

- Reliability patterns
- Handle transient faults
- Develop background jobs
- Cloud design patterns
- Design for self-healing

Reliability checklist

Refer to the complete set of recommendations.

Reliability checklist

Feedback

Was this page helpful? Yes No

Recommendations for designing a reliability testing strategy

Article • 11/15/2023

Applies to this Azure Well-Architected Framework Reliability checklist recommendation:

Expand table

RE:08	**Test for resiliency and availability scenarios by applying the principles of chaos engineering. Ensure that your graceful degradation implementation and scaling strategies are effective by performing active malfunction and simulated load testing.**

This guide describes the recommendations for designing a reliability testing strategy to validate and optimize the reliability of your workload. Reliability testing focuses on the resiliency and availability of your workload, specifically the critical flows that you identify when you design your solution. This guide provides general testing guidance and guidance that's specific to fault injection and chaos engineering.

Definitions

Expand table

Term	Definition
Availability	The amount of time that an application workload runs in a healthy state without significant downtime.
Chaos engineering	The practice of subjecting applications and services to real-world stresses and failures. The goal of chaos engineering is to build and validate resilience to unreliable conditions and missing dependencies.
Fault injection	The act of introducing an error to a system to test the resiliency of the system.
Recoverability	A synonym for resiliency.
Resiliency	An application workload's ability to withstand and recover from failure modes.

Key design strategies

Test reliability preparedness

- Routinely perform testing to validate existing thresholds, targets, and assumptions. When a major change occurs in your workload, run regular testing. Perform most testing in

testing and staging environments. It's also beneficial to run a subset of tests against the production system. Plan a one-to-one parity of key test environments with the production environment.

- Automate testing to help ensure consistent test coverage and reproducibility. Automate common testing tasks and integrate them into your build processes. Manually testing software is tedious and susceptible to error, but you can conduct manual exploratory testing. For cases in which you need to develop automated testing, use manual testing to determine the scope of the tests to develop.
- Adopt a shift-left testing approach to perform resiliency and availability testing early in the development cycle.
- Adapt a simple documentation format, so it's easy for everyone to understand the process and the results of every regular test.
- Share the documented results with the appropriate teams, like operational teams, technology leadership, business stakeholders, and disaster recovery stakeholders. The results should inform the refinement of reliability targets, such as service-level objectives (SLOs), service-level agreements (SLAs), recovery time objectives (RTOs), and recovery point objectives (RPOs).
- Create a regular testing cadence for your backups. Restore the data to isolated systems to help ensure that the backups are valid and that restores are functional.
- Document and share recovery time metrics with your disaster recovery stakeholders to ensure that expectations for recovery are appropriate.
- Use industry standard deployment testing procedures to help ensure that you have an automated, predictable, and efficient deployment process.
- Test your workload's ability to withstand transient failures. For more information, see Recommendations for handling transient faults.
- Test your workload's ability to respond to changes in load patterns and spikes in usage. Use this information to help you test your scaling strategy. For information about load and stress testing, see Recommendations for testing.
- Test how your workload handles failures in dependent services or other dependencies by using fault injection.
- Test and validate how your self-healing and self-preservation design responds to malfunctions. Test automated and manual recovery operations.

- Test your disaster recovery plan to respond to catastrophic failures and other major incidents.
- Test your workload's ability to degrade gracefully and minimize the blast radius of component malfunction by using fault injection.

Take advantage of planned and unplanned outages

When your workload is offline due to planned maintenance or an unplanned outage, you have a unique opportunity to perform testing and improve your understanding of your workload. The following sections provide recommendations for each scenario.

Planned maintenance

When you have planned maintenance windows for updates or patches, you can test components and flows that aren't involved in the maintenance work. Perform tests without the potential risk of unexpectedly degrading the workload or taking it offline altogether. If you have enough time during your maintenance window, you can also test the components and flows that are involved in the maintenance after the maintenance work is complete.

Unplanned outage

Use every outage incident as an opportunity to learn more about your workload and improve its resiliency by following these steps, ordered by priority:

- Get the workload back online for your customers. To do so, you might perform a workaround for the issue, resolve the issue, or initiate the recovery processes.
- Determine the root cause of the outage and address it. If you can fix the root cause as part of the investigation, document the root cause and the measures that you took to fix it. If the issue requires taking an additional maintenance window at a later time, ensure that your mitigation measures can handle the expected load by thoroughly testing it. Ensure that you have set up sufficient monitoring to cover your mitigation measures.
- If applicable, look for the same issue, or configuration weaknesses that might be affected by similar issues, across all the components in your workload. Use this opportunity to proactively address those components. Consult your incident history to detect patterns of similar issues across your workload.
- Use your findings to improve your testing strategy. Ensure that you have successfully addressed the root cause and similar problems by directly testing the same failure.

Use fault injection and chaos engineering

Fault-injection testing follows the principles of chaos engineering by highlighting the workload's ability to react to component failures. Perform fault-injection testing in pre-production and production environments. Apply testing to infrastructure and application layers. Apply the information that you learned Recommendations for performing failure mode analysis to ensure that you test only faults that you prioritize and that you have mitigation strategies that address faults. The key guidelines of chaos engineering are:

- **Be proactive.** Don't wait for failures to happen. Try to anticipate failures by conducting chaos experiments to discover and fix issues before they affect your production environment.
- **Embrace failure.** Accept and learn from the failures that occur in your system. See failures as a natural part of complex systems and use them as opportunities to learn and improve your system's reliability.
- **Break the system.** Deliberately inject faults or stress into your system to test its resilience. Simulate real-world failures or disruptions to test and improve your workload's recovery capabilities.
- **Identify and address single points of failure early.** As you test, consult and update your failure mode analysis to validate and address faults in your documentation. Apply reliability approaches, like redundancy and segmentation, to increase your workload's availability and minimize downtime.
- **Install guardrails and graceful mitigation.** Implement safety measures, like the Circuit Breaker pattern or the Throttling pattern, to increase availability. Implement graceful degradation approaches that enable business continuity during failures.
- **Minimize the blast radius.** Implement fault isolation strategies to help ensure that, even if a failure occurs, its scope is limited. The system continues to function with minimal effect on your customers.
- **Build immunity.** Use chaos engineering experiments to improve your workload's ability to prevent and recover from failures.

Chaos engineering is an integral part of workload team culture and an ongoing practice, not a short-term tactical effort in response to a single outage. Follow this standard method when you design your chaos experiments:

1. Start with a hypothesis. Each experiment should have a clear goal, like testing a given flow's ability to withstand the loss of a particular component.

2. Measure baseline behavior. Ensure that you have consistent reliability and performance metrics for the flow and components involved in a given experiment to compare with the degraded state when running your experiment.
3. Inject a fault or faults. The experiment should intentionally target specific components that can be recovered quickly and you should have an informed expectation of the effect that the fault injection will cause to help control the experiment's blast radius.
4. Monitor the resulting behavior. Gather telemetry about the individual flow components and the end-to-end flow behavior that the experiment targets to properly understand the effects of the fault. Compare the metrics that you gather with the baseline metrics for a full picture of the fault injection results.
5. Document the process and observations. Keeping detailed records of your experiments will inform the future decisions about the workload design, ensuring that you address the gaps that have been revealed over time.
6. Identify and act on the result. Plan for remediation steps that can be added to your workload backlog as improvements. Ensure that design improvement plans are reviewed and tested in nonproduction environments according to the same processes as other deployments.

Periodically validate your process, architecture choices, and code to quickly detect technical debt, integrate new technologies, and adapt to changing requirements.

When you conduct fault-injection experiments, you:

- Confirm that monitoring is in place and alerts are set up.
- Validate your process of assigning a directly responsible individual (DRI) to take ownership of an incident.
- Ensure that your documentation and investigation processes are up to date.

Integrate the following recommendations and considerations to optimize your chaos testing strategy:

- Challenge system assumptions. With testing, you try to improve the resiliency of your workload and your workload design strategies. Look for opportunities to inject faults into components and flows that you assume are reliable based on past experiences. They might not be reliable in your new workload.

- Validate change, such as the topology, platform, and resources. Without thorough testing, including fault-injection testing, you might have an incomplete picture of your workload after changes are made. For example, you might inadvertently introduce new dependencies or broken existing dependencies in ways that aren't immediately apparent.

- Use SLA buffers. Limit chaos testing to stay within your SLAs and avoid potential reputation or financial effects from outages. Your flow and component recovery targets

help define the scope of your testing.

- Establish an error budget as an investment in chaos and fault injection. Your error budget is the difference between achieving 100 percent of the SLO and achieving the agreed upon SLO.
- Stop the experiment if it goes beyond scope. Unknown results are an expected outcome of chaos experiments. Strive to achieve balance between collecting substantial result data and affecting as few production users as possible.
- Work closely with development teams to ensure the relevance of the injected failures. Use past incidents or issues as a guide. Examine dependencies and evaluate the results when you remove those dependencies.
- Identify and document previously undiscovered dependencies between different components within your workload that are revealed through chaos testing.
- Adjust recovery plans as necessary to account for dependencies that are discovered during chaos testing.
- Use the results from your experiments and tests as the basis for new experiments and tests. As unexpected behaviors arise, new tests might target those behaviors directly and give you the opportunity to design remediation strategies for them.

> **Tradeoff**: Fault-injection testing in production can be disruptive and can potentially cause downtime. Be transparent with stakeholders about this possibility and ensure that you have safeguards in place to terminate experiments and roll back plans to quickly reverse the failures that you introduce. To guard against unintended outages in production, ensure that you plan for sufficient redundancy and that your stakeholders understand the cost tradeoff.

Azure facilitation

Azure Test Plans is an easy-to-use, browser-based test management solution that provides all the capabilities required for planned manual testing, user acceptance testing, exploratory testing, and gathering feedback from stakeholders.

Azure Chaos Studio is a managed service that uses chaos engineering to help you measure, understand, and improve your cloud application and service resilience. Azure Chaos Studio reached general availability at Ignite 2023 and has lots of features to help you get started with Fault injection and resiliency testing for your application using Azure infrastructure.

Related links

- Backup and disaster recovery for Azure applications
- Checklist for reliability testing
- Test applications for availability and resiliency

Reliability checklist

Refer to the complete set of recommendations.

Reliability checklist

Recommendations for designing a disaster recovery strategy

Article • 04/03/2025

Applies to this Azure Well-Architected Framework Reliability checklist recommendation:

Expand table

RE:09	**Implement structured, tested, and documented business continuity and disaster recovery (BCDR) plans that align with the recovery targets. Plans must cover all components and the system as a whole.**

This guide describes recommendations for designing a reliable disaster recovery strategy for a workload. To meet internal service-level objectives (SLOs) or even a service-level agreement (SLA) that you have guaranteed for your customers, you must have a robust and reliable disaster recovery strategy. Failures and other major issues are expected. Your preparations to deal with these incidents determine how much your customers can trust your business to reliably deliver for them. A disaster recovery strategy is the backbone of preparation for major incidents.

Definitions

Expand table

Term	Definition
Failover	The automated and/or manual shifting of production workload traffic from an unavailable region to an unaffected geographical region.
Failback	The automated and/or manual shifting of production workload traffic from a failover region back to the primary region.

Key design strategies

This guide assumes that you have already performed the following tasks as part of your reliability planning:

- Identify critical and noncritical flows.
- Perform failure mode analysis (FMA) for your flows.

- Identify reliability targets.
- Design for reliability through redundancy, scaling, self-preservation, and self-healing.
- Design a robust testing strategy.

A reliable disaster recovery (DR) strategy builds on the foundation of a reliable workload architecture. Address reliability at every stage of building your workload to ensure that necessary pieces for optimized recovery are in place before you start designing your DR strategy. This foundation ensures that your workload's reliability targets, like recovery time objective (RTO) and recovery point objective (RPO), are realistic and achievable.

Maintain a disaster-recovery plan

The cornerstone of a reliable DR strategy for a workload is the *DR plan*. Your plan should be a living document that's routinely reviewed and updated as your environment evolves. Present the plan to the appropriate teams (operations, technology leadership, and business stakeholders) regularly (every six months, for example). Store it in a highly available, secure data store such as OneDrive for Business.

Follow these recommendations to develop your DR plan:

- Clearly define what constitutes a disaster and therefore requires activation of the DR plan.
    - Disasters are large-scale issues. They might be regional outages, outages of services like Microsoft Entra ID or Azure DNS, or severe malicious attacks like ransomware attacks or DDoS attacks.
    - Identify failure modes that aren't considered disasters, such as the failure of a single resource, so that operators don't mistakenly invoke their DR escalations. These failure modes can be addressed by troubleshooting the issue in place, redeploying the failed resources, or utilizing a Backup Plan
- Build the DR plan on your FMA documentation. Ensure that your DR plan captures the failure modes and mitigation strategies for outages that are defined as disasters. Update both your DR plan and your FMA documents in parallel so they're accurate when the environment changes or when testing uncovers unexpected behaviors.
    - Whether you develop DR plans for nonproduction environments depends on your business requirements and cost impacts. For example, if you offer quality-

assurance (QA) environments to certain customers for prerelease testing, you might want to include those environments in your DR planning.

- Clearly define roles and responsibilities within the workload team and understand any related external roles within your organization. Roles should include:
  - The party responsible for declaring a disaster.
  - The party responsible for declaring incident closure.
  - Operations roles.
  - Testing and validation roles.
  - Internal and external communications roles.
  - Retrospective and root-cause analysis (RCA) lead roles.
- Define the escalation paths that the workload team must follow to ensure that recovery status is communicated to stakeholders.
- Capture component-level recovery procedures, data estate-level recovery, and workload-wide recovery processes. Include a prescribed order of operations to ensure that components are recovered in the least impactful way. For example, recover and check databases before you recover the application.
  - Detail each component-level recovery procedure as a step-by-step guide. Include screenshots if possible.
  - Define your team's responsibilities versus your cloud hosting provider's responsibilities. For example, Microsoft is responsible for restoring a PaaS (platform as a service), but you're responsible for rehydrating data and applying your configuration to the service.
  - Include prerequisites for running the procedure. For example, list the required scripts or credentials that need to be gathered.
  - Capture the root cause of the incident and perform mitigation before you start recovery. For example, if the cause of the incident is a security issue, mitigate that issue before you recover the affected systems in your failover environment.
- Depending on the redundancy design for your workload, you might need to do significant post-failover work before you make the workload available to your customers again. Post-failover work could include DNS updates, database connection string updates, and traffic routing changes. Capture all of the post-failover work in your recovery procedures.

> **Note**
>
> Your redundancy design might allow you to automatically recover from major incidents fully or partially, so be sure that your plan includes processes and procedures around these scenarios. For example, if you have a fully active-active design that spans **availability zones or regions**, you might be able to transparently fail over automatically after an availability zone or regional outage and minimize the steps in your DR plan that need to be performed. Similarly, if you designed your workload by using **deployment stamps**, you might suffer only a partial outage if the stamps are deployed zonally. In this case, your DR plan should cover how to recover stamps in unaffected zones or regions.

- If you need to redeploy your app in the failover environment, use tooling to automate the deployment process as much as possible. Ensure that your DevOps pipelines have been predeployed and configured in the failover environments so that you can immediately begin your app deployments. Use automated end-to-end deployments, with manual approval gates where necessary, to ensure a consistent and efficient deployment process. The full deployment duration needs to align with your recovery targets.
  - When a stage of the deployment process requires manual intervention, document the manual steps. Clearly define roles and responsibilities.

- Automate as much of the procedure as you can. In your scripts, use declarative programming because it allows idempotence. When you can't use declarative programming, be mindful about developing and running your custom code. Use retry logic and circuit breaker logic to avoid wasting time on scripts that are stuck on a broken task. Because you run these scripts only in emergencies, you don't want incorrectly developed scripts to cause more damage or slow down your recovery process.

  > **Note**
  >
  > Automation poses risks. Trained operators need to monitor the automated processes carefully and intervene if any process encounters issues. To minimize the risk that automation will react to false positives, be thorough in your DR drills. Test all phases of the plan. Simulate detection to generate alerting, and then move through the entire recovery procedure.
  >
  > Remember that your DR drills should validate or inform updates to your recovery target metrics. If you find that your automation is susceptible to false

positives, you might need to increase your failover thresholds.

- Separate the failback plan from the DR plan to avoid potential confusion with the DR procedures. The failback plan should follow all of the DR plan's development and maintenance recommendations and should be structured in the same way. Any manual steps that were necessary for failover should be mirrored in the failback plan. Failback might happen quickly after failover, or it might take days or weeks. Consider failback as separate from failover.
  - The need to fail back is situational. If you're routing traffic between regions for performance reasons, failing back the load originally in the failed-over region is important. In other cases, you might have designed your workload to function fully regardless of which production environment it's located in at any time.

Conduct disaster-recovery drills

A DR testing practice is as important as a well-developed DR plan. Many industries have compliance frameworks that require a specified number of DR drills to be performed regularly. Regardless of your industry, regular DR drills are paramount to your success.

Follow these recommendations for successful DR drills:

- Perform at least one production DR drill per year. Tabletop (dry run) drills or nonproduction drills help ensure that the involved parties are familiar with their roles and responsibilities. These drills also help operators build familiarity ("muscle memory") by following recovery processes. But only production drills truly test the validity of the DR plan and the RTO and RPO metrics. Use your production drills to time recovery processes for components and flows to ensure that the RTO and RPO targets that have been defined for your workload are achievable. For functions that are out of your control, like DNS propagation, ensure that the RTO and RPO targets for the flows that involve those functions account for possible delays beyond your control.

- Use tabletop drills not only to build familiarity for seasoned operators but also to educate new operators about DR processes and procedures. Senior operators should take time to let new operators perform their role and should watch for improvement opportunities. If a new operator is hesitant or confused by a step in a procedure, review that procedure to ensure that it's clearly written.

Considerations

- Performing DR drills in production can cause unexpected catastrophic failures. Be sure to test recovery procedures in nonproduction environments during your initial deployments.
- Give your team as much maintenance time as possible during drills. When planning for maintenance time, use the recovery metrics that you capture during testing as *minimum time necessary* allotments.
- As your DR drill practices mature, you learn which procedures you can run in parallel and which you must run in sequence. Early in your drill practices, assume that every procedure must be run in sequence and that you need extra time in each step to handle unanticipated issues.

Define and maintain Backup Plans for resources within critical flows

Backup is an important part of your overall recovery process. Oftentimes it is just a part of your environment that needs recovery. DR plans are usually application or even region wide. Accidental or malicious deletion of data, file corruption, malware, and targeted ransomware attacks can all affect the availability of your workload. Having solid backup plans for each part of your environment is just as important as having an effective DR plan, as a DR plan depends on a solid backup plan to be effective. Like your DR plan, backup plans also need to be agreed upon by the appropriate levels of management, revisited regularly for possible updates and documented in a highly available, secure data store.

- Determine appropriate backup solutions for the different Azure services that are part of the critical paths within your workload.
- Define required retention periods for each different service.
- Understand that one tool may not work for everything. Azure Backup tools can cover many resource types but not all.
- Sometimes the best option to restore certain types of objects is a redeployment from some level type of highly-available repository. (Azure DevOps, GitHub or others)
- Data services will have different requirements than application related objects.
- Be sure to consider a multi-region storage strategy for your backup data to create cross-region recoverability.

- Run regular, scheduled test restores of backup data to ensure that services are working as expected.

Azure DR facilitation

Many Azure products have built-in failover capabilities. Familiarize yourself with these capabilities and include them in recovery procedures.

For IaaS (infrastructure as a service) systems, use Azure Site Recovery to automate failover and recovery. Refer to the following articles for common PaaS products:

- Azure App Service
- Azure Container Apps
- Azure Kubernetes Service
- Azure SQL Database
- Azure Event Hubs
- Azure Cache for Redis

Example

See the DR for Azure data platform series for guidance about preparing an enterprise data estate for DR.

Azure Backup facilitation

Many Azure products have built-in backup capabilities. Familiarize yourself with these capabilities and include them in recovery procedures.

For IaaS (infrastructure as a service) systems, use Azure Backup to facilitate backup of VMs and VM related services and some data services. Refer to the following articles for common products:

- Azure App Service
- Azure Kubernetes Service
- Azure SQL Database
- Azure Files

Related links

- Recommendations for designing for redundancy
- Recommendations for highly available multi-region design
- Recommendations for using availability zones and regions

Reliability checklist

Refer to the complete set of recommendations.

Reliability checklist

Feedback

Was this page helpful? Yes No

Recommendations for designing a reliable monitoring and alerting strategy

Article • 02/15/2024

Applies to this Azure Well-Architected Framework Reliability checklist recommendation:

Expand table

RE:10	**Measure and publish the solution's health indicators. Continuously capture uptime and other reliability data from across the workload and also from individual components and key flows.**

This guide describes the recommendations for designing a reliable monitoring and alerting strategy. Implement this strategy to keep your operations teams informed of your environment's health status and ensure that you meet the established reliability targets for your workload.

Definitions

Expand table

Term	Definition
Metrics	Numerical values that are collected at regular intervals. Metrics describe some aspects of a system at a particular time.
Resource logs	Data that a system generates. It provides information about the state of the system.
Traces	Data that provides information about the path that a request travels through services and components.

Key design strategies

Before you create a monitoring and alerting strategy, perform the following tasks for your workload as part of your reliability planning:

- Identify critical and noncritical flows.
- Perform failure mode analysis (FMA) for your flows.

- Identify reliability targets.
- Design for reliability by implementing redundancy, scaling, self-preservation, and self-healing.
- Design a robust testing strategy.

Create a monitoring and alerting strategy to ensure that your workload operates reliably. A monitoring and alerting strategy provides awareness to your operations teams so they're notified of changes in your workload's condition and can quickly address issues. Build a robust and reliable monitoring strategy on the health models that you develop for your critical flows and the workloads that the critical flows comprise. The health model defines healthy, degraded, and unhealthy states. Design your monitoring posture to immediately catch changes in these states. When health states change from healthy to degraded or unhealthy, alerting mechanisms trigger the automatic corrective measures and the alerts to the appropriate teams.

Implement the following recommendations to design a monitoring and alerting strategy that meets the requirements of your business.

General guidance

- Understand the difference between metrics, logs, and traces.
- Enable logging for all cloud resources. Use automation and governance in your deployments to enable diagnostic logging throughout your environment.
- Forward all diagnostic logs to a centralized data sink and analytics platform, like a Log Analytics workspace. If you have regional data sovereignty requirements, you must use local data sinks in the regions that are subject to those requirements.

>  **Tradeoff**: There are cost implications for storing and querying logs. Notice how your log analysis and retention affects your budget, and determine the best balance of utilization to meet your requirements. For more information, see Best practices for cost optimization.

- If your workloads are subject to one or more compliance frameworks, some of the component logs that handle sensitive information are also subject to those frameworks. Send the relevant component logs to a security information and event management (SIEM) system, like Microsoft Sentinel.
- Create a log retention policy that incorporates long-term retention requirements that the compliance frameworks impose on your workload.

- Use structured logging for all log messages to optimize querying the log data.
- Configure alerts to trigger when values pass critical thresholds that correlate to a health model state change, like green to yellow or red.

  Threshold configuration is a practice of continuous improvement. As your workload evolves, the thresholds you define might change. In some cases, dynamic thresholds are a good option for your monitoring strategy.
- Consider using alerts when states improve, such as red to yellow or red to green, so that the operations teams can track these events for future reference.
- Visualize the real-time health of your environment by using custom dashboards.
- Use data that's gathered during incidents to continuously improve your health models and your monitoring and alerting strategy.
- Incorporate cloud platform monitoring and alerting services, including:
  - Platform-level health, like Azure Service Health.
  - Resource-level health, like Azure Resource Health.
- Incorporate purpose-built advanced monitoring and analytics that your cloud provider offers, like Azure Monitor insight tools.
- Implement backup and recovery monitoring to capture:
  - The data replication status to ensure that your workload achieves recovery within the target recovery point objective (RPO).
  - Successful and failed backups and recoveries.
  - The recovery duration to inform your disaster recovery planning.

Monitor applications

- Create health probes or check functions and run them regularly from outside the application. Ensure that you test from multiple locations that are geographically close to your customers.
- Log data while the application runs in the production environment. You need sufficient information to diagnose the cause of issues in the production state.
- Log events at service boundaries. Include a correlation ID that flows across service boundaries. If a transaction flows through multiple services and one of them fails,

the correlation ID helps you track requests across your application and pinpoint why the transaction failed.

- Use asynchronous logging. Synchronous logging operations sometimes block your application code, which causes requests to back up as logs are written. Use asynchronous logging to preserve availability during application logging.
- Separate application logging from auditing. Audit records are commonly maintained for compliance or regulatory requirements and must be complete. To avoid dropped transactions, maintain audit logs separate from diagnostic logs.
- Use telemetry correlation to ensure that you can map transactions through the end-to-end application and critical system flows. This process is vital for performing root cause analysis (RCA) for failures. Collect platform-level metrics and logs, such as CPU percentage, network in, network out, and disk operations per second, from the application to inform a health model and to detect and predict issues. This approach can help distinguish between transient and nontransient faults.
- Use white box monitoring to instrument the application with semantic logs and metrics. Collect application-level metrics and logs, such as memory consumption or request latency, from the application to inform a health model and to detect and predict issues.
- Use black box monitoring to measure platform services and the resulting customer experience. Black box monitoring tests externally visible application behavior without knowledge of the internals of the system. This approach is common for measuring customer-centric service-level indicators (SLIs), service-level objectives (SLOs), and service-level agreements (SLAs).

> ⓘ **Note**
>
> For more information about application monitoring, see **Health Endpoint Monitoring pattern**.

Monitor data and storage

- Monitor the availability metrics of your storage containers. When this metric drops below 100 percent, it indicates failing writes. Transient drops in availability might happen when your cloud provider manages the load. Track the availability trends to determine if there's an issue with your workload.

In some cases, a drop in the availability metrics for a storage container indicates a bottleneck in the compute layer that's associated with the storage container.

- There are many metrics to monitor for databases. In the context of reliability, the important metrics to monitor include:
  - Query duration
  - Timeouts
  - Wait times
  - Memory pressure
  - Locks

Azure facilitation

- Azure Monitor is a comprehensive monitoring solution that's used to collect, analyze, and respond to monitoring data from your cloud and on-premises environments.
- Log Analytics is a tool in the Azure portal that's used to edit and run log queries against data in the Log Analytics workspace.
- Application Insights is an extension of Azure Monitor. It provides application performance monitoring (APM) features.
- Azure Monitor insights are advanced analytics tools that help monitor Azure services, like virtual machines, application services, and containers. Insights are built on top of Azure Monitor and Log Analytics.
- Azure Monitor for SAP solutions is an Azure-native monitoring product for SAP landscapes that run on Azure.
- Azure Policy helps to enforce organizational standards and to assess compliance at scale.
- Azure Business Continuity Center gives you insights into your business continuity estate. As you apply the approaches given for business continuity and disaster recovery (BCDR), use Azure Business Continuity Center to centralize management of business continuity protection across Azure and hybrid workloads. Azure Business Continuity Center pinpoints resources that lack proper protection (via backup or disaster recovery) and takes corrective actions. The tool facilitates

unified monitoring and lets you establish governance and auditing compliance through Azure Policy, all conveniently accessible in one location.

- For multiple workspace best practices, see Design a Log Analytics workspace architecture.

Example

For examples of real-world monitoring solutions, see Web application monitoring on Azure and Baseline architecture for an Azure Kubernetes Service cluster.

Related links

- Alerting for DevOps
- Alerting for operations
- Monitoring and diagnostics guidance
- Web application monitoring on Azure

Reliability checklist

Refer to the complete set of recommendations.

Reliability checklist

Feedback

Was this page helpful? Yes No

Security quick links

Apply security guidance to your architecture to help ensure the confidentiality, integrity, and availability of your data and systems.

Learn key points

QUICKSTART

Design principles

Checklist

Tradeoffs

Security patterns

Azure Well-Architected Review assessment

TRAINING

Security

VIDEO

Defense in depth security in Azure

Review design principles

CONCEPT

Plan your security readiness

Design to protect confidentiality

Design to protect integrity

Design to protect availability

Sustain and evolve your security posture

Create a security foundation

HOW-TO GUIDE

Establish a security baseline

Improve the security of your development lifecycle

Classify data

Monitor workload security

Model threats

Protect workload assets

HOW-TO GUIDE

Segment components

Manage identities and access

Protect the network

Use encryption

Harden resources

Guard application secrets

Validate and improve security

HOW-TO GUIDE

Perform security testing

Respond to incidents

Explore related resources

REFERENCE

Microsoft Defender for Cloud: Azure security recommendations

Microsoft Defender for Cloud: AWS recommendations

Microsoft Defender for Cloud: Google Cloud recommendations

Azure security documentation

Microsoft cloud security benchmark

Security design principles

Article • 11/15/2023

A Well-Architected workload must be built with a zero-trust approach. A secure workload is **resilient to attacks** and incorporates the interrelated security **principles of confidentiality, integrity, and availability** (also known as the *CIA triad*) in addition to meeting business goals. Any security incident has the potential to become a major breach that damages the brand and reputation of the workload or organization. To measure the security efficacy of your overall strategy for a workload, start with these questions:

- Do your defensive investments provide meaningful cost and friction to prevent attackers from compromising your workload?
- Will your security measures be effective in restricting the blast radius of an incident?
- Do you understand how controlling the workload could be valuable for an attacker? Do you understand the impact to your business if the workload and its data are stolen, unavailable, or tampered with?
- Can the workload and operations quickly detect, respond to, and recover from disruptions?

As you design your system, use the Microsoft Zero Trust model as the compass to mitigate security risks:

- **Verify explicitly** so that **only trusted identities** perform **intended and allowed actions** that originate from **expected locations**. This safeguard makes it harder for attackers to impersonate legitimate users and accounts.
- **Use least-privilege access** for the **right identities**, with the **right set of permissions**, for the **right duration**, and to the **right assets**. Limiting permissions helps keep attackers from abusing permissions that legitimate users don't even need.
- **Assume breach** of security controls and design compensating controls that **limit risk and damage** if a primary layer of defense fails. Doing so helps you to defend your workload better by thinking like an attacker who's interested in success (regardless of how they get it).

Security isn't a one-time effort. You must implement this guidance on a recurring basis. Continuously improve your defenses and security knowledge to help keep your

workload safe from attackers who are constantly gaining access to innovative attack vectors as they're developed and added to automated attack kits.

The design principles are intended to establish an ongoing security mindset to help you continuously improve the security posture of your workload as the attempts of attackers continuously evolve. These principles should guide the security of your architecture, design choices, and operational processes. Start with the recommended approaches and **justify the benefits for a set of security requirements**. After you set your strategy, drive actions by using the Security checklist as your next step.

If these principles aren't applied properly, a negative impact on business operations and revenue can be expected. Some consequences might be obvious, like penalties for regulatory workloads. Others might not be so obvious and could lead to ongoing security problems before they're detected.

In many mission-critical workloads, security is the primary concern, alongside reliability, given that some attack vectors, like data exfiltration, don't affect reliability. Security and reliability can pull a workload in opposite directions because security-focused design can introduce points of failure and increase operational complexity. The effect of security on reliability is often indirect, introduced by way of operational constraints. Carefully consider tradeoffs between security and reliability.

By following these principles, you can improve security effectiveness, harden workload assets, and build trust with your users.

Plan your security readiness

Expand table

Strive to adopt and implement security practices in architectural design decisions and operations with minimal friction.

As a workload owner, you have a shared responsibility with the organization to protect assets. Create a **security readiness plan** that's aligned with business priorities. It will lead to well-defined processes, adequate investments, and appropriate accountabilities. The plan should provide the workload requirements to the organization, which also shares responsibility for protecting assets. Security plans should factor into your strategy for reliability, health modeling, and self-preservation.

In addition to organizational assets, the workload itself needs to be protected from intrusion and exfiltration attacks. All facets of Zero Trust and the CIA triad should be factored into the plan.

Functional and non-functional requirements, budget constraints, and other considerations shouldn't restrict security investments or dilute assurances. At the same time, you need to engineer and plan security investments with those constraints and restrictions in mind.

Expand table

Approach	Benefit
Use segmentation as a strategy to plan security boundaries in the workload environment, processes, and team structure to **isolate access and function**. Your segmentation strategy should be driven by business requirements. You can base it on criticality of components, division of labor, privacy concerns, and other factors.	You'll be able to **minimize operational friction** by defining roles and establishing **clear lines of responsibility**. This exercise also helps you **identify the level of access** for each role, especially for critical-impact accounts. Isolation enables you to **limit exposure of sensitive flows** to only roles and assets that need access. Excessive exposure could inadvertently lead to information flow disclosure. To summarize, you'll be able to **right-size security efforts** based on the needs of each segment.
Continuously **build skills** through **role-based security training** that meets the requirements of the organization and the use cases of the workload.	A highly skilled team can design, implement, and monitor **security controls that remain effective** against attackers, who constantly look for new ways to exploit the system. Organization-wide training typically focuses on developing a broader skill set for securing the common elements. However, with role-based training, you focus on **developing deep expertise** in the platform offerings and security features that address workload concerns. You need to implement both approaches to defend against adversaries through **good design and effective operations**.
Make sure there's an incident response plan for your workload. Use industry frameworks that define the standard operating procedure for preparedness, detection, containment, mitigation, and post-incident activity.	At the time of crisis, confusion must be avoided. If you have a well-documented plan, responsible roles can **focus on execution** without wasting time on uncertain actions. Also, a comprehensive plan can help you ensure that **all remediation requirements are fulfilled**.
Strengthen your security posture by understanding the security compliance	Clarity about compliance requirements will help you **design for the right security assurances** and

Approach	Benefit
requirements that are imposed by influences outside the workload team, like organizational policies, regulatory compliance, and industry standards.	**prevent non-compliance** issues, which could lead to penalties. Industry standards can provide a baseline and influence your choice of tools, policies, security safeguards, guidelines, risk-management approaches, and training. If you know that the workload adheres to compliance, you'll be able to **instill confidence** in your user base.
Define and enforce team-level security standards across the lifecycle and operations of the workload. **Strive for consistent practices** in operations like coding, gated approvals, release management, and data protection and retention.	Defining good security practices can **minimize negligence** and the surface area for potential errors. The team will **optimize efforts and the outcome will be predictable** because approaches are made more consistent. Observing security standards over time will enable you to **identify opportunities for improvement, possibly including automation**, which will streamline efforts further and increase consistency.
Align your incident response with the **Security Operation Center (SOC) centralized function** in your organization.	Centralizing incident response functions enables you to take advantage of specialized IT professionals who can detect incidents in real time to address potential threats as quickly as possible.

Design to protect confidentiality

Expand table

Prevent exposure to privacy, regulatory, application, and proprietary information through access restrictions and obfuscation techniques.

Workload data can be classified by user, usage, configuration, compliance, intellectual property, and more. That data can't be shared or accessed beyond the established trust boundaries. Efforts to protect confidentiality should focus on access controls, opacity, and keeping an audit trail of activities that pertain to data and the system.

Expand table

Approach	Benefit
Implement **strong access controls** that grant access only on a need-to-know basis.	*Least privilege*. The workload will be protected from **unauthorized access** and prohibited activities. Even when access is from trusted identities, the **access permissions and exposure time will be minimized** because the communication path is open for a limited period.
Classify data based on its type, sensitivity, and potential risk. Assign a confidentiality level for each. Include system components that are in scope for the identified level.	*Verify explicitly*. This evaluation helps you right-size security measures. You'll also be able to identify data and components that have a **high potential impact** and/or exposure to risk. This exercise adds **clarity** to your information protection strategy and helps ensure **agreement**.
Safeguard your data at rest, in transit, and during processing by using **encryption**. Base your strategy on the assigned confidentiality level.	*Assume breach*. Even if an attacker gets access, they **won't be able to read properly encrypted** sensitive data. Sensitive data includes configuration information that's used to gain further access inside the system. Data encryption can help you **contain risks**.
Guard against exploits that might cause unwarranted exposure of information.	*Verify explicitly*. It's crucial to minimize vulnerabilities in authentication and authorization implementations, code, configurations, operations, and those that stem from the social habits of the system's users. Up-to-date security measures enable you to **block known security vulnerabilities** from entering the system. You can also **mitigate new vulnerabilities** that can appear over time by implementing routine operations throughout the development cycle, continuously improving security assurances.
Guard against data exfiltration that results from malicious or inadvertent access to data.	*Assume breach*. You'll be able to contain blast radius by **blocking unauthorized data transfer**. Additionally, controls applied to networking, identity, and encryption will protect data at various layers.

Approach	Benefit
Maintain the level of confidentiality as data flows through various components of the system.	*Assume breach*. Enforcing confidentiality levels throughout the system enables you to provide a consistent level of hardening. Doing so can **prevent vulnerabilities** that might result from moving data to a lower security tier.
Maintain an **audit trail** of all types of access activities.	*Assume breach*. Audit logs support **faster detection and recovery** in case of incidents and help with ongoing security monitoring.

Design to protect integrity

Expand table

Prevent corruption of design, implementation, operations, and data to avoid disruptions that can stop the system from delivering its intended utility or cause it to operate outside the prescribed limits. The system should provide information assurance throughout the workload lifecycle.

The key is to implement controls that prevent tampering of business logic, flows, deployment processes, data, and even the lower stack components, like the operating system and boot sequence. Lack of integrity can introduce vulnerabilities that can lead to breaches in confidentiality and availability.

Expand table

Approach	Benefit
Implement strong access controls that authenticate and authorize access to the system. **Minimize access based on privilege, scope, and time.**	*Least privilege*. Depending on the strength of the controls, you'll be able to **prevent or reduce risks from unapproved modifications**. This helps ensure that data is consistent and trustworthy. Minimizing access limits the extent of potential corruption.
Continuously protect against vulnerabilities and detect them in your supply chain to block attackers from injecting software faults	*Assume breach*. Knowing the origin of software and verifying its

Approach	Benefit
into your infrastructure, build system, tools, libraries, and other dependencies. Supply chain should scan for vulnerabilities during **build time and runtime**.	authenticity throughout the lifecycle will **provide predictability**. You'll **know about vulnerabilities well in advance** so that you can proactively remediate them and keep the system secure in production.
Establish trust and verify by using cryptography techniques like attestation, code signing, certificates, and encryption. Protect those mechanisms by allowing reputable decryption.	*Verify explicitly*, *least privilege*. You'll know that changes to data or access to the system **is verified by a trusted source**. Even if encrypted data is intercepted in transit by a malicious actor, the actor won't be able to unlock or decipher the content. You can use digital signatures to ensure that the data wasn't tampered with during transmission.
Ensure backup data is immutable and encrypted when data is replicated or transferred.	*Verify explicitly*. You'll be able to recover data with confidence that backup **data wasn't changed at rest**, inadvertently or maliciously.
Avoid or mitigate system implementations that allow your workload to operate outside its intended limits and purposes.	*Verify explicitly*. When your system has strong safeguards that check whether usage aligns with its intended limits and purposes, the scope for potential abuse or tampering of your compute, networking, and data stores is reduced.

Design to protect availability

Expand table

Prevent or minimize system and workload downtime and degradation in the event of a security incident by using strong security controls. You must maintain data integrity during the incident and after the system recovers.

You need to balance availability architecture choices with security architecture choices. The system should have availability guarantees to ensure that users have access to data and that data is reachable. From a security perspective, users should operate within the allowed access scope, and the data must be trusted. Security controls should block bad actors, but they shouldn't block legitimate users from accessing the system and data.

Expand table

Approach	Benefit
Prevent compromised identities from misusing access to gain control of the system. Check for **overly pervasive scope and time limits** to minimize risk exposure.	*Least privilege*. This strategy **mitigates the risks of excessive, unnecessary, or misused access permissions** on crucial resources. Risks include unauthorized modifications and even the deletion of resources. Take advantage of the platform-provided just-in-time (JIT), just-enough-access (JEA), and time-based security modes to replace standing permissions wherever possible.
Use security controls and design patterns to **prevent attacks and code flaws from causing resource exhaustion** and blocking access.	*Verify explicitly*. The **system won't experience downtime** caused by malicious actions, like distributed denial of service (DDoS) attacks.
Implement **preventative measures for attack vectors that exploit vulnerabilities** in application code, networking protocols, identity systems, malware protection, and other areas.	*Assume breach*. Implement code scanners, apply the latest security patches, update software, and protect your system with effective antimalware on an ongoing basis. You'll be able to reduce the attack surface to ensure business continuity.
Prioritize security controls on the **critical components and flows** in the system that are susceptible to risk.	*Assume breach*, *verify explicitly*. Regular detection and prioritization exercises can help you **apply security expertise to the critical aspects** of the system. You'll be able to focus on the most likely and damaging threats and start your risk mitigation in areas that need the most attention.
Apply at least the same level of **security rigor in your recovery resources and processes** as you do in the primary environment, including security controls and frequency of backup.	*Assume breach*. You should have a preserved safe system state available in disaster recovery. If you do, you can fail over to a secure secondary system or location and restore backups that won't introduce a threat. A well-designed process can prevent a security incident from hindering the recovery process. Corrupted backup data or encrypted data that can't be deciphered can slow down recovery.

Sustain and evolve your security posture

Expand table

Incorporate continuous improvement and apply vigilance to stay ahead of attackers who are continuously evolving their attack strategies.

Your security posture must not degrade over time. You must continually improve security operations so that new disruptions are handled more efficiently. Strive to align improvements with the phases defined by industry standards. Doing so leads to better preparedness, reduced time to incident detection, and effective containment and mitigation. Continuous improvement should be based on lessons learned from past incidents.

It's important to measure your security posture, enforce policies to maintain that posture, and regularly validate your security mitigations and compensating controls in order to continuously improve your security posture in the face of evolving threats.

Expand table

Approach	Benefit
Create and maintain a comprehensive asset inventory that includes classified information about resources, locations, dependencies, owners, and other metadata that's relevant to security. As much as possible, **automate** inventory to derive data from the system.	A well-organized inventory provides a **holistic view of the environment**, which puts you in an advantageous position against attackers, especially during post-incident activities. It also creates a business rhythm to drive communication, upkeep of critical components, and the decommissioning of orphaned resources.
Perform threat modeling to identify and mitigate potential threats.	You'll have a **report of attack vectors** prioritized by their severity level. You'll be able to identify threats and vulnerabilities quickly and set up countermeasures.
Regularly **capture data to quantify your current state** against your established security baseline and **set priorities for remediations**. Take advantage of platform-provided features for **security posture management** and **the enforcement of**	You need accurate reports that bring clarity and consensus to focus areas. You'll be able to immediately **execute technical remediations**, starting with the highest priority items. You'll also **identify gaps**, which provide opportunities for improvement. Implementing enforcement helps prevent violations

Approach	Benefit
compliance imposed by external and internal organizations.	and regressions, which preserves your security posture.
Run periodic security tests that are conducted by experts external to the workload team who attempt to ethically hack the system. Perform routine and integrated **vulnerability scanning** to detect exploits in infrastructure, dependencies, and application code.	These tests enable you to validate security defenses by **simulating real-world attacks** by using techniques like penetration testing. Threats can be introduced as part of your change management. Integrating scanners into the deployment pipelines enables you to automatically detect vulnerabilities and even quarantine usage until the vulnerabilities are removed.
Detect, respond, and recover with swift and effective security operations.	The primary benefit of this approach is that it enables you to **preserve or restore the security assurances of the CIA triad** during and after an attack. You need to be alerted as soon as a threat is detected so that you can start your investigations and take appropriate actions.
Conduct post-incident activities like root-cause analyses, postmortems, and incident reports.	These activities provide insight into the impact of the breach and into resolution measures, which drives improvements in defenses and operations.
Get current, and stay current. Stay current on updates, patching, and security fixes. Continuously evaluate the system and improve it based on audit reports, benchmarking, and lessons from testing activities. Consider automation, as appropriate. Use threat intelligence powered by security analytics for dynamic detection of threats. At regular intervals, review the workload's conformance to Security Development Lifecycle (SDL) best practices.	You'll be able to ensure that your **security posture doesn't degrade over time**. By integrating findings from real-world attacks and testing activities, you'll be able to combat attackers who continuously improve and exploit new categories of vulnerabilities. Automation of repetitive tasks **decreases the chance of human error** that can create risk. SDL reviews bring clarity around security features. SDL can help you maintain an inventory of workload assets and their security reports, which cover origin, usage, operational weaknesses, and other factors.

Next steps

Security checklist

Design review checklist for Security

Article • 11/14/2023

This checklist presents a set of security recommendations to help you ensure your workload is secure and aligned with the Zero Trust model. If you haven't checked the following boxes and considered the tradeoffs, then your design might be at risk. Carefully consider all of the points covered in the checklist to gain confidence in your workload's security.

Checklist

	Code	Recommendation
☐	SE:01	**Establish a security baseline** that's aligned to compliance requirements, industry standards, and platform recommendations. Regularly measure your workload architecture and operations against the baseline to sustain or improve your security posture over time.
☐	SE:02 SE:02	**Maintain a secure development lifecycle** by using a hardened, mostly automated, and auditable software supply chain. Incorporate a secure design by using threat modeling to safeguard against security-defeating implementations.
☐	SE:03	**Classify and consistently apply sensitivity and information type labels** on all workload data and systems involved in data processing. Use classification to influence workload design, implementation, and security prioritization.
☐	SE:04	**Create intentional segmentation and perimeters** in your architecture design and in the workload's footprint on the platform. The segmentation strategy must include networks, roles and responsibilities, workload identities, and resource organization.
☐	SE:05	**Implement strict, conditional, and auditable identity and access management (IAM)** across all workload users, team members, and system components. Limit access exclusively to *as necessary*. Use modern industry standards for all authentication and authorization implementations. Restrict and rigorously audit access that's not based on identity.
☐	SE:06	**Isolate, filter, and control network traffic** across both ingress and egress flows. Apply defense-in-depth principles by using localized network controls at all available network boundaries across both east-west and north-south traffic.
☐	SE:07	**Encrypt data by using modern, industry-standard methods** to guard confidentiality and integrity. Align the encryption scope with data classifications, and prioritize native platform encryption methods.
☐	SE:08	**Harden all workload components** by reducing extraneous surface area and tightening configurations to increase attacker cost.

	Code	Recommendation
☐	SE:09	**Protect application secrets** by hardening their storage and restricting access and manipulation and by auditing those actions. Run a reliable and regular rotation process that can improvise rotations for emergencies.
☐	SE:10	**Implement a holistic monitoring strategy** that relies on modern threat detection mechanisms that can be integrated with the platform. Mechanisms should reliably alert for triage and send signals into existing SecOps processes.
☐	SE:11	**Establish a comprehensive testing regimen** that combines approaches to prevent security issues, validate threat prevention implementations, and test threat detection mechanisms.
☐	SE:12	**Define and test effective incident response procedures** that cover a spectrum of incidents, from localized issues to disaster recovery. Clearly define which team or individual runs a procedure.

Next steps

We recommend that you review the Security tradeoffs to explore other concepts.

Security tradeoffs

Security tradeoffs

Article • 10/10/2024

Security provides confidentiality, integrity, and availability assurances of a workload's system and its users' data. Security controls are required for the workload and for the software development and operational components of the system. When teams design and operate a workload, they can almost never compromise on security controls.

During the design phase of a workload, it's important to consider how decisions based on the Security design principles and the recommendations in the Design review checklist for Security might influence the goals and optimizations of other pillars. Certain security decisions might benefit some pillars but constitute tradeoffs for others. This article describes example tradeoffs that a workload team might encounter when establishing security assurances.

Security tradeoffs with Reliability

>  **Tradeoff: Increased complexity.** The Reliability pillar prioritizes simplicity and recommends that points of failure are minimized.

- Some security controls can increase the risk of misconfiguration, which can lead to service disruption. Examples of security controls that can introduce misconfiguration include network traffic rules, identity providers, virus scanning exclusions, and role-based or attribute-based access control assignments.
- Increased segmentation usually results in a more complex environment in terms of resource and network topology and operator access. This complexity can lead to more points of failure in processes and in workload execution.
- Workload security tooling is often incorporated into many layers of a workload's architecture, operations, and runtime requirements. These tools might affect resiliency, availability, and capacity planning. Failure to account for limitations in the tooling can lead to a reliability event, like SNAT port exhaustion on an egress firewall.

>  **Tradeoff: Increased critical dependencies.** The Reliability pillar recommends minimizing critical dependencies. A workload that minimizes critical dependencies, especially external ones, has more control over its points of failure.

The Security pillar requires a workload to explicitly verify identities and actions. Verification occurs via critical dependencies on key security components. If those components aren't available or if they malfunction, verification might not complete. This failure puts the workload in a degraded state. Some examples of these critical single-point-of-failure dependencies are:

- Ingress and egress firewalls.
- Certificate revocation lists.
- Accurate system time provided by a Network Time Protocol (NTP) server.
- Identity providers, like Microsoft Entra ID.

>  **Tradeoff: Increased complexity of disaster recovery.** A workload must reliably recover from all forms of disaster.

- Security controls might affect recovery time objectives. This effect can be caused by the additional steps that are needed to decrypt backed up data or by operational access delays created by site reliability triage.
- Security controls themselves, for example secret vaults and their contents or edge DDoS protection, need to be part of the disaster recovery plan of the workload and must be validated via recovery drills.
- Security or compliance requirements might limit data residency options or access control restrictions for backups, potentially further complicating recovery by segmenting even offline replicas.

>  **Tradeoff: Increased rate of change.** A workload that experiences runtime change is exposed to more risk of reliability impact due to that change.

- Stricter patching and update policies lead to more changes in a workload's production environment. This change comes from sources like these:
  - Application code being released more frequently because of updates to libraries or updates to base container images
  - Increased routine patching of operating systems
  - Staying current with versioned applications or data platforms
  - Applying vendor patches to software in the environment
- Rotation activities for keys, service principal credentials, and certificates increase the risk of transient issues due to the timing of the rotation and clients using the new value.

Security tradeoffs with Cost Optimization

>  **Tradeoff: Additional infrastructure.** One approach to cost optimizing a workload is to look for ways to reduce the diversity and number of components and increase density.

Some workload components or design decisions exist only to protect the security (confidentiality, integrity, and availability) of systems and data. These components, although they enhance the security of the environment, also increase costs. They must also be subject to cost optimization themselves. Some example sources for these security-centric additional resources or licensing costs are:

- Compute, network, and data segmentation for isolation, which sometimes involves running separate instances, preventing co-location and reducing density.
- Specialized observability tooling, like a SIEM that can perform aggregation and threat intelligence.
- Specialized networking appliances or capabilities, like firewalls or distributed denial-of-service prevention.
- Data classification tools that are required for capturing sensitivity and information-type labels.
- Specialized storage or compute capabilities to support encryption at rest and in transit, like an HSM or confidential-compute functionality.
- Dedicated testing environments and testing tools to validate that security controls are functioning and to uncover previously undiscovered gaps in coverage.

The preceding items often also exist outside of production environments, in preproduction and disaster recovery resources.

>  **Tradeoff: Increased demand on infrastructure.** The Cost Optimization pillar prioritizes driving down demand on resources to enable the use of cheaper SKUs, fewer instances, or reduced consumption.

- *Premium SKUs*: Some security measures in cloud and vendor services that can benefit the security posture of a workload might only be found in more expensive SKUs or tiers.

- *Log storage*: High fidelity security monitoring and audit data that provide broad coverage increase storage costs. Security observability data is also often stored for longer periods of time than would typically be needed for operational insights.

- *Increased resource consumption*: In-process and on-host security controls can introduce additional demand for resources. Encryption for data at rest and in transit can also increase demand. Both scenarios can require higher instance counts or larger SKUs.

> **Tradeoff: Increased process and operational costs.** Personnel process costs are part of the overall total cost of ownership and are factored into a workload's return on investment. Optimizing these costs is a recommendation of the Cost Optimization pillar.

- A more comprehensive and strict patch management regime leads to an increase in time and money spent on these routine tasks. This increase is often coupled with the expectation of investing in preparedness for ad hoc patching for zero-day exploits.
- Stricter access controls to reduce the risk of unauthorized access can lead to more complex user management and operational access.
- Training and awareness for security tools and processes take up employee time and also incur costs for materials, instructors, and possibly training environments.
- Complying with regulations might necessitate additional investments for audits and generating compliance reporting.
- Planning for and conducting drills for security-incident response preparedness takes time.
- Time needs to be allocated for designing and performing routine and ad hoc processes that are associated with security, like key or certificate rotation.
- The security validation of the SDLC usually requires specialized tools. Your organization might need to pay for these tools. Prioritizing and remediating issues found during testing also takes time.
- Hiring third-party security practitioners to perform white-box testing or testing that's performed without the knowledge of a system's internal workings (sometimes known as *black-box testing*), including penetration testing, incurs costs.

Security tradeoffs with Operational Excellence

> **Tradeoff: Complications in observability and serviceability.** Operational Excellence requires architectures to be serviceable and observable. The most

> serviceable architectures are those that are the most transparent to everyone involved.

- Security benefits from extensive logging that provides high fidelity insight into the workload for alerting on deviations from baselines and for incident response. This logging can generate a significant volume of logs, which can make it harder to provide insights that are targeted at reliability or performance.
- When compliance guidelines for data masking are followed, specific segments of logs or even large amounts of tabular data are redacted to protect confidentiality. The team needs to evaluate how this observability gap might affect alerting or hinder incident response.
- Strong resource segmentation increases the complexity of observability by requiring additional cross-service distributed tracing and correlation for capturing flow traces. The segmentation also increases the surface area of compute and data to service.
- Some security controls impede access by design. During incident response, these controls can slow down workload operators' emergency access. Therefore, incident response plans need to include more emphasis on planning and drills in order to reach acceptable efficacy.

>  **Tradeoff: Decreased agility and increased complexity.** Workload teams measure their velocity so that they can improve the quality, frequency, and efficiency of delivery activities over time. Workload complexity factors into the effort and risk involved in operations.

- Stricter change control and approval policies to reduce the risk of introducing security vulnerabilities can slow down the development and safe deployment of new features. However, the expectation of addressing security updates and patching can increase demand for more frequent deployments. Additionally, human-gated approval policies in operational processes can make it more difficult to automate those processes.
- Security testing results in findings that need to be prioritized, potentially blocking planned work.
- Routine, ad hoc, and emergency processes might require audit logging to meet compliance requirements. This logging increases the rigidity of running the processes.

- Workload teams might increase the complexity of identity management activities as the granularity of role definitions and assignments is increased.
- An increased number of routine operational tasks that are associated with security, like certificate management, increases the number of processes to automate.

>  **Tradeoff: Increased coordination efforts.** A team that minimizes external points of contact and review can control their operations and timeline more effectively.

- As external compliance requirements from the larger organization or from external entities increase, the complexity of achieving and proving compliance with auditors also increases.
- Security requires specialized skills that workload teams don't typically have. Those proficiencies are often sourced from the larger organization or from third parties. In both cases, coordination of effort, access, and responsibility needs to be established.
- Compliance or organizational requirements often require maintained communication plans for responsible disclosure of breaches. These plans must be factored into security coordination efforts.

Security tradeoffs with Performance Efficiency

>  **Tradeoff: Increased latency and overhead.** A performant workload reduces latency and overhead.

- Inspection security controls, like firewalls and content safety filters, are located in the flows that they secure. Those flows are therefore subject to additional verification, which adds latency to requests. In highly decoupled architecture the distributed nature can lead to these inspections happening multiple times for a single user or data flow transaction.
- Identity controls require each invocation of a controlled component to be verified explicitly. This verification consumes compute cycles and might require network traversal for authorization.
- Encryption and decryption require dedicated compute cycles. These cycles increase the time and resources consumed by those flows. This increase is usually

correlated with the complexity of the algorithm and the generation of high-entropy and diverse initialization vectors (IVs).

- As the extensiveness of logging increases, the impact on system resources and network bandwidth for streaming those logs can also increase.
- Resource segmentation frequently introduces network hops in a workload's architecture.

>  **Tradeoff: Increased chance of misconfiguration.** Reliably meeting performance targets depends on predictable implementations of the design.

A misconfiguration or overextension of security controls can impact performance because of inefficient configuration. Examples of security control configurations that can affect performance include:

- Firewall rule ordering, complexity, and quantity (granularity).
- Failing to exclude key files from file integrity monitors or virus scanners. Neglecting this step can lead to lock contention.
- Web application firewalls performing deep packet inspection for languages or platforms that are irrelevant for the components that are being protected.

Related links

Explore the tradeoffs for the other pillars:

- Reliability tradeoffs
- Cost Optimization tradeoffs
- Operational Excellence tradeoffs
- Performance Efficiency tradeoffs

Feedback

Was this page helpful? Yes No

Cloud design patterns that support security

Article • 10/10/2024

When you design workload architectures, you should use industry patterns that address common challenges. Patterns can help you make intentional tradeoffs within workloads and optimize for your desired outcome. They can also help mitigate risks that originate from specific problems, which can affect reliability, performance, cost, and operations. Those risks might be indicative of lack of security assurances, if left unattended can pose significant risks to the business. These patterns are backed by real-world experience, are designed for cloud scale and operating models, and are inherently vendor agnostic. Using well-known patterns as a way to standardize your workload design is a component of operational excellence.

Many design patterns directly support one or more architecture pillars. Design patterns that support the Security pillar prioritize concepts like segmentation and isolation, strong authorization, uniform application security, and modern protocols.

Design patterns for security

The following table summarizes cloud design patterns that support the goals of security.

Expand table

Pattern	Summary
Ambassador	Encapsulates and manages network communications by offloading cross-cutting tasks that are related to network communication. The resulting helper services initiate communication on behalf of the client. This mediation point provides an opportunity to augment security on network communications.
Backends for Frontends	Individualizes the service layer of a workload by creating separate services that are exclusive to a specific frontend interface. Because of this separation, the security and authorization in the service layer that support one client can be tailored to the functionality provided by that client, potentially reducing the surface area of an API and lateral movement among different backends that might expose different capabilities.
Bulkhead	Introduces intentional and complete segmentation between components to isolate the blast radius of malfunctions. You can also use this strategy to contain security incidents to the compromised bulkhead.

Pattern	Summary
Claim Check	Separates data from the messaging flow, providing a way to separately retrieve the data related to a message. This pattern supports keeping sensitive data out of message bodies, instead keeping it managed in a secured data store. This configuration enables you to establish stricter authorization to support access to the sensitive data from services that are expected to use the data, but remove visibility from ancillary services like queue monitoring solutions.
Federated Identity	Delegates trust to an identity provider that's external to the workload for managing users and providing authentication for your application. By externalizing user management and authentication, you can get evolved capabilities for identity-based threat detection and prevention without needing to implement these capabilities in your workload. External identity providers use modern interoperable authentication protocols.
Gatekeeper	Offloads request processing that's specifically for security and access control enforcement before and after forwarding the request to a backend node. Adding a gateway into the request flow enables you to centralize security functionality like web application firewalls, DDoS protection, bot detection, request manipulation, authentication initiation, and authorization checks.
Gateway Aggregation	Simplifies client interactions with your workload by aggregating calls to multiple backend services in a single request. This topology often reduces the number of touch points a client has with a system, which reduces the public surface area and authentication points. The aggregated backends can stay fully network-isolated from clients.
Gateway Offloading	Offloads request processing to a gateway device before and after forwarding the request to a backend node. Adding a gateway into the request flow enables you to centralize security functionality like web application firewalls and TLS connections with clients. Any offloaded functionality that's platform-provided already offers enhanced security.
Publisher/Subscriber	Decouples components in an architecture by replacing direct client-to-service communication with communication via an intermediate message broker or event bus. This replacement introduces an important security segmentation boundary that enables queue subscribers to be network-isolated from the publisher.
Quarantine	Ensures external assets meet a team-agreed quality level before being authorized to consume them in the workload. This check serves as a first security validation of external artifacts. The validation on an artifact is conducted in a segmented environment before it's used within the secure development lifecycle (SDL).
Sidecar	Extends the functionality of an application by encapsulating nonprimary or cross-cutting tasks in a companion process that exists alongside the main application. By encapsulating these tasks and deploying them out-of-

Pattern	Summary
	process, you can reduce the surface area of sensitive processes to only the code that's needed to accomplish the task. You can also use sidecars to add cross-cutting security controls to an application component that's not natively designed with that functionality.
Throttling	Imposes limits on the rate or throughput of incoming requests to a resource or component. You can design the limits to help prevent resource exhaustion that could result from automated abuse of the system.
Valet Key	Grants security-restricted access to a resource without using an intermediary resource to proxy the access. This pattern enables a client to directly access a resource without needing long-lasting or standing credentials. All access requests start with an auditable transaction. The granted access is then limited in both scope and duration. This pattern also makes it easier to revoke the granted access.

Next steps

Review the cloud design patterns that support the other Azure Well-Architected Framework pillars:

- Cloud design patterns that support reliability
- Cloud design patterns that support cost optimization
- Cloud design patterns that support operational excellence
- Cloud design patterns that support performance efficiency

Feedback

Was this page helpful? Yes No

Recommendations for establishing a security baseline

Article • 11/14/2023

Applies to Azure Well-Architected Framework Security checklist recommendation:

Expand table

SE:01	**Establish a security baseline aligned to compliance requirements, industry standards, and platform recommendations. Regularly measure your workload architecture and operations against the baseline to sustain or improve your security posture over time.**

This guide describes the recommendations for establishing a security baseline. A security baseline is a document that specifies your organization's bare minimum security requirements and expectations across a range of areas. A good security baseline helps you:

- Keep your data and systems secure.
- Comply with regulatory requirements.
- Minimize risk of oversight.
- Reduce the likelihood of breaches and subsequent business effects.

Security baselines should be published widely throughout your organization so that all stakeholders are aware of the expectations.

This guide provides recommendations about setting a security baseline that's based on internal and external factors. Internal factors include business requirements, risks, and asset evaluation. External factors include industry benchmarks and regulatory standards.

Definitions

Expand table

Term	Definition
Baseline	The minimum level of security affordances that a workload must have to avoid being exploited.
Benchmark	A standard that signifies the security posture that the organization aspires to. It's evaluated, measured, and improved over time.
Controls	Technical or operational controls on the workload that help prevent attacks

Term	Definition
	and increase attacker costs.
Regulatory requirements	A set of business requirements, driven by industry standards, that laws and authorities impose.

Key design strategies

A security baseline is a structured document that defines a set of security criteria and capabilities that the workload must fulfill in order to increase security. In a more mature form, you can extend a baseline to include a set of policies that you use to set guardrails.

The baseline should be considered the standard for measuring your security posture. The goal should always be full attainment while keeping a broad scope.

Your security baseline should never be an ad-hoc effort. Industry standards, compliance (internal or external) or regulatory requirements, regional requirements, and the cloud platform benchmarks are main drivers for the baseline. Examples include Center for Internet Security (CIS) Controls, National Institute of Standards and Technology (NIST), and platform-driven standards, such as Microsoft cloud security benchmark (MCSB). All of these standards are considered a starting point for your baseline. Build the foundation by incorporating security requirements from the business requirements.

For links to the preceding assets, see Related links.

Create the baseline by gaining consensus among business and technical leaders. The baseline shouldn't be restricted to technical controls. It should also include the operational aspects of managing and maintaining the security posture. So, the baseline document also serves as the organization's commitment to investment toward workload security. The security baseline document should be distributed widely within your organization to ensure there's awareness about the workload's security posture.

As the workload grows and the ecosystem evolves, it's vital to keep your baseline in synch with the changes to ensure the fundamental controls are still effective.

Creating a baseline is a methodical process. Here are some recommendations about the process:

- **Asset inventory**. Identify stakeholders of workload assets and the security objectives for those assets. In the asset inventory, classify by security requirements and criticality. For information about data assets, see Recommendations on data classification.

- **Risk assessment**. Identity potential risks associated with each asset and prioritize them.
- **Compliance requirements**. Baseline any regulatory or compliance for those assets and apply industry best practices.
- **Configuration standards**. Define and document specific security configurations and settings for each asset. If possible, templatize or find a repeatable, automated way to apply the settings consistently across the environment.
- **Access control and authentication**. Specify the role-based access control (RBAC) and multifactor authentication (MFA) requirements. Document what *just enough access* means at the asset level. Always start with the principle of least privilege.
- **Patch management**. Apply latest versions on all the resource types to strengthen against attack.
- **Documentation and communication**. Document all configurations, policies, and procedures. Communicate the details to the relevant stakeholders.
- **Enforcement and accountability**. Establish clear enforcement mechanisms and consequences for noncompliance with the security baseline. Hold individuals and teams accountable for maintaining security standards.
- **Continuous monitoring**. Assess the effectiveness of the security baseline through observability and make improvements overtime.

Define the baseline

Here are some common categories that should be part of a baseline. The following list isn't exhaustive. It's intended as an overview of the document's scope.

Regulatory compliance

A workload might be subject to regulatory compliance for specific industry segments, there might be some geographic restrictions, and so on. It's key to understand the requirements as given in the regulatory specifications because those influence the design choices and in some cases must be included in the architecture.

The baseline should include regular evaluation of the workload against regulatory requirements. Take advantage of the platform-provided tools, such as Microsoft Defender for Cloud, which can identify areas of noncompliance. Work with the organization's compliance team to make sure all requirements are met and maintained.

Architecture components

The baseline needs prescriptive recommendations for the main components of the workload. These usually include technical controls for networking, identity, compute, and data. Reference the security baselines provided by the platform and add the missing controls to the architecture.

> Refer to Example.

Development processes

The baseline must have recommendations about:

- System classification.
- The approved set of resource types.
- Tracking the resources.
- Enforcing policies for using or configuring resources.

The development team needs to have a clear understanding of the scope for security checks. For example, threat modeling is a requirement in making sure that potential threats are identified in code and in deployment pipelines. Be specific about static checks and vulnerability scanning in your pipeline and how regularly the team needs to perform those scans.

For more information, see Recommendations on threat analysis.

The development process should also set standards on various testing methodologies and their cadence. For more information, see Recommendations on security testing.

Operations

The baseline must set standards on using threat detection capabilities and raising alerts on anomalous activities that indicate actual incidents. Threat detection needs to include all layers of the workload, including all the endpoints that are reachable from hostile networks.

The baseline should include recommendations for setting up incident response processes, including communication and a recovery plan, and which of those processes can be automated to expedite detection and analysis. For examples, see Security baselines for Azure overview.

The incident response should also include a recovery plan and the requirements for that plan, such as resources for regularly taking and protecting backups.

You develop data breach plans by using industry standards and recommendations provided by the platform. The team then has a comprehensive plan to follow when a breach is discovered. Also, check with your organization to see if there's coverage through cyberinsurance.

Training

Develop and maintain a security training program to ensure the workload team is equipped with the appropriate skills to support the security goals and requirements. The team needs fundamental security training, but use what you can from your organization to support specialized roles. Role-based security training compliance and participation in drills are part of your security baseline.

Apply the baseline

Use the baseline to drive initiatives, such as:

- **Preparedness toward design decisions**. Create the security baseline and publish it before you start the architecture design process. Ensure team members are fully aware of your organization's expectations early, which avoids costly rework caused by a lack of clarity. You can use baseline criteria as workload requirements that the organization has committed to and design and validate controls against those constraints.

- **Measure your design**. Grade the current decisions against the current baseline. The baseline sets actual thresholds for criteria. Document any deviations that are deferred or deemed long-term acceptable.

- **Drive improvements**. While the baseline sets attainable goals, there are always gaps. Prioritize the gaps in your backlog and remediate based on prioritization.

- **Track your progress against the baseline**. Continuous monitoring of security measures against a set baseline is essential. Trend analysis is a good way of reviewing security progress over time and can reveal consistent deviations from the baseline. Use automation as much as possible, pulling data from various sources, internal and external, to address current issues and prepare for future threats.

- **Set guardrails**. Where possible, your baseline criteria must have guardrails. Guardrails enforce required security configurations, technologies, and operations, based on internal factors and external factors. Internal factors include business requirements, risks, and asset evaluation. External factors include benchmarks,

regulatory standards, and threat environment. Guardrails help minimize the risk of inadvertent oversight and punitive fines for noncompliance.

Explore Azure Policy for custom options or use built-in initiatives like CIS benchmarks or Azure Security Benchmark to enforce security configurations and compliance requirements. Consider creating Azure Policies and initiatives out of baselines.

Evaluate the baseline regularly

Continuously improve security standards incrementally towards the ideal state to ensure continual risk reduction. Conduct periodic reviews to ensure that the system is up-to-date and in compliance with external influences. Any change to the baseline must be formal, agreed upon, and sent through proper change management processes.

Measure the system against the new baseline and prioritize remediations based on their relevance and effect on the workload.

Ensure that the security posture doesn't degrade over time by instituting auditing and monitoring compliance with organizational standards.

Azure facilitation

The Microsoft cloud security benchmark (MCSB) is a comprehensive security best practice framework that you can use as a starting point for your security baseline. Use it along with other resources that provide input to your baseline.

For more information, see Introduction to the Microsoft cloud security benchmark.

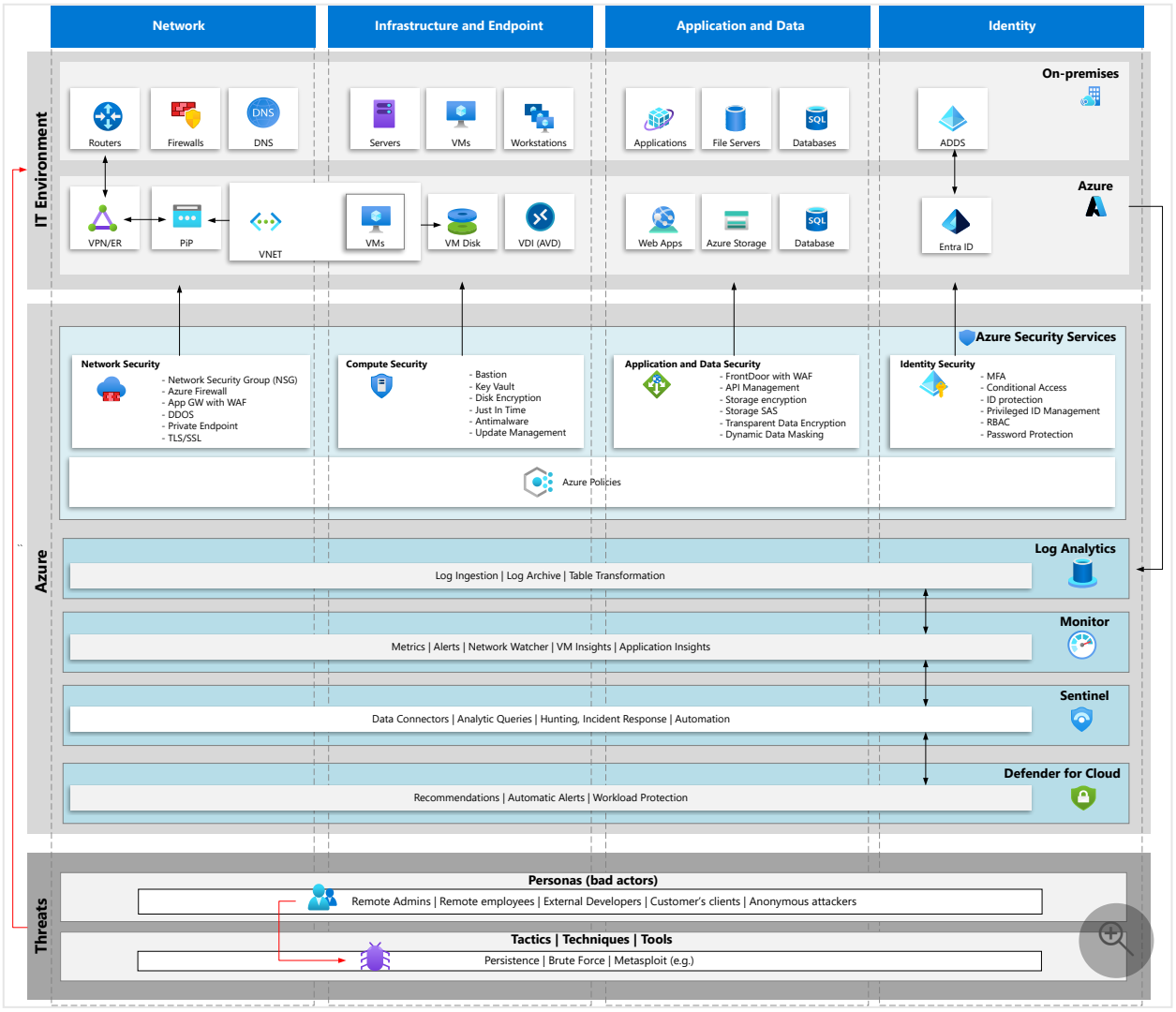
Use the Microsoft Defender for Cloud (MDC) regulatory compliance dashboard to track those baselines and be alerted if a pattern outside of a baseline is detected. For more information, see the Customize the set of standards in your regulatory compliance dashboard.

Other features that help in establishing and improving the baseline:

- Create custom Azure security policies
- Understand security policies, initiatives, and recommendations
- Regulatory compliance checks

Example

This logical diagram shows an example security baseline for architectural components that encompass network, infrastructure, endpoint, application, data, and identity to demonstrate how a common IT environment may be securely protected. Other recommendation guides build on this example.



Infrastructure

A common IT environment, with an on-premises layer with basic resources.

Azure Security services

Azure security services and features by the types of resources they protect.

Azure security monitoring services

The monitoring services available on Azure that go beyond simple monitoring services, including security information event management (SIEM) and security orchestration automated response (SOAR) solutions and Microsoft Defender for Cloud.

Threats

This layer brings a recommendation and reminder that threats may be mapped according to your organization's concerns regarding threats, regardless of the methodology or matrix-like Mitre Attack Matrix or Cyber Kill chain.

Related links

- Microsoft compliance
- Security baselines for Azure overview
- What is incident response? Plan and steps
- Azure Security benchmarks

Community links

- CIS Microsoft Azure Foundations Benchmark
- Cybersecurity framework | NIST

Security checklist

Refer to the complete set of recommendations.

Security checklist

Feedback

Was this page helpful? Yes No

Recommendations for securing a development lifecycle

Article • 08/30/2024

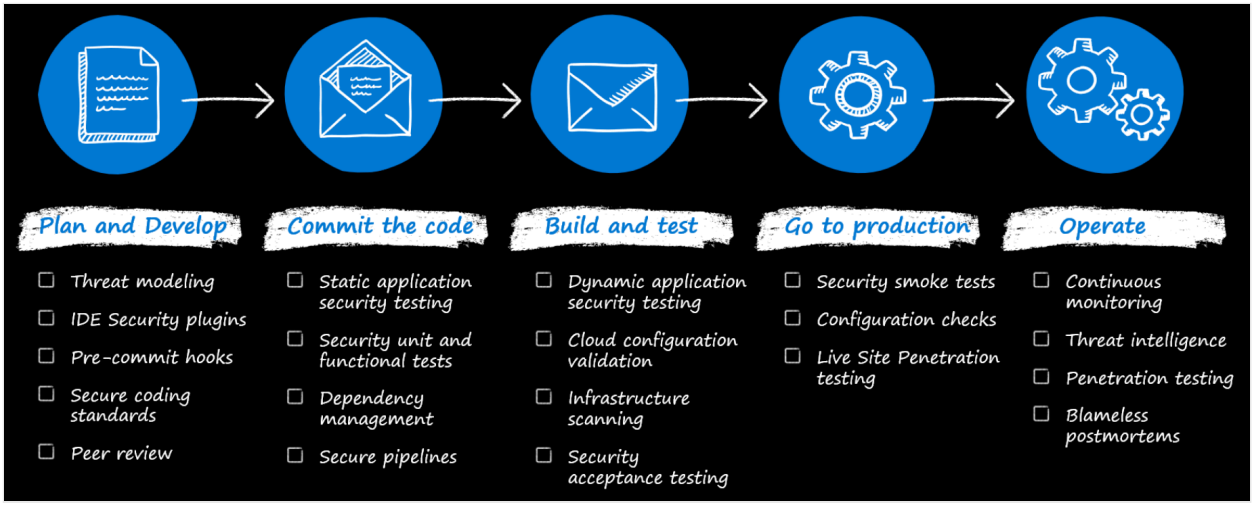
Applies to this Azure Well-Architected Framework Security checklist recommendation:

Expand table

SE:02	**Maintain a secure development lifecycle by using a hardened, mostly automated, and auditable software supply chain. Incorporate a secure design by using threat modeling to safeguard against security-defeating implementations.**

Related guide: Threat analysis

This guide describes the **recommendations for hardening your code, development environment, and software supply chain** by applying security best practices throughout the development cycle. To understand this guidance, you should have knowledge of DevSecOps.



DevSecOps integrates security into DevOps processes by:

- Automating security testing and validation.
- Implementing tools like security pipelines to scan code and infrastructure as code (IaC) for vulnerabilities.

At the core of a workload is the application code that implements business logic. The code and the process of developing code must be **free of security defects** to ensure confidentiality, integrity, and availability.

It's not enough to secure just the infrastructure plane by using controls on identity and networking and other measures. **Prevent bad implementation of code or a compromised code block** to strengthen your overall security posture. The usage plane, that is, the application code, must also be hardened. The process of integrating security into your development lifecycle is essentially a hardening process. Like resource hardening, tightening up code development is also context-agnostic. The focus is on enhancing security and not the functional requirements of the application. For information related to hardening, see Recommendations for hardening resources.

Definitions

Expand table

Term	Definition
Security Development Lifecycle (SDL)	A set of practices provided by Microsoft that supports security assurance and compliance requirements.
Software development lifecycle (SDLC)	A multistage, systematic process for developing software systems.

Key design strategies

Security measures should be integrated at multiple points into your existing Software Development Lifecycle (SDLC) to ensure:

- Design choices don't lead to security gaps.
- Application code and configuration don't create vulnerabilities because of exploitable implementation and improper coding practices.
- Software acquired via the supply chain doesn't introduce security threats.
- Application code, build, and deployment processes aren't tampered with.
- Vulnerabilities revealed through incidents are mitigated.
- Unused assets are properly decommissioned.
- Compliance requirements aren't compromised or reduced.
- Audit logging is implemented in developer environments.

The following sections provide security strategies for the commonly practiced phases of SDLC.

Collect and document the security requirements

The goal of the requirements phase is to **gather and analyze the functional and non-functional requirements** for an application or a new feature of an application. This phase is important because it facilitates the creation of guardrails that are tailored to the objectives of the application. Protecting the data and integrity of your application should be a core requirement throughout every phase of the development lifecycle.

For example, consider an application that needs to support critical user flows that enable the user to upload and manipulate data. The security design choices should cover assurances for the user's interaction with the application, like authenticating and authorizing the user identity, allowing only permitted actions on the data, and preventing SQL injection. Similarly, cover non-functional requirements like availability, scalability, and maintainability. Security choices should include segmentation boundaries, firewall ingress and egress, and other cross-cutting security concerns.

All these decisions should lead to a good definition of the security posture of the application. **Document the security requirements in an agreed-upon specification** and reflect it in the backlog. It should explicitly state the security investments and the tradeoffs and risks that the business is willing to take on if the investments aren't approved by business stakeholders. For example, you might document the need to use a web application firewall (WAF) in front of your application, like Azure Front Door or Azure Application Gateway. If business stakeholders aren't prepared to accept the additional cost of running a WAF, they need to accept the risk that application-layer attacks might be directed toward the application.

Security requirement gathering is a critical part of this phase. Without this effort, the design and implementation phases will be based on unstated choices, which can lead to security gaps. You might need to change the implementation later to accommodate security, which can be expensive.

Translate security requirements to technical requirements

During the design phase, **the security requirements are converted to technical requirements**. In your technical specification, document all design decisions to prevent ambiguity during implementation. Here are some typical tasks:

Define the security dimension of the system architecture

Overlay the architecture with security controls. For example, controls that are practical on the isolation boundaries per your segmentation strategy, the types of identities needed for the components of the application, and the type of encryption methods to

use. For some example architectures, see the illustrations in the Example sections of the Identity and access management and Networking articles.

Evaluate platform-provided affordances

It's important to understand the **division of responsibility between you and the cloud provider**. Avoid overlap with Azure native security controls, for example. You'll get better security coverage and be able to reallocate development resources to the needs of the application.

For example, if your design calls for a web application firewall on ingress, you can offload that responsibility to a load balancer like Application Gateway or Azure Front Door. Avoid replicating features as custom code in your application.

Choose only trusted frameworks, libraries, and supply chain software. Your design should also specify secure version control. Application dependencies should be sourced from trusted parties. **Third-party vendors should be able to meet your security requirements** and share their responsible disclosure plan. Any security incident should be promptly reported so that you can take necessary actions. Also, certain libraries might be prohibited by your organization. For example, software might be secure from vulnerabilities but still disallowed because of licensing restrictions.

To ensure that this guidance is followed by all contributors to the software, **maintain a list of approved and/or unapproved frameworks, libraries, and vendors**. When possible, place guardrails in the development pipelines to support the list. As much as possible, **automate the use of tools to scan dependencies** for vulnerabilities.

Determine the security design patterns that the application code should implement.

Patterns can support security concerns like segmentation and isolation, strong authorization, uniform application security, and modern protocols. Some operational patterns, such as the Quarantine pattern, can help verify and block the use of software that could potentially introduce security vulnerabilities.

For more information, see Cloud design patterns that support security.

Store application secrets securely

Securely implement the use of application secrets and pre-shared keys that your application uses. **Credentials and application secrets should never be stored in the source code tree.** Use external resources like Azure Key Vault to ensure that, if your

source code becomes available to a potential attacker, no further access can be obtained. In general, find ways to avoid secrets. Using managed identities, when possible, is one way to achieve that goal. For more information, see Recommendations for managing application secrets.

Define test plans

Define clear test cases for security requirements. Evaluate whether you can **automate those tests in your pipelines**. If your team has processes for manual testing, include security requirements for those tests.

> **Note**
>
> Perform threat modeling during this phase. Threat modeling can confirm that design choices are aligned with security requirements and expose gaps that you should mitigate. If your workload handles highly sensitive data, invest in security experts who can help you conduct threat modelling.

The initial threat modeling exercise should occur during the design phase when the software's architecture and high-level design are being defined. Doing it during that phase helps you to identify potential security issues before they're incorporated into the system's structure. However, this exercise isn't a one-time activity. It's a continuous process that should continue throughout the software's evolution.

For more information, see Recommendations for threat analysis.

Secure development and testing practices

During the development and testing phase, the goal is to **prevent security defects** and tampering in code, build, and deployment pipelines.

Be well-trained in secure code practices

The development team should **have formal and specialized training in secure coding practices**. For example, web and API developers might need specific training to protect against cross-site scripting attacks, and back-end developers can benefit from in-depth training to avoid database-level attacks like SQL injection attacks.

Developers should be required to complete this training before they can gain access to production source code.

You should also perform internal peer code reviews to promote continuous learning.

Use security test tools

Perform threat modeling to evaluate the security of the application's architecture.

Use **static application security testing (SAST)** to analyze code for vulnerabilities. Integrate this methodology into the developer environment to detect vulnerabilities in real time.

Use **dynamic application security testing (DAST)** during runtime. This tool chain can check for errors in security domains and simulate a set of attacks to test the application's security resilience. When possible, integrate this tool into your build pipelines.

Follow industry standards for secure coding practices. For more information, see the Community resources section of this article.

Use linters and code analyzers to prevent credentials from getting pushed to the source code repository. For example, .NET Compiler Platform (Roslyn) Analyzers inspect your application code.

During the build process, **use pipeline add-ons to catch credentials in the source code**. Scan all dependencies, like third-party libraries and framework components, as part of the continuous integration process. Investigate vulnerable components that are flagged by the tool. Combine this task with other code scanning tasks that inspect code churn, test results, and coverage.

Use a combination of tests. For information about security testing in general, see Recommendations for security testing.

Write just enough code

When you reduce your code footprint, you also reduce the chances of security defects. **Reuse code and libraries that are already in use and have been through security validations** instead of duplicating code.

Taking advantage of Azure features is another way to prevent unnecessary code. One way is to use managed services. For more information, see Use platform as a service (PaaS) options.

Write code with a deny-all approach by default. Create allowlists only for entities that need access. For example, if you have code that needs to determine whether a

privileged operation should be allowed, you should write it so that the *deny* outcome is the default case and the *allow* outcome occurs only when specifically permitted by code.

Protect developer environments

Developer workstations need to be protected with strong network and identity controls to prevent exposure. Make sure security updates are applied diligently.

Build agents are highly privileged and have access to the build server and the code. They must be protected with the same rigor as your workload components. This means that **access to build agents must be authenticated and authorized**, they should be network-segmented with firewall controls, they should be subject to vulnerability scanning, and so on. Microsoft-hosted build agents should be preferred over self-hosted build agents. Microsoft-hosted agents provide benefits like clean virtual machines for each run of a pipeline.

Custom build agents add management complexity and can become an attack vector. **Build machine credentials must be stored securely**, and you need to regularly remove any temporary build artifacts from the file system. You can achieve network isolation by only allowing outgoing traffic from the build agent, because it's using the pull model of communication with Azure DevOps.

The source code repository must be safeguarded as well. Grant access to code repositories on a need-to-know basis and reduce exposure of vulnerabilities as much as possible to avoid attacks. **Have a thorough process to review code** for security vulnerabilities. Use security groups for that purpose, and implement an approval process that's based on business justifications.

Protect code in deployment pipelines

It's not enough to just secure code. If it runs in exploitable pipelines, all security efforts are futile and incomplete. **Build and release environments must also be protected** because you want to prevent bad actors from running malicious code in your pipeline.

Maintain an up-to-date inventory of every component that's integrated into your application

Every new component that's integrated into an application increases the attack surface. To ensure proper accountability and alerting when new components are added or updated, you should have an inventory of these components. Store it outside of the build environment. **On a regular basis, check that your manifest matches what's in**

your build process. Doing so helps ensure that no new components that contain back doors or other malware are added unexpectedly.

Pipeline tasks

- **Pull tasks in your pipeline from trusted sources**, like Azure Marketplace. Run tasks that are written by your pipeline vendor. We recommend GitHub tasks or GitHub Actions. If you use GitHub workflows, prefer Microsoft-authored tasks. Also, validate tasks because they run in the security context of your pipeline.

- **Pipeline secrets.** Deployment assets that run inside a pipeline have access to all the secrets in that pipeline. **Have proper segmentation in place for different stages of the pipeline** to avoid unnecessary exposure. Use secret stores that are built into the pipeline. Remember that you can avoid using secrets in some situations. Explore the use of workload identities (for pipeline authentication) and managed identities (for service-to-service authentication).

Keep different environments separate

Data used in different environments must be kept separate. **Production data shouldn't be used in lower environments** because those environments might not have the strict security controls that production has. Avoid connecting from a non-production application to a production database, and avoid connecting non-production components to production networks.

Progressive exposure

Use progressive exposure to **release features to a subset of users** based on chosen criteria. If there are issues, the impact is minimized to those users. This approach is a common risk mitigation strategy because it reduces surface area. As the feature matures and you have more confidence in security assurances, you can gradually release it to a broader set of users.

Protect code in production

The production phase presents the **last responsible opportunity to fix security gaps**. Keep a record of the golden image that's released in production.

Keep versioned artifacts

Keep a catalog of all deployed assets and their versions. This information is useful during incident triage, when you're mitigating issues, and when you're getting the system back to working state. Versioned assets can also be compared against published Common Vulnerabilities and Exposures (CVE) notices. You should use automation to perform these comparisons.

Emergency fixes

Your automated pipeline design should have the flexibility to **support both regular and emergency deployments**. This flexibility is important to support rapid and responsible security fixes.

A release is typically associated with multiple approval gates. Consider creating an emergency process to accelerate security fixes. The process might involve communication among teams. The pipeline should allow for quick roll-forward and rollback deployments that address security fixes, critical bugs, and code updates that occur outside of the regular deployment lifecycle.

> ⓘ **Note**
>
> Always prioritize security fixes over convenience. A security fix shouldn't introduce a regression or bug. If you want to accelerate the fix through an emergency pipeline, carefully consider which automated tests can be bypassed. Evaluate the value of each test against the execution time. For example, unit tests usually complete quickly. Integration or end-to-end tests can run for a long time.

Maintain code security throughout its lifecycle

The goal of this phase is to **make sure security posture doesn't decay over time**. SDLC is an ongoing agile process. Concepts covered in the preceding phases apply to this phase because requirements change over time.

Patch management. Keep software, libraries, and infrastructure components up to date with security patches and updates.

Continuous improvement. Continuously assess and improve the security of the software development process by taking into account code reviews, feedback, lessons learned, and evolving threats.

Decommission legacy assets that are stale or no longer in use. Doing so reduces the surface area of the application.

Maintenance also includes incident fixes. If issues are found in production, they need to be promptly integrated back into the process so that they don't recur.

Continuously improve your secure coding practices to keep up with the threat landscape.

Azure facilitation

Microsoft Security Development Lifecycle (SDL) recommends secure practices that you can apply to your development lifecycle. For more information, see Microsoft Security Development Lifecycle.

Defender for DevOps and the SAST tools are included as part of GitHub Advanced Security or Azure DevOps. These tools can help you track a security score for your organization.

Follow the Azure security recommendations that are described in these resources:

- Design secure applications on Azure
- Develop secure applications on Azure
- Deploy secure applications on Azure
- Secure development best practices on Azure
- Training: Learn how Microsoft supports secure software development as part of a cybersecurity solution

Community links

To find credentials in source code, consider using tools like GitHub Advanced Security and OWASP source code analysis tools.

Validate the security of any open-source code in your application. These free tools and resources can help you with your assessment:

- Mend Bolt
- npm-audit
- OWASP Dependency-Check
- GitHub Dependabot
- Microsoft Security DevOps Azure DevOps extension
- OWASP Secure Coding Practices
- OWASP Top Ten

Related links

- Cloud design patterns that support security
- Design secure applications on Azure
- Deploy secure applications on Azure
- Develop secure applications on Azure
- Microsoft Security Development Lifecycle
- Recommendations for building a segmentation strategy
- Recommendations for hardening resources
- Recommendations for managing application secrets
- Recommendations for security testing
- Recommendations for threat analysis
- Secure development best practices on Azure
- Training: Learn how Microsoft supports secure software development as part of a cybersecurity solution
- Use platform as a service (PaaS) options

Security checklist

Refer to the complete set of recommendations.

Security checklist

Feedback

Was this page helpful? Yes No

Recommendations for threat analysis

Article • 06/18/2024

Applies to this Azure Well-Architected Framework Security checklist recommendation:

Expand table

SE:02	**Maintain a secure development lifecycle by using a hardened, mostly automated, and auditable software supply chain. Incorporate a secure design by using threat modeling to safeguard against security-defeating implementations.**

Related guide: Recommendations for securing a development lifecycle

A comprehensive analysis to identify threats, attacks, vulnerabilities, and counter measures is crucial during the design phase of a workload. *Threat modeling* is an engineering exercise that includes defining security requirements, identifying and mitigating threats, and validating those mitigations. You can use this technique at any stage of application development or production, but it's most effective during the design stages of new functionality.

This guide describes the recommendations for doing threat modeling so that you can identify security gaps quickly and design your security defenses.

Definitions

Expand table

Term	Definition
Software development lifecycle (SDLC)	A multistage, systematic process for developing software systems.
STRIDE	A Microsoft-defined taxonomy for categorizing types of threats.
Threat modeling	A process for identifying potential security vulnerabilities in the application and system, mitigating risks, and validating security controls.

Key design strategies

Threat modeling is a crucial process that an organization should integrate into its SDLC. Threat modeling is not solely a developer's task. It's a shared responsibility between:

- The workload team, which is responsible for the technical aspects of the system.

- Business stakeholders, who understand the business outcomes and have a vested interest in security.

There's often a disconnect between organizational leadership and technical teams regarding business requirements for critical workloads. This disconnect can lead to unwanted outcomes, particularly for security investments.

When the workload team is doing a threat modeling exercise, it should consider both business and technical requirements. The workload team and business stakeholders must agree on security-specific needs of the workload so that they can make adequate investments in the countermeasures.

The security requirements serve as guide for the entire process of threat modeling. To make it an effective exercise, the workload team should have a security mindset and be trained in threat modeling tools.

Understand the scope of the exercise

A clear understanding of the scope is crucial for effective threat modeling. It helps focus efforts and resources on the most critical areas. This strategy involves defining the boundaries of the system, taking inventory of the assets that need to be protected, and understanding the level of investment that's required in security controls.

Gather information about each component

A workload architecture diagram is a starting point for gathering information because it provides a visual representation of the system. The diagram highlights technical dimensions of the system. For example, it shows user flows, how data moves through the network, data sensitivity levels and information types, and identity access paths.

This detailed analysis can often provide insight into potential vulnerabilities in the design. It's important to understand the functionality of each component and its dependencies.

Evaluate the potential threats

Analyze each component from an outside-in perspective. For example, how easily can an attacker gain access to sensitive data? If attackers gain access to the environment, can they move laterally and potentially access or even manipulate other resources? These questions help you understand how an attacker might exploit workload assets.

Classify the threats by using an industry methodology

One methodology for classifying threats is STRIDE, which the Microsoft Security Development Lifecycle uses. Classifying threats helps you understand the nature of each threat and use appropriate security controls.

Mitigate the threats

Document all the identified threats. For each threat, define security controls and the response to an attack if those controls fail. Define a process and timeline that minimize exposure to any identified vulnerabilities in the workload, so that those vulnerabilities can't be left unaddressed.

Use the *assume breach* approach. It can help identify controls needed in the design to mitigate risk if a primary security control fails. Evaluate how likely it is for the primary control to fail. If it does fail, what is the extent of the potential organizational risk? Also, what is the effectiveness of the compensating control? Based on the evaluation, apply defense-in-depth measures to address potential failures of security controls.

Here's an example:

Expand table

Ask this question	To determine controls that...
Are connections authenticated through Microsoft Entra ID, Transport Layer Security (TLS) with mutual authentication, or another modern security protocol that the security team approved: - Between users and the application? - Between application components and services?	Prevent unauthorized access to the application components and data.
Are you limiting access to only accounts that need to write or modify data in the application?	Prevent unauthorized data tampering or alteration.
Is the application activity logged and fed into a security information and event management (SIEM) system through Azure Monitor or a similar solution?	Detect and investigate attacks quickly.
Is critical data protected with encryption that the security team approved?	Prevent unauthorized copying of data at rest.
Are inbound and outbound network traffic encrypted through TLS?	Prevent unauthorized copying of data in transit.

Ask this question	To determine controls that...
Is the application protected against distributed denial of service (DDoS) attacks through services such as Azure DDoS Protection?	Detect attacks designed to overload the application so it can't be used.
Does the application store sign-in credentials or keys to access other applications, databases, or services?	Identify whether an attack can use your application to attack other systems.
Do the application controls allow you to fulfill regulatory requirements?	Protect users' private data and avoid compliance fines.

Track threat modeling results

We highly recommend that you use a *threat modeling tool*. Tools can automate the process of identifying threats and produce a comprehensive report of all identified threats. Be sure to communicate the results to all interested teams.

Track the results as part of the workload team's backlog to allow for accountability in a timely way. Assign tasks to individuals who are responsible for mitigating a particular risk that threat modeling identified.

As you add new features to the solution, update the threat model and integrate it into the code management process. If you find a security problem, make sure there's a process to triage the problem based on severity. The process should help you determine when and how to remediate the problem (for example, in the next release cycle or in a faster release).

Regularly review business-critical workload requirements

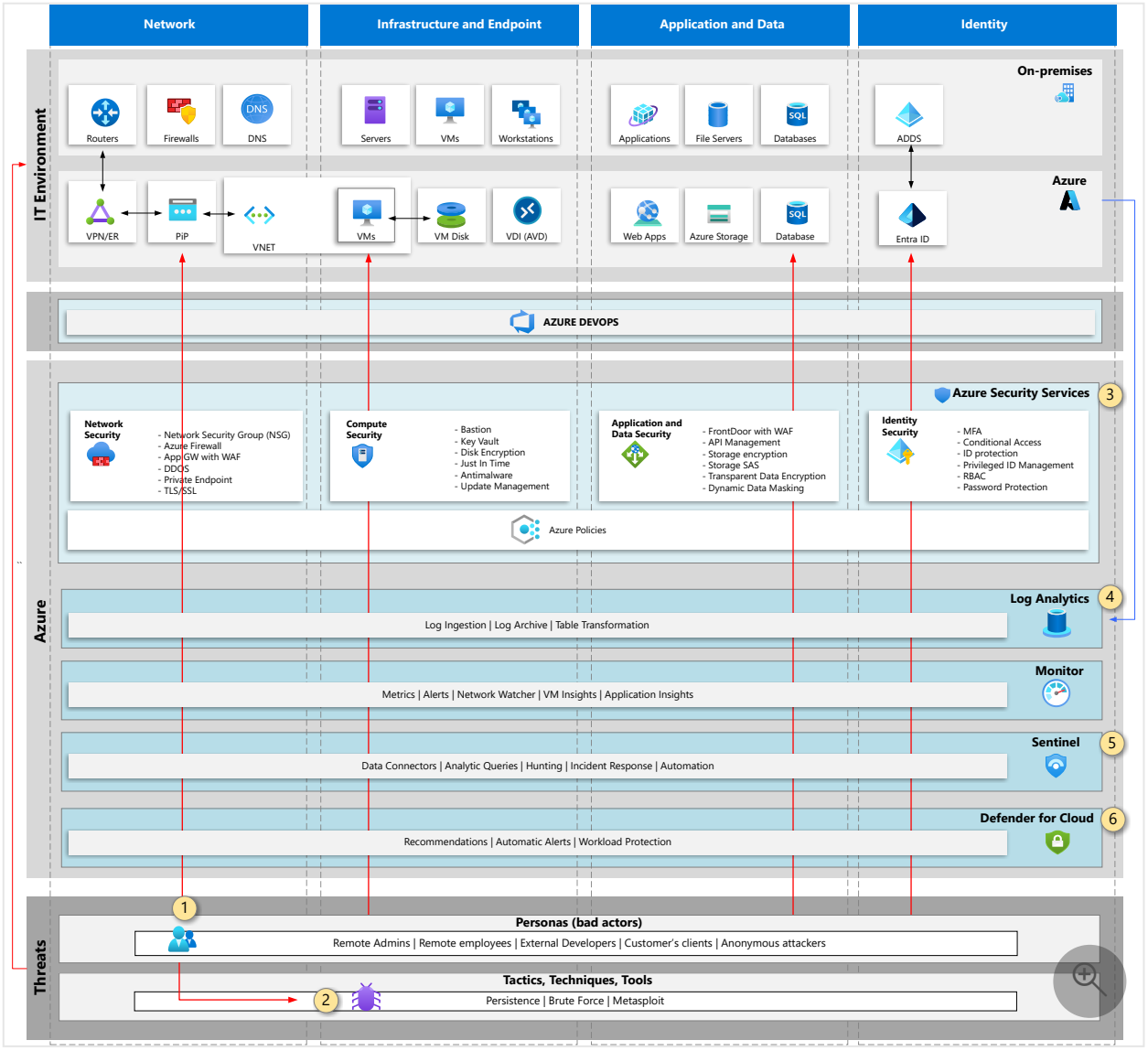
Meet regularly with executive sponsors to define requirements. These reviews provide an opportunity to align expectations and ensure operational resource allocation to the initiative.

Azure facilitation

The Microsoft Security Development Lifecycle provides a threat modeling tool to assist with the threat modeling process. This tool is available at no additional cost. For more information, see the Threat Modeling page.

Example

This example builds on the Information Technology (IT) environment established in the security baseline (SE:01). This approach provides a broad understanding of the threat landscape across different IT scenarios.



1. **Development Lifecycle personas**. There are many personas involved in a development life cycle, including developers, testers, final users, and administrators. All of them may be compromised and put your environment at risk through vulnerabilities or threats created intentionally.

2. **Potential attackers**. Attackers consider a wide range of tools available easily to be used at any time to explore your vulnerabilities and start an attack.

3. **Security controls**. As part of threat analysis, identify Azure security services to be used to protect your solution and how effective those solutions are.

4. **Log collection**. Logs from Azure resources and some on-premises components may be sent to Azure Log Analytics so you may understand the behavior of your solution developed and try to capture initial vulnerabilities.

5. **Security information event management (SIEM) solution**. Microsoft Sentinel may be added even in an early stage of the solution so you can build some analytics queries to mitigate threats and vulnerabilities, anticipating your security environment when you are in production.

6. **Microsoft Defender for Cloud** might make some security recommendations to improve the security posture.

Related links

- STRIDE
- Threat Modeling

Community links

Open Web Application Security Project (OWASP) has documented a threat modeling approach for applications.

Security checklist

Refer to the complete set of recommendations.

Security checklist

Feedback

Was this page helpful? Yes No

Recommendations for data classification

Article • 11/14/2023

Applies to Azure Well-Architected Framework Security checklist recommendation:

SE:03	**Classify and consistently apply sensitivity labels on all workload data and systems involved in data processing. Use classification to influence workload design, implementation, and security prioritization.**

This guide describes the recommendations for data classification. Most workloads store various types of data. Not all data is equally sensitive. Data classification helps you categorize data based on its sensitivity level, information type, and scope of compliance so you can apply the correct level of protection. Protection includes access controls, retention policies for different information types, and so on. While the actual security controls based on data classification are out of scope for this article, it provides recommendations for categorizing data based on the preceding criteria set by your organization.

Definitions

Term	Definition
Classification	A process to categorize workload assets by sensitivity levels, information type, compliance requirements, and other criteria provided by the organization.
Metadata	An implementation for applying taxonomy to assets.
Taxonomy	A system to organize classified data by using an agreed upon structure. Typically, a hierarchical depiction of data classification. It has named entities that indicate categorization criteria.

Key design strategies

Data classification is a crucial exercise that often drives building a system of record and its function. Classification also helps you correctly size security assurances and helps the triage team expediate discovery during incident response. A prerequisite to the design process is to clearly understand whether data should be treated as confidential, restricted, public, or any other sensitivity classification. It's also essential to determine the locations where data is stored, because the data might be distributed across multiple environments.

Data discovery is necessary to locate the data. Without that knowledge, most designs adopt a middle-ground approach, which might or might not serve the security requirements. Data can be overprotected, resulting in cost and performance inefficiencies. Or it might not be protected enough, which adds to the attack surface.

Data classification is often a cumbersome exercise. There are tools available that can discover data assets and suggest classifications. But don't just rely on tooling. Have a process in place where team members diligently do the exercises. Then use tooling to automate when that's practical.

Along with these best practices, see Create a well-designed data classification framework.

Understand organization-defined taxonomy

Taxonomy is a hierarchical depiction of data classification. It has named entities that indicate the categorization criteria.

In general, there isn't a universal standard for classification or for defining taxonomy. It's driven by an organization's motivation for protecting data. Taxonomy might capture compliance requirements, promised features for the workload users, or other criteria driven by business needs.

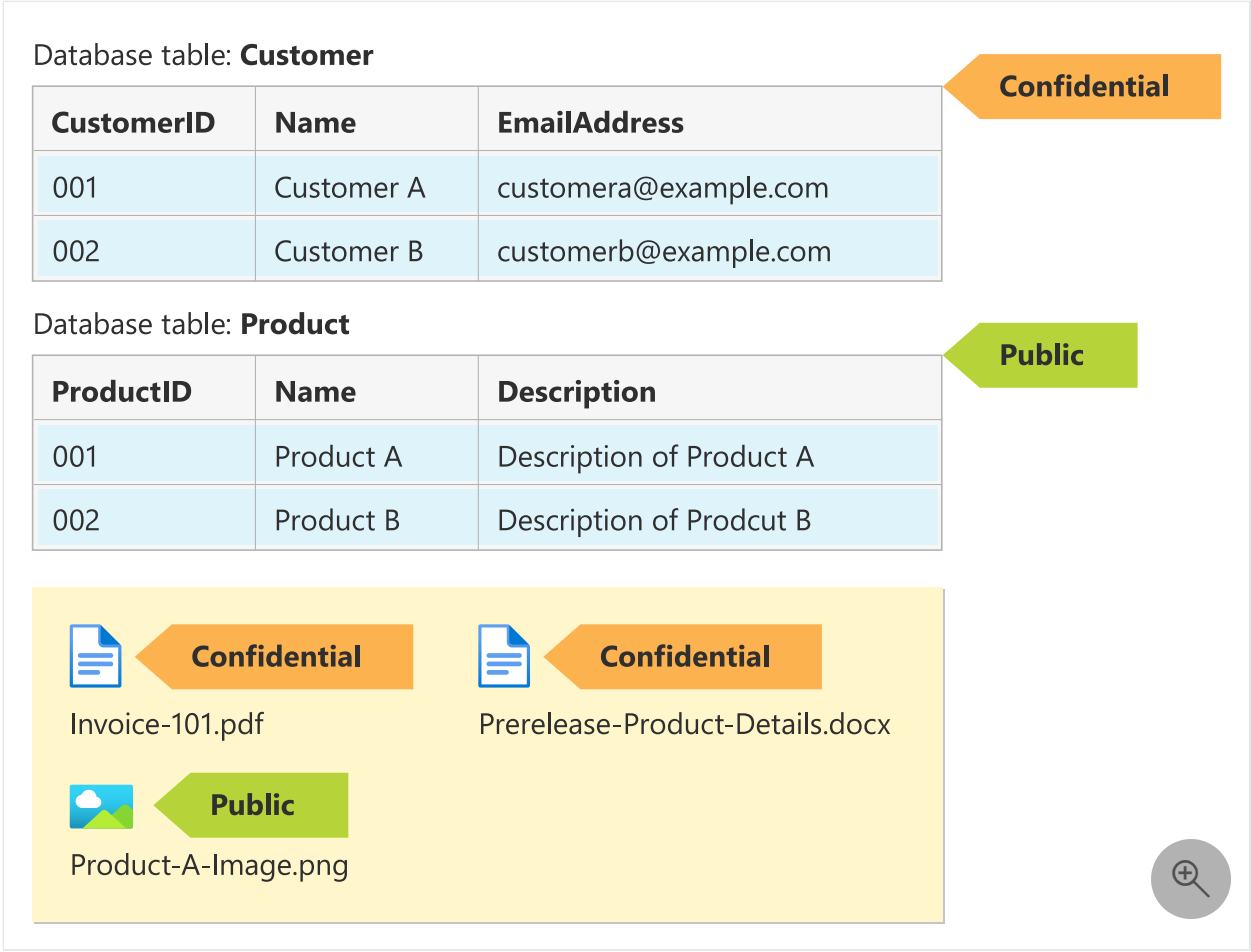
Here are some example classification labels for sensitivity levels, information type, and scope of compliance.

Sensitivity	Information type	Scope of compliance
Public, General, Confidential, Highly Confidential, Secret, Top Secret, Sensitive	Financial, Credit Card, Name, Contact Info, Credentials, Banking, Networking, SSN, Health fields, Date of Birth, Intellectual Property, personal data	HIPAA, PCI, CCPA, SOX, RTB

As a workload owner, rely on your organization to provide you with a well-defined taxonomy. All workload roles must have a shared understanding of the structure, nomenclature, and definition of the sensitivity levels. Don't define your own classification system.

Define the classification scope

Most organizations have a diverse set of labels.



Clearly identify which data assets and components are in-scope and out-of-scope for each sensitivity level. You should have a clear objective on the outcome. The objective could be quicker triage, accelerated disaster recovery, or regulatory audits. When you clearly understand the objectives, it ensures you correctly size your classification efforts.

Start with these simple questions and expand as necessary based on your system complexity:

- What's the origin of data and information type?
- What's the expected restriction based on access? For example, is it public information data, regulatory, or other expected use cases?
- What's the data footprint? Where is data stored? How long should the data be retained?
- Which components of the architecture interact with the data?
- How does the data move through the system?
- What information is expected in the audit reports?
- Do you need to classify preproduction data?

Take inventory of your data stores

If you have an existing system, take inventory of all data stores and components that are in scope. On the other hand, if you're designing a new system, create a data flow

dimension of the architecture and have an initial categorization per taxonomy definitions. Classification applies to the system as a whole. It's distinctly different from classifying configuration secrets and nonsecrets.

Define your scope

Be granular and explicit when defining the scope. Suppose your data store is a tabular system. You want to classify sensitivity at the table level or even the columns within the table. Also, be sure to extend classification to nondata store components that might be related or have a part in processing the data. For example, have you classified the backup of your highly sensitive data store? If you're caching user-sensitive data, is the caching data store in scope? If you use analytical data stores, how is the aggregated data classified?

Design according to classification labels

Classification should influence your architectural decisions. The most obvious area is your segmentation strategy, which should consider the varied classification labels.

For example, the labels influence the traffic isolation boundaries. There might be critical flows where end-to-end transport layer security (TLS) is required, while other packets can be sent over HTTP. If there are messages transmitted over a message broker, certain messages might need to be signed.

For data at rest, the levels will affect the encryption choices. You might choose to protect highly sensitive data through double encryption. Different application secrets might even require control with varied levels of protection. You might be able to justify storing secrets in a hardware security module (HSM) store, which offers higher restrictions. Compliance labels also dictate decisions about the right protection standards. For example, The PCI-DSS standard mandates the use of FIPS 140-2 Level 3 protection, which is available only with HSMs. In other cases, it might be acceptable for other secrets to be stored in a regular secret management store.

If you need to protect data in use, you might want to incorporate confidential compute in the architecture.

Classification information should move with the data as it transitions through the system and across components of the workload. Data labeled as confidential should be treated as confidential by all components that interact with it. For example, be sure to protect personal data by removing or obfuscating it from any kind of application logs.

Classification impacts the design of your report in the way data should be exposed. For example, based on your information type labels, do you need to apply a data masking algorithm for obfuscation as a result of the information type label? Which roles should have visibility into the raw data versus masked data? If there are any compliance requirements for reporting, how is data mapped to regulations and standards? When you have this understanding, it's easier to demonstrate compliance with specific requirements and generate reports for auditors.

It also impacts the data lifecycle management operations, such as data retention and decommissioning schedules.

Apply taxonomy for querying

There are many ways to apply taxonomy labels to the identified data. Using a classification schema with metadata is the most common way to indicate the labels. Standardization through schema makes sure that reporting is accurate, minimizes chances of variation, and avoids the creation of custom queries. Build automated checks to catch invalid entries.

You can apply labels manually, programmatically, or use a combination of both. The architecture design process should include design of the schema. Whether you have an existing system or are building a new one, when applying labels, maintain consistency in the key/value pairs.

Keep in mind that not all data can be clearly classified. Make an explicit decision about how the data that can't be classified should be represented in reporting.

The actual implementation depends on the type of resources. Certain Azure resources have built-in classification systems. For example, Azure SQL Server has a classification engine, supports dynamic masking, and can generate reports based on metadata. Azure Service Bus supports including a message schema that can have attached metadata. When you design your implementation, evaluate the features supported by the platform and take advantage of them. Make sure metadata used for classification is isolated and stored separately from the data stores.

There are also specialized classification tools that can detect and apply labels automatically. These tools are connected to your data sources. Microsoft Purview has autodiscover capabilities. There are also third-party tools that offer similar capabilities. The discovery process should be validated through manual verification.

Review data classification regularly. Classification maintenance should be built into operations, otherwise stale metadata can lead to erroneous results for the identified objectives and compliance issues.

> **Tradeoff**: Be mindful of the cost tradeoff on tooling. Classification tools require training and can be complex.

Ultimately, classification must roll up to the organization through central teams. Get input from them about the expected report structure. Also, take advantage of centralized tools and processes to have organizational alignment and also alleviate operational costs.

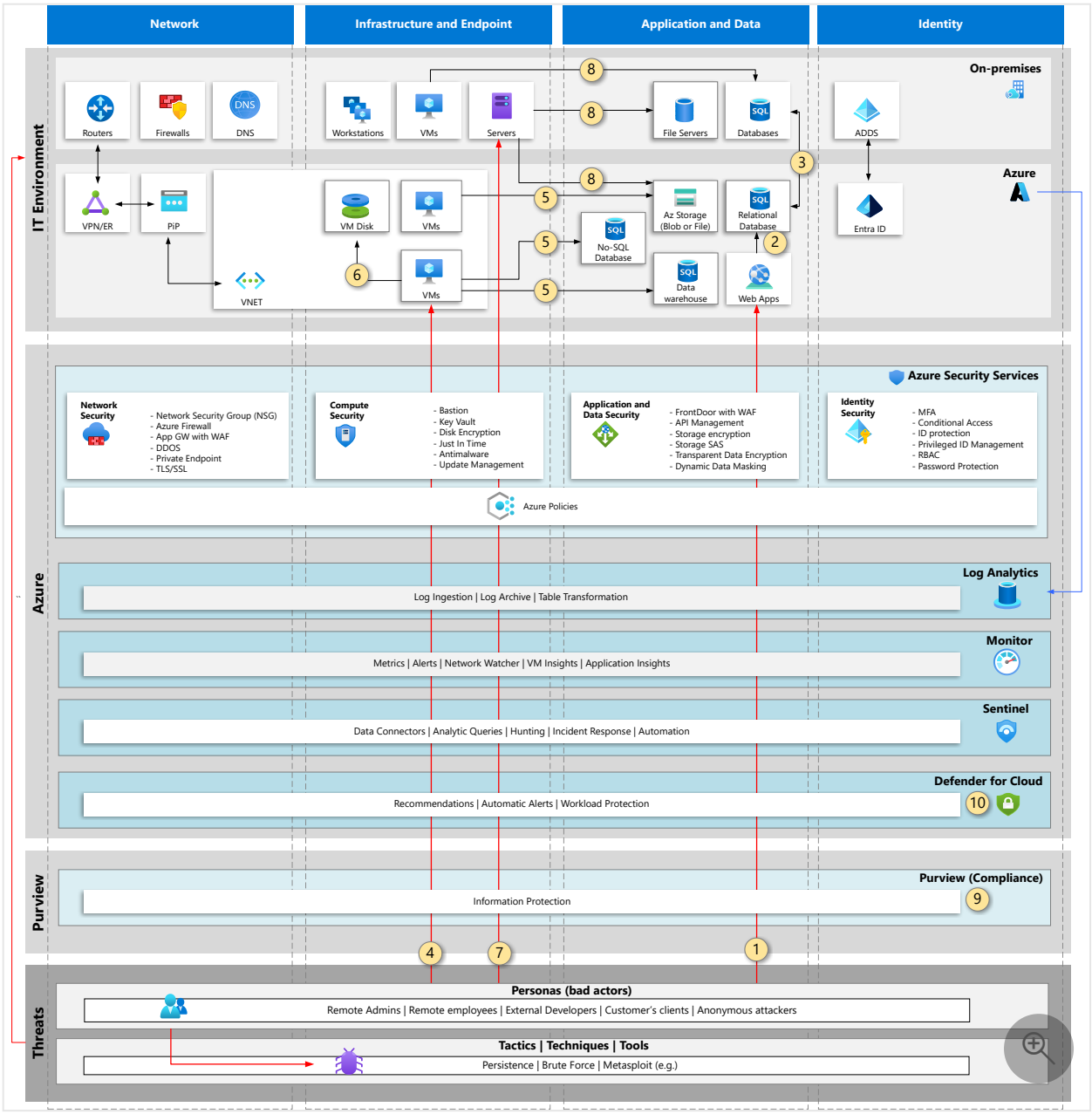
Azure facilitation

Microsoft Purview unifies Azure Purview and Microsoft Purview solutions to provide visibility into data assets throughout your organization. For more information, see What is Microsoft Purview?

Azure SQL Database, Azure SQL Managed Instance, and Azure Synapse Analytics offer built-in classification features. Use these tools to discover, classify, label, and report the sensitive data in your databases. For more information, see Data discovery and classification.

Example

This example builds on the Information Technology (IT) environment established in the security baseline (SE:01). The example diagram below shows data stores where data is classified.



1. Data stored on databases and disks should only be accessible to a few users, such as Administrators, Database administrators. Then, it's usual that common users or customers' final clients have access only to layers that are exposed to the internet, such as applications or jump boxes.

2. Applications communicate with the databases or data stored on disks, such as object storage or file servers.

3. In some cases, data might be stored in an on-premises environment and the public cloud. Both need to be classified consistently.

4. In an operator use case, remote administrators need access jump boxes on the cloud or a virtual machine running the workload. Access permissions should be given as per the data classification labels.

5. Data moves through the virtual machines to the backend databases and data should be treated with the same level of confidentiality throughout the traversal points.

6. Workloads store data directly in virtual machine disks. Those disks are in scope for classification.

7. In a hybrid environment, different personas may access workloads on-premises through different mechanisms to connect to different data storage technologies or databases. Access must be granted as per the classification labels.

8. The on-premises servers connect to important data that need to be classified and protected such as file servers, object storage, and different types of databases, such as relational, no-SQL, and data warehouse.

9. Microsoft Purview Compliance provides a solution to classify files and emails.

10. Microsoft Defender for Cloud provides a solution that helps your company to track compliance in your environment, including many of your services used to store data, mentioned in these se cases above.

Organizational alignment

Cloud Adoption Framework provides guidance for central teams about how to classify data so that workload teams can follow the organizational taxonomy.

For more information, see What is data classification? - Cloud Adoption Framework.

Related links

- Data classification and sensitivity label taxonomy - Microsoft Service Assurance
- Create a well-designed data classification framework - Microsoft Service Assurance

Next step

Refer to the complete set of recommendations.

Security checklist

Recommendations for building a segmentation strategy

Article • 11/14/2023

Applies to Well-Architected Framework Security checklist recommendation:

SE:04	**Create intentional segmentation and perimeters in your architecture design and the workload's footprint on the platform. The segmentation strategy must include networks, roles and responsibilities, workload identities, and resource organization.**

A segment is a logical section of your solution that needs to be secured as one unit. A segmentation strategy defines how one unit should be separated from other units with its own set of security requirements and measures.

This guide describes the recommendations for **building a unified segmentation strategy**. Using perimeters and isolation boundaries in workloads, you can design a security approach that works for you.

Definitions

Term	Definition
Containment	A technique to contain the blast radius if an attacker gains access to a segment.
Least-privilege access	A Zero Trust principle that aims at minimizing a set of permissions to complete a job function.
Perimeter	The trust boundary around a segment.
Resource organization	A strategy to group related resources by flows within a segment.
Role	A set of permissions needed to complete a job function.
Segment	A logical unit that's isolated from other entities and protected by a set of security measures.

Key design strategies

The concept of segmentation is commonly used for networks. However, the same underlying principle can be used throughout a solution, including segmenting resources for management purposes and access control.

Segmentation helps you **design a security approach that applies defense in depth** based on the principles of the Zero Trust model. Ensure that an attacker who breaches one network segment can't gain access to another by segmenting workloads with different identity controls. In a secure system, identity and network attributes block unauthorized access and hide the assets from being exposed. Here are some examples of segments:

- Subscriptions that isolate workloads of an organization
- Resource groups that isolate workload assets
- Deployment environments that isolate deployment by stages
- Teams and roles that isolate job functions related to workload development and management
- Application tiers that isolate by workload utility
- Microservices that isolate one service from another

Consider these key elements of segmentation to make sure you're building a comprehensive defense in depth strategy:

- The **boundary or perimeter** is the entry edge of a segment where you apply security controls. Perimeter controls should block access to the segment unless explicitly allowed. The goal is to prevent an attacker from breaking through the perimeter and gaining control of the system. For example, an application tier might accept an end user's access token when it processes a request. But the *data* tier might require a different access token that has a specific permission, which only the application tier can request.

- **Containment** is the exit edge of a segment that prevents lateral movement in the system. The goal of containment is to minimize the effect of a breach. For example, an Azure virtual network might be used to configure routing and network security groups to only allow traffic patterns that you expect, avoiding traffic to arbitrary network segments.

- **Isolation** is the practice of grouping entities with similar assurances together to protect them with a boundary. The goal is ease of management and the containment of an attack within an environment. For example, you might group the resources that relate to a specific workload into one Azure subscription, and then apply access control so that only specific workload teams can access the subscription.

It's important to note the distinction between perimeters and isolation. Perimeter refers to the points of location that should be checked. Isolation is about grouping. Actively contain an attack by using these concepts together.

Isolation doesn't mean creating silos in the organization. **A unified segmentation strategy provides alignment between the technical teams and sets clear lines of responsibility.** Clarity reduces the risk of human error and automation failures that can lead to security vulnerabilities, operational downtime, or both. Suppose a security breach is detected in a component of a complex enterprise system. It's important that everyone understands who's responsible for that resource so that the appropriate person is included in the triage team. The organization and stakeholders can quickly identify how to respond to different kinds of incidents by creating and documenting a good segmentation strategy.

>  **Tradeoff**: Segmentation introduces complexity because there's overhead in management. There's also a tradeoff in cost. For example, more resources are provisioned when deployment environments that run side by side are segmented.

>  **Risk**: Micro-segmentation beyond a reasonable limit loses the benefit of isolation. When you create too many segments, it becomes difficult to identify points of communication or to allow for valid communication paths within the segment.

Identity as the perimeter

Various identities such as people, software components, or devices access workload segments. Identity is a perimeter that should be the primary line of defense to **authenticate and authorize access across isolation boundaries**, regardless of where the access request originates. Use identity as a perimeter to:

- **Assign access by role.** Identities only need access to the segments required to do their job. Minimize anonymous access by understanding the roles and responsibilities of the requesting identity so that you know the entity that's requesting access to a segment and for what purpose.

  An identity might have different access scopes in different segments. Consider a typical environment setup, with separate segments for each stage. Identities associated with the developer role have read-write access to the development environment. As the deployment moves to staging, those permissions are curbed. By the time the workload is promoted to production, scope for developers is reduced to read-only access.

- **Consider application and management identities separately.** In most solutions, users have a different level of access than developers or operators. In some

applications, you might use different identity systems or directories for each type of identity. Consider using access scopes and creating separate roles for each identity.

- **Assign least-privilege access.** If the identity is allowed access, determine the level of access. Start with the least privilege for each segment and broaden that scope only when needed.

  By applying the least privilege, you limit the negative effects if the identity is ever compromised. If access is limited by time, the attack surface is reduced further. Time-limited access is especially applicable to critical accounts, such as administrators or software components that have a compromised identity.

>  **Tradeoff**: The performance of the workload can be affected by identity perimeters. Verifying each request explicitly requires extra compute cycles and extra network IO.
>
> Role-based access control (RBAC) also results in management overhead. Keeping track of identities and their access scopes can become complex in role assignments. The workaround is to assign roles to security groups instead of individual identities.

>  **Risk**: Identity settings can be complex. Misconfigurations can affect the reliability of the workload. For example, suppose there's a misconfigured role assignment that's denied access to a database. The requests start failing, eventually causing reliability issues that can't otherwise be detected until runtime.

For information about identity controls, see Identity and access management.

In contrast to network access controls, identity validates access control at access time. It's highly recommended to conduct regular access review and require an approval workflow to obtain privileges for critical impact accounts. For example, see Identity segmentation patterns.

Networking as a perimeter

Identity perimeters are network agnostic while network perimeters augment identity but never replace it. Network perimeters are established to control blast radius, block unexpected, prohibited, and unsafe access, and obfuscate workload resources.

While the primary focus of the identity perimeter is least privilege, you should assume there will be a breach when you're designing the network perimeter.

Create software-defined perimeters in your networking footprint using Azure services and features. When a workload (or parts of a given workload) is placed into separate segments, you **control traffic from or to those segments to secure communication paths**. If a segment is compromised, it's contained and prevented from laterally spreading through the rest of your network.

Think like an attacker to achieve a foothold within the workload and establish controls to minimize further expansion. The controls should detect, contain, and stop attackers from gaining access to the entire workload. Here are some examples of network controls as a perimeter:

- Define your edge perimeter between public networks and the network where your workload is placed. Restrict line of sight from public networks to your network as much as possible.
- Implement demilitarized zones (DMZs) in front of the application with proper controls via firewalls.
- Create micro-segmentation within your private network by grouping parts of the workload into separate segments. Establish secure communication paths between them.
- Create boundaries based on intent. For example, segment workload functional networks from operational networks.

For common patterns related to networking segmentation, see Networking segmentation patterns.

> **Tradeoff**: Network security controls are often expensive because they're included with the premium SKUs. Configuring rules on firewalls often results in overwhelming complexity requiring broad exceptions.
>
> Private connectivity changes architectural design, often adding more components such as jump boxes for private access to compute nodes.
>
> Because network perimeters are based on control points, or hops, on the network, each hop can be a potential point of failure. These points can have an effect on the reliability of the system.

> **Risk**: Network controls are rule-based and there's a significant chance of misconfiguration, which is a reliability concern.

For information about network controls, see Networking and connectivity.

Roles and responsibilities

Segmentation that prevents confusion and security risks is achieved by **clearly defining lines of responsibility** within a workload team.

Document and share roles and functions to create consistency and facilitate communication. Designate groups or individual roles that are responsible for key functions. Consider the built-in roles in Azure before creating custom roles for objects.

Consider consistency while accommodating several organizational models when assigning permissions for a segment. These models can range from a single centralized IT group to mostly independent IT and DevOps teams.

>  **Risk**: Membership of groups can change over time as employees join or leave teams or change roles. Management of roles across segments can result in management overhead.

Resource organization

Segmentation allows you to **isolate workload resources from other parts of the organization** or even within the team. Azure constructs, such as management groups, subscriptions, environments, and resource groups, are ways of organizing your resources that promote segmentation. Here are some examples of resource-level isolation:

- Polyglot persistence involves a combination of data storing technologies instead of a single database system to support segmentation. Use polyglot persistence for separation by various data models, separation of functionalities such as data storage and analytics, or to separate by access patterns.
- Allocate one service for each server when organizing your compute. This level of isolation minimizes complexity and can help contain an attack.
- Azure provides built-in isolation for some services, for example separation of compute from storage. For other examples, see Isolation in the Azure public cloud.

>  **Tradeoff**: Resource isolation might result in an increase in total cost of ownership (TCO). For data stores, there might be added complexity and coordination during disaster recovery.

Azure facilitation

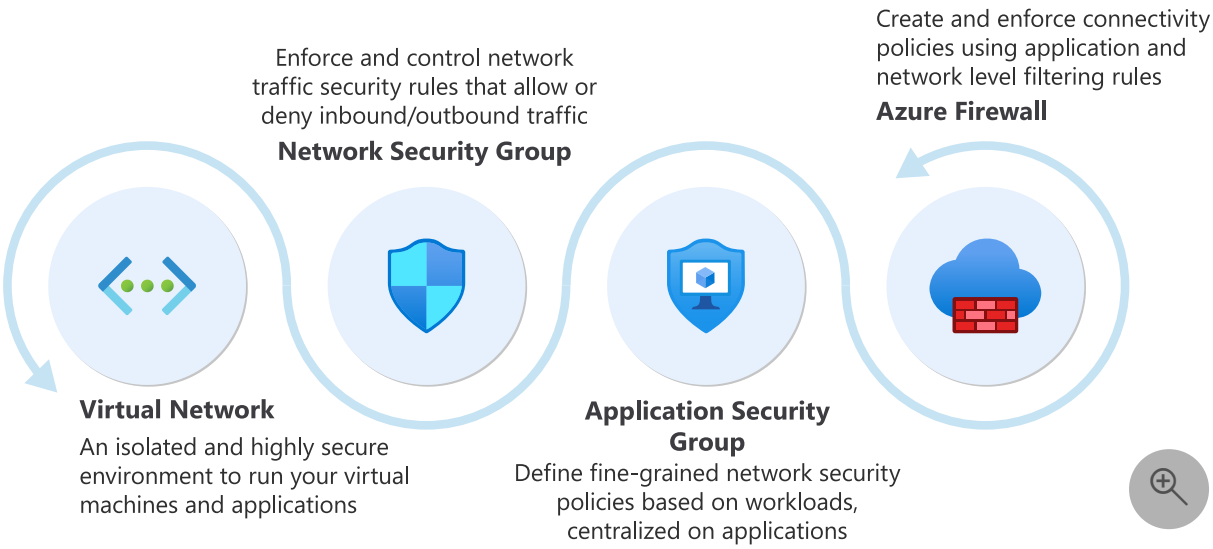
Certain Azure services are available for use in implementing a segmentation strategy, as outlined in the following sections.

Identity

Azure RBAC supports segmentation by isolating access by job function. Only certain actions are allowed for certain roles and scopes. For example, job functions that only need to observe the system can be assigned reader permissions versus contributor permissions that allow the identity to manage resources.

For more information, see Best practices for RBAC.

Networking



Virtual networks: Virtual networks provide network-level containment of resources without adding traffic between two virtual networks. Virtual networks are created in private address spaces within a subscription

Network security groups (NSG): An access control mechanism for controlling traffic between resources in virtual networks and external networks, such as the internet. Implement user-defined routes (UDR) to control the next hop for traffic. NSGs can take your segmentation strategy to a granular level by creating perimeters for a subnet, a virtual machine (VM), or a group of VMs. For information about possible operations with subnets in Azure, see Subnets.

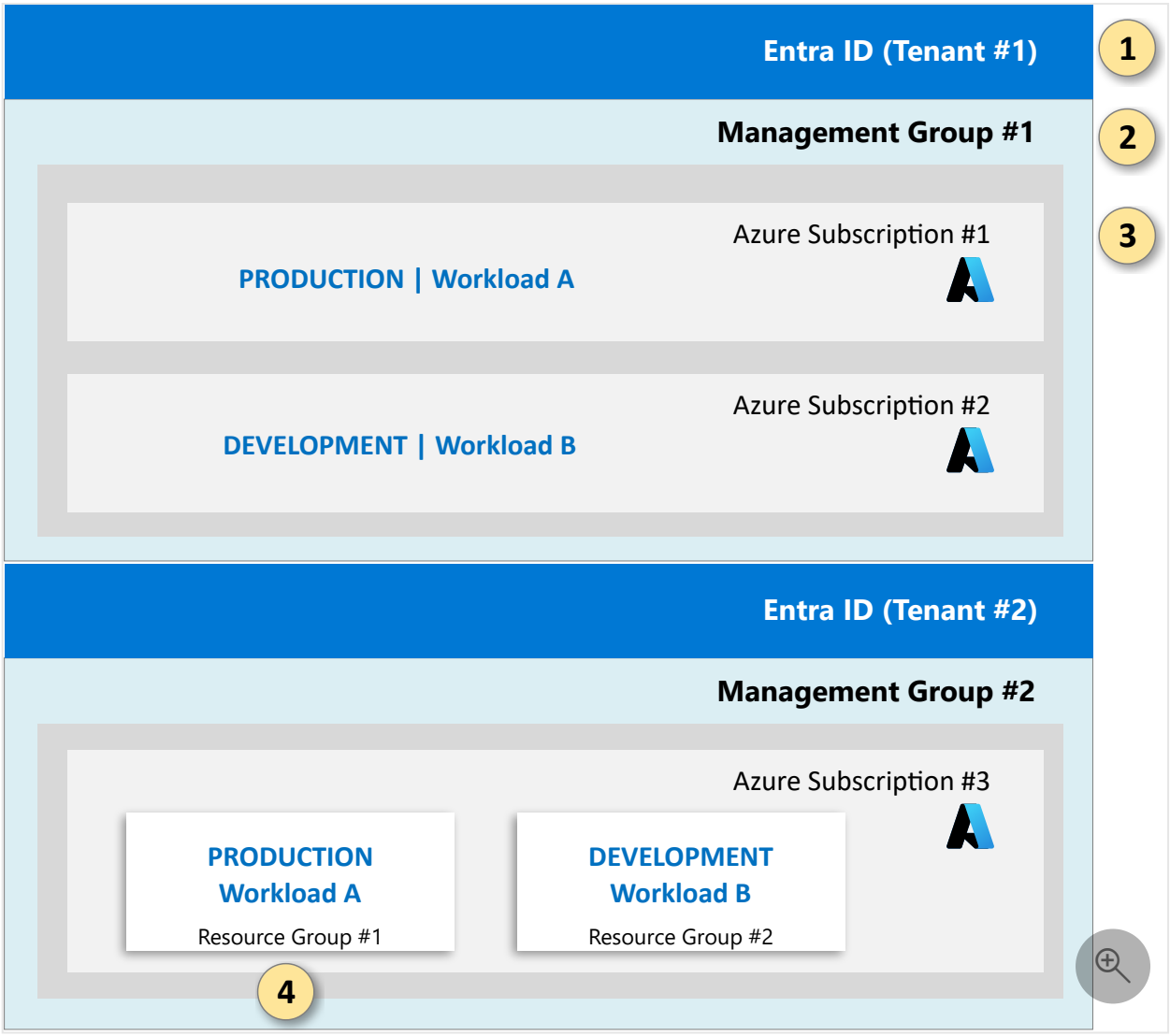
Application security groups (ASGs): ASGs allow you to group a set of VMs under an application tag and define traffic rules that are then applied to each of the underlying VMs.

Azure Firewall: A cloud-native service, which can be deployed in your virtual network or in Azure Virtual WAN hub deployments. Use Azure Firewall to filter traffic flowing between cloud resources, the internet, and on-premises resources. Use Azure Firewall or Azure Firewall Manager to create rules or policies that allow or deny traffic using layer 3 to layer 7 controls. Filter internet traffic using Azure Firewall and third parties by directing traffic through third-party security providers for advanced filtering and user protection. Azure supports network virtual appliance deployment, which helps segmentation from third-party firewalls.

Example

Here are some common patterns for segmenting a workload in Azure. Choose a pattern based on your needs.

This example builds on the Information Technology (IT) environment established in the security baseline (SE:01). The diagram below shows segmentation at the management group level done by an organization.



Identity segmentation patterns

Pattern 1: Job title-based grouping

One way to organize security groups is by job title like software engineer, database administrator, site reliability engineer, quality assurance engineer, or security analyst. This approach involves **creating security groups for your workload team** based on their roles, without considering the work that needs to be accomplished. Grant security groups RBAC permissions, standing or just in time (JIT), according to their responsibilities in the workload. Assign human and service principles to security groups based on their as-needed access.

Membership is highly visible at the role assignment level, making it easy to see what a *role* has access to. Each person is usually a member of only one security group, which makes onboarding and offboarding easy. However, unless job titles overlap perfectly with responsibilities, title-based grouping isn't ideal for least-privilege implementation. You might end up combining implementation with function-based grouping.

Pattern 2: Function-based grouping

Function-based grouping is a security group organization method that reflects discrete work that needs to be accomplished, not taking into account your team structure. With this pattern, you **grant security groups RBAC permissions, standing or JIT as needed**, according to their required function in the workload.

Assign human and service principles to security groups based on their as-needed access. Where possible, use existing homogeneous groups as members of the function-based groups, such as those groups from pattern 1. Examples of function-based groups include:

- Production database operators
- Preproduction database operators
- Production certificate rotation operators
- Preproduction certificate rotation operators
- Production live-site/triage
- Preproduction all access

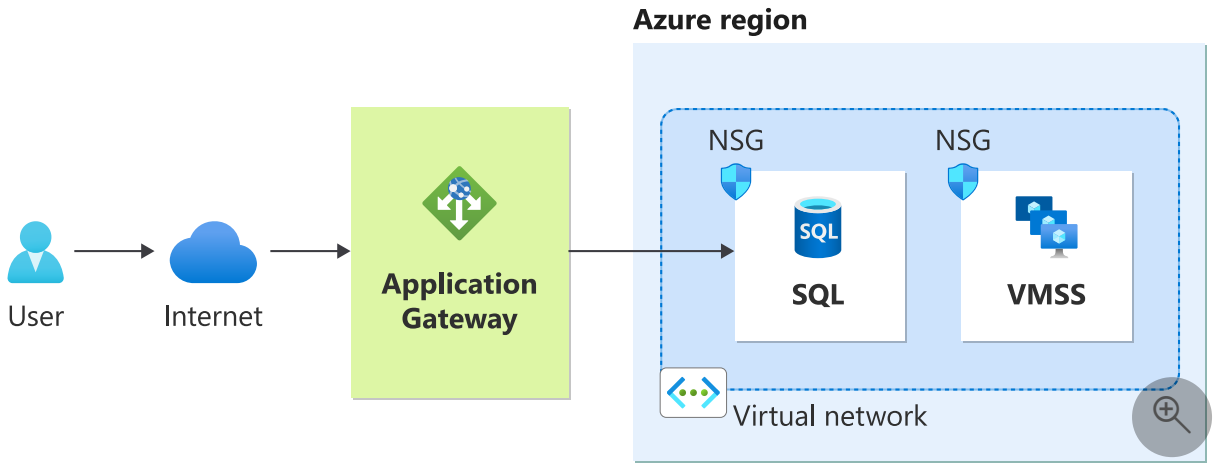
This approach maintains the strictest least-privilege access and provides security groups where scope is evident, which makes it easy to audit memberships relative to job duties performed. Often a built-in Azure role exists to match this job function.

However, membership is abstracted at least one layer, forcing you to go to the identity provider to understand who's in the group when looking from the resource perspective. Additionally, one person needs to have multiple memberships maintained for complete coverage. The matrix of overlapping security groups can be complex.

Pattern 2 is recommended to make the access patterns the focus, not the organization chart. Organization charts and member roles sometimes change. Capturing your workload's identity and access management from a functional perspective allows you to abstract your team organization from the secure management of the workload.

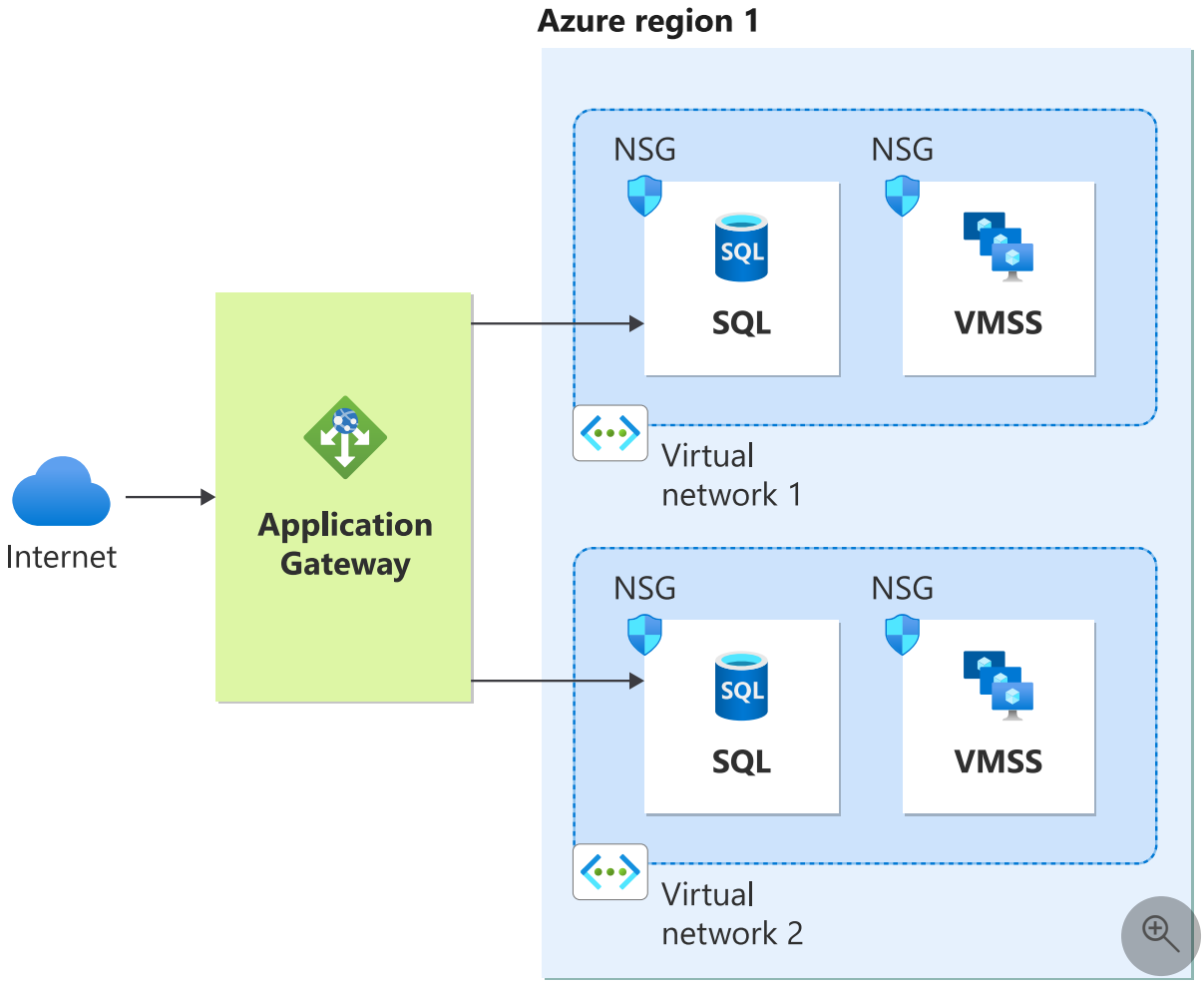
Networking segmentation patterns

Pattern 1: Segmentation within a workload (soft boundaries)



In this pattern, the workload is placed in a single virtual network using subnets to mark boundaries. **Segmentation is achieved using two subnets**, one for database and one for web workloads. You must configure NSGs that allow Subnet 1 to only communicate with Subnet 2 and Subnet 2 to only communicate with the internet. This pattern provides layer 3 level control.

Pattern 2: Segmentation within a workload



This pattern is an example of platform-level segmentation. Workload c**omponents are spread across multiple networks without peering between them**. All communication is routed through an intermediary that serves as a public access point. The workload team owns all networks.

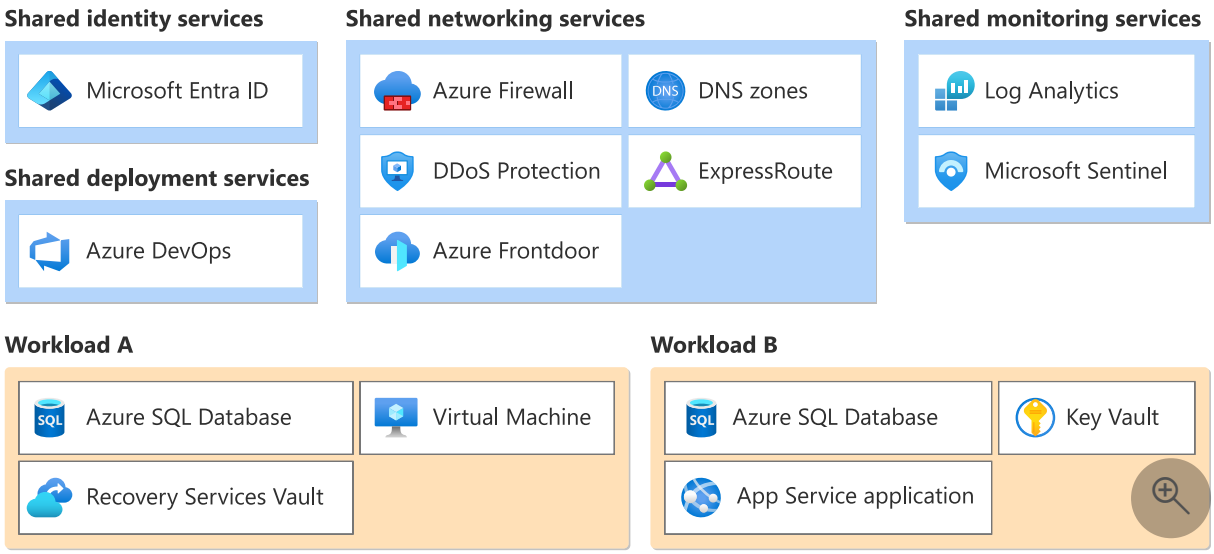
Pattern 2 provides containment but has the added complexity of virtual network management and sizing. Communication between the two networks takes place over the public internet, which can be a risk. There's also latency with public connections. However, the two networks can be peered, breaking segmentation by connecting them to create a larger segment. Peering should be done when no other public endpoints are needed.

Considerations	Pattern 1	Pattern 2
Connectivity and routing: How each segment communicates	System routing provides default connectivity to workload components. No external component can communicate with the workload.	Within the virtual network, same as pattern 1. Between networks, the traffic goes over the public internet. There's no direct connectivity between the networks.

Considerations	Pattern 1	Pattern 2
Network-level traffic filtering	Traffic between the segments is allowed by default. Use NSGs or ASGs to filter traffic.	Within the virtual network, same as pattern 1. Between the networks, you can filter both ingress and egress traffic through a firewall.
Unintended open public endpoints	Network interface cards (NICs) don't get public IPs. Virtual networks aren't exposed to internet API management.	Same as pattern 1. Intended open public endpoint on one virtual network, which can be misconfigured to accept more traffic.

Resource organization

Organize Azure resources based on ownership responsibility



Consider an Azure estate that contains multiple workloads and shared service components like hub virtual networks, firewalls, identity services, and security services like Microsoft Sentinel. Components throughout the estate should be grouped based on their functional areas, workloads, and ownership. For example, shared networking resources should be grouped together into a single subscription and managed by a networking team. Components that are dedicated to individual workloads should be in their own segment and might be further divided based on application tiers or other organizational principles.

Grant access to manage resources within individual segments by creating RBAC role assignments. For example, the cloud networking team might be granted administrative

access to the subscription that contains their resources, but not to individual workload subscriptions.

A good segmentation strategy makes it possible to easily identify the owners of each segment. Consider using Azure resource tags to annotate resource groups or subscriptions with the owner team.

Configure and review access control

Grant appropriate access based on need by clearly defining segments for your resources.

Consider the principle of least privilege when you define access control policies. It's important to distinguish between *control plane operations* (management of the resource itself) and *data plane operations* (access to the data stored by the resource). For example, suppose you have a workload that contains a database with sensitive information about employees. You might grant management access to some users that need to configure settings like database backups or users that monitor the performance of the database server. However, these users shouldn't be able to query the sensitive data stored in the database. Select permissions that grant the minimum scope needed for users to perform their duties. Regularly review role assignments for each segment and remove access that's no longer required.

> **Note**
>
> Some highly privileged roles, like the owner role in RBAC, give users the ability to grant other users access to a resource. Limit how many users or groups are assigned the owner role, and regularly review audit logs to ensure they only perform valid operations.

Related links

- Isolation in the Azure public cloud
- Recommendations for RBAC
- Virtual networks overview
- ASGs
- Azure Firewall
- Firewall Manager overview

Security checklist

Refer to the complete set of recommendations.

Security checklist

Recommendations for identity and access management

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Security checklist recommendation:

> **SE:05** **Implement strict, conditional, and auditable identity and access management (IAM) across all workload users, team members, and system components. Limit access exclusively to *as necessary*. Use modern industry standards for all authentication and authorization implementations. Restrict and rigorously audit access that's not based on identity.**

This guide describes the recommendations for authenticating and authorizing identities that are attempting to access your workload resources.

From a technical control perspective, **identity is always the primary perimeter**. This scope doesn't just include the edges of your workload. It also includes individual components that are inside your workload. Typical identities include:

- **Humans**. Application users, admins, operators, auditors, and bad actors.
- **Systems**. Workload identities, managed identities, API keys, service principals, and Azure resources.
- **Anonymous**. Entities who haven't provided any evidence about who they are.

Definitions

Terms	Definition
Authentication (AuthN)	A process that verifies that an identity is who or what it says it is.
Authorization (AuthZ)	A process that verifies whether an identity has permission to perform a requested action.
Conditional access	A set of rules that allows actions based on specified criteria.
IdP	An identity provider, like Microsoft Entra ID.
Persona	A job function or a title that has a set of responsibilities and actions.
Preshared keys	A type of secret that's shared between a provider and consumer and used through a secure and agreed upon mechanism.

Terms	Definition
Resource identity	An identity defined for cloud resources that's managed by the platform.
Role	A set of permissions that define what a user or group can do.
Scope	Different levels of organizational hierarchy where a role is permitted to operate. Also a group of features in a system.
Security principal	An identity that provides permissions. It can be a user, a group, or a service principal. Any group members get the same level of access.
User identity	An identity for a person, like an employee or an external user.
Workload identity	A system identity for an application, service, script, container, or other component of a workload that's used to authenticate itself to other services and resources.

> **Note**
>
> An identity can be grouped with other, similar identities under a parent called a *security principal*. A security group is an example of a security principal. This hierarchical relationship simplifies maintenance and improves consistency. Because identity attributes aren't handled at the individual level, chances of errors are also reduced. In this article, the term *identity* is inclusive of security principals.

The role of an identity provider

An identity provider (IdP) is a cloud-hosted service that stores and manages users as digital identities.

Take advantage of the capabilities provided by a trusted IdP for your identity and access management. Don't implement custom systems to replace an IdP. IdP systems are improved frequently based on the latest attack vectors by capturing billions of signals across multiple tenants each day. Microsoft Entra ID is the IdP for Azure cloud platform.

Authentication

Authentication is a process that verifies identities. The requesting identity is required to provide some form of verifiable identification. For example:

- A user name and password.

- A preshared secret, like an API key that grants access.
- A shared access signature (SAS) token.
- A certificate that's used in TLS mutual authentication.

As much as possible, the verification process should be handled by your IdP.

Authorization

Authorization is a process that allows or denies actions that are requested by the verified identity. The action might be operational or related to resource management.

Authorization requires that you assign permissions to the identities, which you need to do by using the functionality provided by your IdP.

Key design strategies

To get a holistic view of the identity needs for a workload, you need to catalog the flows, workload assets, and personas, and the actions the assets and personas will perform. Your strategy must cover all use cases that handle **the flows that reach the workload or its components (outside-in access) and flows that reach out from the workload to other sources (inside-out access)**.

Each use case will probably have its own set of controls that you need to design with an assume-breach mindset. Based on the identity requirements of the use case or the personas, identify the conditional choices. Avoid using one solution for all use cases. Conversely, the controls shouldn't be so granular that you introduce unnecessary management overhead.

You need to log the identity access trail. Doing so helps validate the controls, and you can use the logs for compliance audits.

Determine all identities for authentication

- **Outside-in access**. Your identity design must authenticate all users that access the workload for various purposes. For example, an end user who accesses the application by calling APIs.

  At a granular level, components of the workload might also need access from outside. For example, an operator who needs access through the portal or access to the compute to run commands.

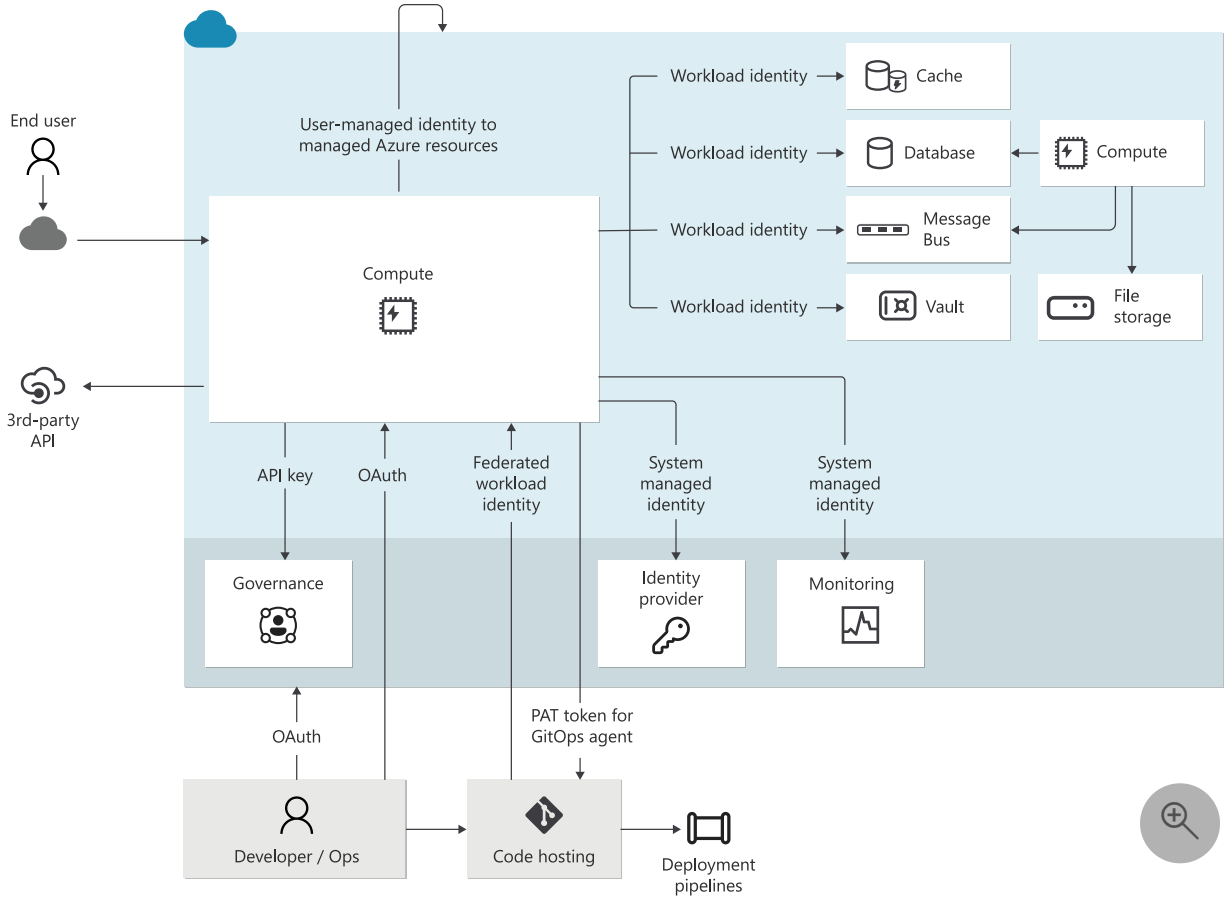
Both are examples of **user identities** that have different personas.

- **Inside-out access**. Your application will need to access other resources. For example, reading from or writing to the data platform, retrieving secrets from the secret store, and logging telemetry to monitoring services. It might even need to access third-party services. These access needs require **workload identity**, which enables the application to authenticate itself against the other resources.

  The concept applies at the component level. In the following example, the container might need access to deployment pipelines to get its configuration. These access needs require **resource identity**.

All these identities should be authenticated by your IdP.

Here's an example of how identity can be implemented in an architecture:



Determine actions for authorization

Next, you need to know what each authenticated identity is trying to do so that those actions can be authorized. The actions can be divided by the type of access that they require:

- **Data plane access**. Actions that take place in the data plane cause data transfer for inside-out or outside-in access. For example, an application reading data from a

database and writing data to a database, fetching secrets, or writing logs to a monitoring sink. At the component level, compute that's pulling or pushing images to or from a registry are considered data plane operations.

- **Control plane access**. Actions that take place in the control plane cause an Azure resource to be created, modified, or deleted. For example, changes to resource properties.

Applications typically target data plane operations, while operations often access both control and data planes. To identify authorization needs, note the operational actions that can be performed on the resource. For information about the permitted actions for each resource, see Azure resource provider operations.

Provide role-based authorization

Based on the responsibility of each identity, authorize actions that should be permitted. **An identity must not be allowed to do more than it needs to do**. Before you set authorization rules, you need to have a clear understanding of who or what is making requests, what that role is allowed to do, and to what extent it can do it. Those factors lead to choices that combine identity, role, and scope.

Consider a workload identity as an exmaple. The application must have data plane access to the database, so read and write actions to the data resource must be allowed. However, does the application need control plane access to the secret store? If the workload identity is compromised by a bad actor, what would the impact to the system be, in terms of confidentiality, integrity, and availability?

Role assignment

A role is a *set of permissions* that's assigned to an identity. Assign roles that only allow the identity to complete the task, and no more. When user's permissions are restricted to their job requirements, it's easier to identify suspicious or unauthorized behavior in the system.

Ask questions like these:

- Is read-only access enough?
- Does the identity need permissions to delete resources?

Limiting the level of access that users, applications, or services have to Azure resources reduces the potential attack surface. If you grant only the minimum permissions that are required to perform specific tasks, the risk of a successful attack or unauthorized access is significantly reduced. For example, security teams only need

read-only access to security attributes for all technical environments. That level is enough to assess risk factors, identify potential mitigations, and report on the risks.

There are scenarios in which users need more access because of the organizational structure and team organization. There might be an overlap between various roles, or single users might perform multiple standard roles. In this case, use multiple role assignments that are based on the business function instead of creating a custom role for each of these users. Doing so makes the roles easier to manage.

Avoid permissions that specifically reference individual resources or users. Granular and custom permissions create complexity and confusion because they don't pass on the intention to new resources that are similar. This can create a complex legacy configuration that's difficult to maintain and negatively impact both security and reliability.

>  **Tradeoff**: A granular access control approach enables better auditing and monitoring of user activities.

A role also has an *associated scope*. The role can operate at the allowed management group, subscription, resource group, or resource scope, or at another custom scope. Even if the identity has a limited set of permissions, widening the scope to include resources that are outside the identity's job function is risky. For example, read access to all source code and data can be dangerous and must be controlled.

You assign roles to identities by using role-based access control (RBAC). **Always use IdP-provided RBAC** to take advantage of features that enable you to apply access control consistently and revoke it rigorously.

Use built-in roles. They're designed to cover most use cases. Custom roles are powerful and sometimes useful, but you should reserve them for scenarios in which built-in roles won't work. Customization leads to complexity that increases confusion and makes automation more complex, challenging, and fragile. These factors all negatively impact security.

Grant roles that start with least privilege and add more based your operational or data access needs. Your technical teams must have clear guidance to implement permissions.

If you want fine-grained control on RBAC, add conditions on the role assignment based on context, such as actions and attributes.

Make conditional access choices

Don't give all identities the same level of access. Base your decisions on two main factors:

- **Time**. How long the identity can access your environment.
- **Privilege**. The level of permissions.

Those factors aren't mutually exclusive. A compromised identity that has more privileges and unlimited duration of access can gain more control over the system and data or use that access to continue to change the environment. Constrain those access factors both as a preventive measure and to control the blast radius.

- *Just in Time (JIT)* approaches provide the required privileges only when they're needed.
- *Just Enough Access (JEA)* provides only the required privileges.

Although time and privilege are the primary factors, there are other conditions that apply. For example, you can also use the device, network, and location from which the access originated to set policies.

Use strong controls that filter, detect, and block unauthorized access, including parameters like user identity and location, device health, workload context, data classification, and anomalies.

For example, your workload might need to be accessed by third-party identities like vendors, partners, and customers. They need the appropriate level of access rather than the default permissions that you provide to full-time employees. Clear differentiation of external accounts makes it easier to prevent and detect attacks that come from these vectors.

Your choice of IdP must be able to provide that differentiation, provide built-in features that grant permissions based on the least privilege, and provide built-in threat intelligence. This includes monitoring of access requests and sign-ins. The Azure IdP is Microsoft Entra ID. For more information, see the Azure facilitation section of this article.

Critical impact accounts

Administrative identities introduce some of the highest impact security risks because the tasks they perform require privileged access to a broad set of these systems and applications. Compromise or misuse can have a detrimental effect on your business and its information systems. Security of administration is one of the most critical security areas.

Protecting privileged access against determined adversaries requires you to take a complete and thoughtful approach to isolate these systems from risks. Here are some strategies:

- **Minimize the number of critical impact accounts.**
- **Use separate roles** instead of elevating privileges for existing identities.
- **Avoid permanent or standing access** by using the JIT features of your IdP. For break glass situations, follow an emergency access process.
- **Use modern access protocols** like passwordless authentication or multifactor authentication. Externalize those mechanisms to your IdP.
- Enforce key security attributes by using **conditional access policies**.
- **Decommission administrative accounts** that aren't being used.

Use a single identity across environments and associate a single identity with the user or principal. Consistency of identities across cloud and on-premises environments reduces human errors and the resulting security risks. Teams in both environments that manage resources need a consistent, authoritative source in order to meet security assurances. Work with your central identity team to ensure that identities in hybrid environments are synchronized.

>  **Risk**: There's a risk associated with synchronizing high privilege identities. An attacker can get full control of on-premises assets, and this can lead to a successful compromise of a cloud account. Evaluate your synchronization strategy by filtering out accounts that can add to the attack surface.

Establish processes to manage the identity lifecycle

Access to identities must not last longer than the resources that the identities access. Ensure that you have a process for disabling or deleting identities when there are changes in team structure or software components.

This guidance applies to source control, data, control planes, workload users, infrastructure, tooling, the monitoring of data, logs, metrics, and other entities.

Establish an identity governance process to manage the lifecycle of digital identities, high-privileged users, external/guest users, and workload users. Implement access reviews to ensure that when identities leave the organization or the team, their workload permissions are removed.

Protect nonidentity based secrets

Application secrets like preshared keys should be considered vulnerable points in the system. In the two-way communication, if the provider or consumer is compromised, significant security risks can be introduced. Those keys can also be burdensome because they introduce operational processes.

When you can, avoid using secrets and consider using identity-based authentication for user access to the application itself, not just to its resources.

The following list provides a summary of guidance. For more information, see Recommendations for application secrets.

- Treat these secrets as entities that can be dynamically pulled from a secret store. They shouldn't be hard coded in your application code, IaC scripts, deployment pipelines, or in any other artifact.
- Be sure that you have the **ability to revoke secrets**.
- Apply operational practices that handle tasks like **key rotation and expiration**.

For information about rotation policies, see Automate the rotation of a secret for resources that have two sets of authentication credentials and Tutorial: Updating certificate auto-rotation frequency in Key Vault.

Keep development environments safe

All code and scripts, pipeline tooling, and source control systems should be considered workload assets. **Access to writes should be gated** with automation and peer review. **Read access to source code should be limited** to roles on a need-to-know basis. Code repositories must have versioning, and **security code reviews** by peers must be a regular practice that's integrated with the development lifecycle. You need to have a process in place that **scans resources regularly** and identifies the latest vulnerabilities.

Use workload identities to grant access to resources from deployment environments, such as GitHub.

Maintain an audit trail

One aspect of identity management is ensuring that the system is auditable. Audits validate whether assume-breach strategies are effective. Maintaining an audit trail helps you:

- Verify that identity is authenticated with strong authentication. **Any action must be traceable** to prevent repudiation attacks.
- **Detect weak or missing authentication protocols** and get visibility into and insights about user and application sign-ins.
- Evaluate access from identities to the workload based on security and **compliance requirements** and consider user account risk, device status, and other criteria and policies that you set.
- **Track progress or deviation** from compliance requirements.

Most resources have data plane access. You need to know the identities that access resources and the actions that they perform. You can use that information for security diagnostics.

For more information, see Recommendations on security monitoring and threat analysis.

Azure facilitation

We recommend that you always use modern authentication protocols that take into account all available data points and use conditional access. **Microsoft Entra ID provides identity and access management in Azure**. It covers the management plane of Azure and is integrated with the data planes of most Azure services. Microsoft Entra ID is the tenant that's associated with the workload subscription. It tracks and manages identities and their allowed permissions and simplifies overall management to minimize the risk of oversight or human error.

These capabilities natively integrate into the same Microsoft Entra identity and permission model for user segments:

- Microsoft Entra ID. Employees and enterprise resources.
- Microsoft Entra External ID. Partners.
- Azure AD B2C. Customers.
- Microsoft Entra federation compatibility list. Third-party federation solutions.

You can use Microsoft Entra ID for authentication and authorization of custom applications via Microsoft Authentication Library (MSAL) or platform features, like authentication for web apps. It covers the management plane of Azure, the data planes of most of Azure services, and integration capabilities for your applications.

You can stay current by visiting What's new in Microsoft Entra ID.

> **Tradeoff**: Microsof Microsoft Entra ID is a single point of failure just like any other foundational service. There's no workaround until the outage is fixed by Microsoft. However, the rich feature set of Microsoft Entra outweighs the risk of using custom solutions.

Azure supports open protocols like OAuth2 and OpenID Connect. We recommend that you use these standard authentication and authorization mechanisms instead of designing your own flows.

Azure RBAC

Azure RBAC represents security principals in Microsoft Entra ID. All role assignments are done via Azure RBAC. Take advantage of built-in roles that provide most of the permissions that you need. For more information, see Microsoft Entra built-in roles.

Here are some use cases:

- By assigning users to roles, you can control access to Azure resources. For more information, see Overview of role-based access control in Microsoft Entra ID.
- You can use Privileged Identity Management to provide time-based and approval-based role activation for roles that are associated with high-impact identities. For more information, see What is Privileged Identity Management?.

For more information about RBAC, see Best practices for Azure RBAC.

For information about attribute-based controls, see What is Azure ABAC?.

Workload identity

Microsoft Entra ID can handle your application's identity. The service principal that's associated with the application can dictate its access scope.

For more information, see What are workload identities?.

The service principal is also abstracted when you use a managed identity. The advantage is that Azure manages all credentials for the application.

Not all services support managed identities. If you can't use managed identities, you can use service principals. However, using service principals increases your management overhead. For more information, see What are managed identities for Azure resources?.

Resource identity

The concept of **managed identities can be extended to Azure resources**. Azure resources can use managed identities to authenticate themselves to other services that support Microsoft Entra authentication. For more information, see Azure services that can use managed identities to access other services.

Conditional access policies

Conditional access describes your policy for an access decision. To use conditional access, you need to understand the restrictions that are required for the use case. Configure Microsoft Entra Conditional Access by setting up an access policy for that's based on your operational needs.

For more information, see Conditional access: Users, groups, and workload identities.

Group access management

Instead of granting permissions to specific users, assign access to groups in Microsoft Entra ID. If a group doesn't exist, work with your identity team to create one. You can then add and remove group members outside of Azure and make sure that permissions are current. You can also use the group for other purposes, like mailing lists.

For more information, see Secure access control using groups in Microsoft Entra ID.

Threat detection

Microsoft Entra ID Protection can help you detect, investigate, and remediate identity-based risks. For more information, see What is Identity Protection?.

Threat detection can take the form of reacting to an alert of suspicious activity or proactively searching for anomalous events in activity logs. User and Entity Behavior Analytics (UEBA) in Microsoft Sentinel makes it easy to detect suspicious activities. For more information, see Identify advanced threats with UEBA.

Hybrid systems

On Azure, **don't synchronize accounts to Microsoft Entra ID that have high privileges in your existing Active Directory**. This synchronization is blocked in the default Microsoft Entra Connect Sync configuration, so you only need to confirm that you haven't customized this configuration.

For information about filtering in Microsoft Entra ID, see Microsoft Entra Connect Sync: Configure filtering.

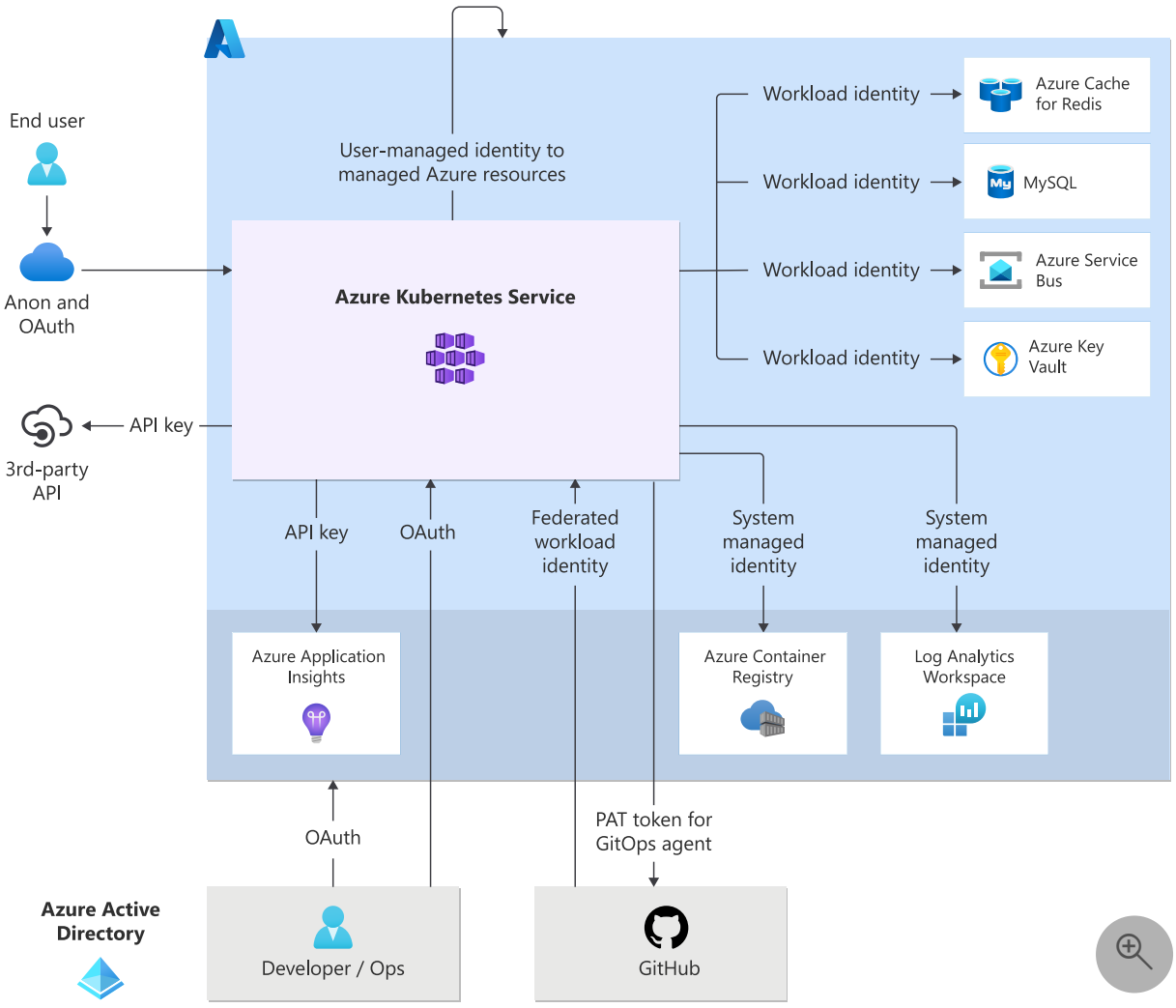
Identity logging

Enable diagnostic settings on Azure resources to emit information that you can use as an audit trail. The diagnostic information shows which identities attempt to access which resources and the outcome of those attempts. The collected logs are sent to Azure Monitor.

> **Tradeoff**: Logging incurs costs because of the data storage that's used to store the logs. It also might cause a performance impact, especially on the code and on logging solutions that you add to the application.

Example

The following example shows an identity implementation. Different types of identities are used together to provide the required levels of access.



Identity components

- **System-managed identities**. Microsoft Entra ID provides access to service data planes that don't face users, like Azure Key Vault and data stores. These identities also control access, via RBAC, to the Azure management plane for workload components, deployment agents, and team members.
- **Workload identities**. The application services in the Azure Kubernetes Service (AKS) cluster use workload identities to authenticate themselves to other components in the solution.
- **Managed identities**. System components in the client role use system-managed identities, including build agents.
- **Human identities**. User and operator authentication is delegated to Microsoft Entra ID or Microsoft Entra ID (native, B2B, or B2C).

The security of preshared secrets is critical for any application. Azure Key Vault provides a secure storage mechanism for these secrets, including Redis and third-party secrets.

A rotation mechanism is used to help ensure that secrets aren't compromised. Tokens for the Microsoft identity platform implementation of OAuth 2 and OpenID Connect are used to authenticate users.

Azure Policy is used to ensure that identity components like Key Vault use RBAC instead of access policies. JIT and JEA provide traditional standing permissions for human operators.

Access logs are enabled across all components via Azure Diagnostics, or via code for code components.

Related links

- Tutorial: Automate the rotation of a secret for resources that have two sets of authentication credentials
- Tutorial: Updating certificate auto-rotation frequency in Key Vault
- What's new in Microsoft Entra ID?
- Microsoft Entra built-in roles
- Overview of role-based access control in Microsoft Entra ID
- What are workload identities?
- What are managed identities for Azure resources?
- Conditional access: Users, groups, and workload identities
- Microsoft Entra Connect Sync: Configure filtering

Security checklist

Refer to the complete set of recommendations.

Security checklist

Recommendations for networking and connectivity

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Security checklist recommendation:

> **SE:05** **Isolate, filter, and control network traffic across both ingress and egress flows. Apply defense in depth principles by using localized network controls at all available network boundaries across both east-west and north-south traffic.**

This guide describes the recommendations for network design. The focus is on **security controls that can filter, block, and detect adversaries crossing network boundaries** at various depths of your architecture.

You can strengthen your identity controls by implementing network-based access control measures. Along with identity-based access control, network security is a high priority for protecting assets. Proper network security controls can provide a defense-in-depth element that can help detect and contain threats, and prevent attackers from gaining entry into your workload.

Definitions

Term	Definition
East-west traffic	Network traffic that moves within a trusted boundary.
Egress flow	Outbound workload traffic.
Hostile network	A network that isn't deployed as part of your workload. A hostile network is considered a threat vector.
Ingress flow	Inbound workload traffic.
Network filtering	A mechanism that allows or blocks network traffic based on specified rules.
Network segmentation or isolation	A strategy that divides a network into small, isolated segments, with security controls applied at the boundaries. This technique helps protect resources from hostile networks, such as the internet.
Network transformation	A mechanism that mutates network packets to obscure them.
North-south traffic	Network traffic that moves from a trusted boundary to external networks that are potentially hostile, and vice versa.

Key design strategies

Network security uses **obscurity to protect workload assets from hostile networks**. Resources that are behind a network boundary are hidden until the boundary controls mark the traffic as safe to move forward. Network security design is built on three main strategies:

- **Segment**. This technique **isolates traffic on separate networks by adding boundaries**. For example, traffic to and from an application tier passes a boundary to communicate with other tiers, which have different security requirements. Layers of segmentation actualize the defense-in-depth approach.

  The foremost security boundary is the **networking edge between your application and public networks**. It's important to clearly define this perimeter so that you establish a boundary for isolating hostile networks. The controls on this edge must be highly effective, because this boundary is your first line of defense.

  Virtual networks provide a logical boundary. By design, a virtual network can't communicate with another virtual network unless the boundary has been intentionally broken through peering. Your architecture should take advantage of this strong, platform-provided security measure.

  You can also use other logical boundaries, such as carved-out **subnets within a virtual network**. A benefit of subnets is that you can use them to group together resources that are within an isolation boundary and have similar security assurances. You can then configure controls on the boundary to filter traffic.

- **Filter**. This strategy helps ensure that **traffic that enters a boundary is expected, allowed, and safe**. From a Zero-Trust perspective, filtering explicitly verifies all available data points at the network level. You can place rules on the boundary to check for specific conditions.

  For example, at the header level, the rules can verify that the traffic originates from an expected location or has an expected volume. But these checks aren't sufficient. Even if the traffic exhibits expected characteristics, the payload might not be safe. Validation checks might reveal an SQL injection attack.

- **Transform**. **Mutate packets at the boundary as a security measure.** For example, you can remove HTTP headers to eliminate the risk of exposure. Or you can turn off Transport Layer Security (TLS) at one point and reestablish it at another hop with a certificate that's managed more rigorously.

Classify the traffic flows

The first step in classifying flows is to study a schematic of your workload architecture. From the schematic, **determine the intent and characteristics of the flow** with respect to the functional utility and operational aspects of your workload. Use the following questions to help classify the flow:

- If the workload needs to communicate with external networks, what should the required level of proximity to those networks be?
- What are the network characteristics of the flow, such as the expected protocol and the source and shape of the packets? Are there any compliance requirements at the networking level?

There are many ways to classify traffic flows. The following sections discuss commonly used criteria.

Visibility from external networks

- **Public**. A workload is public facing if its application and other components are reachable from the public internet. The application is exposed through one or more public IP addresses and public Domain Name System (DNS) servers.
- **Private**. A workload is private if it can only be accessed through a private network such as a virtual private network (VPN). It's exposed only through one or more private IP addresses and potentially through a private DNS server.

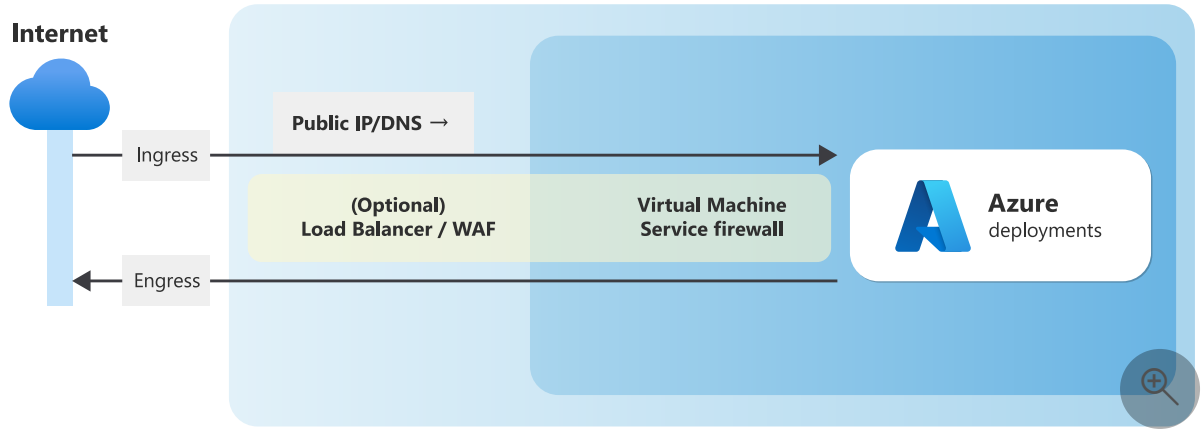
  In a private network, there's no line of sight from the public internet to the workload. For the gateway, you can use a load balancer or firewall. These options can provide security assurances.

Even with public workloads, **strive to keep as much of the workload private as possible**. This approach forces packets to cross through a private boundary when they arrive from a public network. A gateway in that path can function as a transition point by acting as a reverse proxy.

Traffic direction

- **Ingress**. Ingress is inbound traffic that flows toward a workload or its components. To help secure ingress, apply the preceding set of key strategies. Determine what the traffic source is and whether it's expected, allowed, and safe. Attackers who scan public cloud provider IP address ranges can successfully penetrate your defenses if you don't check ingress or implement basic network security measures.

- **Egress**. Egress is outbound traffic that flows away from a workload or its components. To check egress, determine where the traffic is headed and whether the destination is expected, allowed, and safe. The destination might be malicious or associated with data exfiltration risks.



You can also **determine your level of exposure by considering your workload's proximity to the public internet**. For example, the application platform typically serves public IP addresses. The workload component is the face of the solution.

Scope of influence

- **North-south**. Traffic that flows between a workload network and external networks is north-south traffic. This traffic crosses the edge of your network. External networks can be the public internet, a corporate network, or any other network that's outside your scope of control.

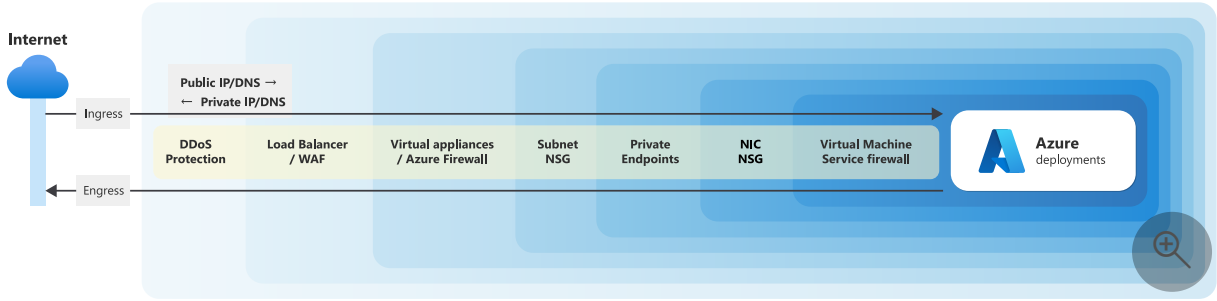
  Ingress and egress can both be north-south traffic.

  As an example, consider the egress flow of a hub-spoke network topology. You can define the networking edge of your workload so that the hub is an external network. In that case, outbound traffic from the virtual network of the spoke is north-south traffic. But if you consider the hub network within your sphere of control, north-south traffic is extended to the firewall in the hub, because the next hop is the internet, which is potentially hostile.

- **East-west**. Traffic that flows within a workload network is east-west traffic. This type of traffic results when components in your workload communicate with each other. An example is traffic between the tiers of an *n*-tier application. In microservices, service-to-service communication is east-west traffic.

To provide defense in depth, maintain **end-to-end control of security affordances that are included in each hop or that you use when packets cross internal segments**.

Different risk levels require different risk remediation methods.



The preceding diagram illustrates network defense in depth in the private cloud. In this diagram, the border between the public and private IP address spaces is significantly farther from the workload than in the public cloud diagram. Multiple layers separate the Azure deployments from the public IP address space.

> **Note**
>
> Identity is always the primary perimeter. Access management must be applied to networking flows. Use managed identities when you use Azure role-based access control (RBAC) between components of your network.

After you classify flows, perform a segmentation exercise to identify firewall injection points on the communication paths of your network segments. When you **design your network defense in depth across all segments and all traffic types, assume a breach at all points**. Use a combination of various localized network controls at all available boundaries. For more information, see Segmentation strategies.

Apply firewalls at the edge

Internet edge traffic is north-south traffic and includes ingress and egress. To detect or block threats, an edge strategy must mitigate as many attacks as possible to and from the internet.

For egress, **send all internet-bound traffic through a single firewall** that provides enhanced oversight, governance, and control of traffic. For ingress, force all traffic from the internet to go through a network virtual appliance (NVA) or a web application firewall.

- Firewalls are usually singletons that are deployed per region in an organization. As a result, they're shared among workloads and owned by a central team. Make sure that any NVAs that you use are configured to support the needs of your workload.

- We recommend that you use Azure native controls as much as possible.

  In addition to native controls, you can also consider partner NVAs that provide advanced or specialized features. Partner firewall and web application firewall vendor products are available in Azure Marketplace.

  The decision to use native features as opposed to partner solutions should be based on your organization's experience and requirements.

  >  **Tradeoff**: Partner capabilities often provide advanced features that can protect against sophisticated, but typically uncommon, attacks. The configuration of partner solutions can be complex and fragile, because these solutions don't integrate with the cloud's fabric controllers. From a cost perspective, native control is preferred because it's cheaper than partner solutions.

Any technological options that you consider should provide security controls and monitoring for both ingress and egress flows. To see options that are available for Azure, see the Edge security section in this article.

Design virtual network and subnet security

The primary objective of a private cloud is to obscure resources from the public internet. There are several ways of achieving this goal:

- **Move to private IP address spaces**, which you can accomplish by using virtual networks. Minimize network line of sight even within your own private networks.
- **Minimize the number of public DNS entries that you use** to expose less of your workload.
- **Add ingress and egress network flow control**. Don't allow traffic that's not trusted.

Segmentation strategy

To minimize network visibility, **segment your network and start with least-privilege network controls**. If a segment isn't routable, it can't be accessed. Broaden the scope to include only segments that need to communicate with each other through network access.

You can segment virtual networks by creating subnets. The criteria for division should be intentional. When you collocate services inside a subnet, make sure that those services can see each other.

You can base your segmentation on many factors. For example, you can place different application tiers in dedicated segments. Another approach is to plan your subnets based on common roles and functions that use well-known protocols.

For more information, see Segmentation strategies.

Subnet firewalls

It's important to inspect each subnet's inbound and outbound traffic. Use the three main strategies discussed earlier in this article, in Key design strategies. Check whether the flow is expected, allowed, and safe. To verify this information, **define firewall rules that are based on the protocol, source, and destination** of the traffic.

On Azure, you set firewall rules in network security groups. For more information, see the Network security groups section in this article.

For an example of a subnet design, see Azure Virtual Network subnets.

Use controls at the component level

After you minimize the visibility of your network, map out your Azure resources from a network perspective and evaluate the flows. The following types of flows are possible:

- **Planned traffic**, or intentional communication between services according to your architecture design. For example, you have planned traffic when your architecture recommends that Azure Functions pulls messages from Azure Service Bus.
- **Management traffic**, or communication that happens as part of the service's functionality. This traffic isn't part of your design, and you have no control over it. An example of managed traffic is the communication between the Azure services in your architecture and the Azure management plane.

Distinguishing between planned and management traffic helps you build localized, or service-level, controls. Have a good understanding of the source and destination at each hop. Especially understand how your data plane is exposed.

As a starting point, determine whether each service is exposed to the internet. If it is, plan how to restrict access. If it isn't, place it in a virtual network.

Service firewalls

If you expect a service to be exposed to the internet, **take advantage of the service-level firewall that's available for most Azure resources**. When you use this firewall, you

can set rules based on access patterns. For more information, see the Azure service firewalls section in this article.

> **Note**
>
> When your component isn't a service, use a host-based firewall in addition to network-level firewalls. A virtual machine (VM) is an example of a component that's not a service.

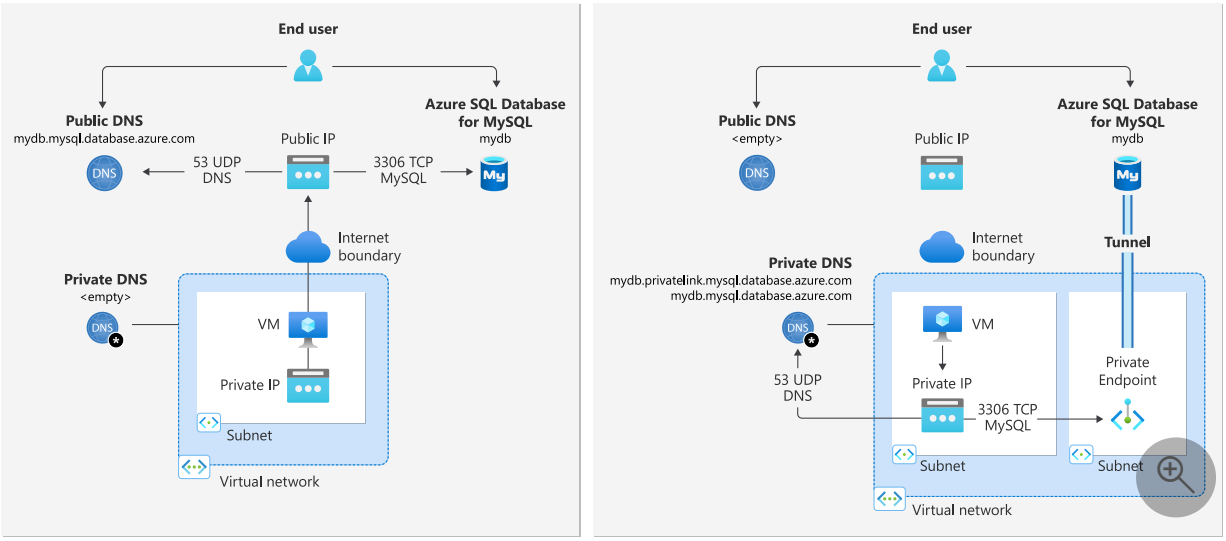
Connectivity to platform as a service (PaaS) services

Consider using **private endpoints to help secure access to PaaS services**. A private endpoint is assigned a private IP address from your virtual network. The endpoint allows other resources in the network to communicate with the PaaS service over the private IP address.

Communication with a PaaS service is achieved by using the service's public IP address and DNS record. That communication occurs over the internet. You can make that communication private.

A tunnel from the PaaS service into one of your subnets creates a private channel. All communication takes place from the component's private IP address to a private endpoint in that subnet, which then communicates with the PaaS service.

In this example, the image on the left shows the flow for publicly exposed endpoints. On the right, that flow is secured by using private endpoints.



For more information, see the Private endpoints section in this article.

> **Note**
>
> We recommend that you use private endpoints in conjunction with service firewalls. A service firewall blocks incoming internet traffic and then exposes the service privately to internal users who use the private endpoint.

Another advantage of using private endpoints is that you don't need to open the ports on the firewall for outbound traffic. **Private endpoints lock down all outbound traffic on the port for the public internet.** Connectivity is limited to resources within the network.

>  **Tradeoff**: Azure Private Link is a paid service that has meters for inbound and outbound data that's processed. You're also charged for private endpoints.

Protect against distributed denial of service (DDoS) attacks

A DDoS attack attempts to exhaust an application's resources to make the application unavailable to legitimate users. DDoS attacks can target any endpoint that's publicly reachable through the internet.

A DDoS attack is usually a massive, widespread, geographically dispersed abuse of your system's resources that makes it hard to pinpoint and block the source.

For Azure support to help protect against these attacks, see the Azure DDoS Protection section in this article.

Azure facilitation

You can use the following Azure services to add defense-in-depth capabilities to your network.

Azure Virtual Network

Virtual Network helps your Azure resources securely communicate with each other, the internet, and on-premises networks.

By default, all resources in a virtual network can engage in outbound communication with the internet. But inbound communication is restricted by default.

Virtual Network offers features for filtering traffic. You can restrict access at the virtual-network level by using a user-defined route (UDR) and a firewall component. At the subnet level, you can filter traffic by using network security groups.

Edge security

By default, ingress and egress both flow over public IP addresses. Depending on the service or topology, either you set these addresses or Azure assigns them. Other ingress and egress possibilities include passing traffic through a load balancer or network address translation (NAT) gateway. But these services are intended for traffic distribution and not necessarily for security.

The following technology choices are recommended:

- Azure Firewall. You can use Azure Firewall at the network edge and in popular network topologies, such as hub-spoke networks and virtual WANs. You typically **deploy Azure Firewall as an egress firewall** that acts as the final security gate before traffic goes to the internet. Azure Firewall can route traffic that uses non-HTTP and non-HTTPS protocols, such as Remote Desktop Protocol (RDP), Secure Shell Protocol (SSH), and File Transfer Protocol (FTP). The feature set of Azure Firewall includes:
  - Destination network address translation (DNAT), or port forwarding.
  - Intrusion detection and prevention system (IDPS) signature detection.
  - Strong layer 3, layer 4, and fully qualified domain name (FQDN) network rules.

  > **Note**
  >
  > Most organizations have a forced tunneling policy that forces traffic to flow through an NVA.

  If you don't use a virtual WAN topology, **you must deploy a UDR** with a `NextHopType` of `Internet` to your NVA's private IP address. UDRs are applied at the subnet level. By default, subnet-to-subnet traffic doesn't flow through the NVA.

  You can also use Azure Firewall simultaneously for ingress. It can route HTTP and HTTPS traffic. In higher-tiered SKUs, Azure Firewall offers TLS termination so that you can implement payload-level inspections.

  The following practices are recommended:

  - **Enable diagnostics settings** in Azure Firewall to collect traffic flow logs, IDPS logs, and DNS request logs.

- Be as specific as possible in rules.
- Where it's practical, avoid FQDN service tags. But when you use them, use the regional variant, which allows communication with all endpoints of the service.
- Use IP groups to define sources that must share the same rules over the life of the IP group. IP groups should reflect your segmentation strategy.
- Override the infrastructure FQDN allow rule only if your workload requires absolute egress control. Overriding this rule comes with a reliability tradeoff, because Azure platform requirements change on services.

>  **Tradeoff**: Azure Firewall can impact your performance. Rule order, quantity, TLS inspection, and other factors can cause significant latency.
>
> There can also be an impact on the reliability of your workload. It might experience source network address translation (SNAT) port exhaustion. To help overcome this problem, add public IP addresses as needed.

>  **Risk**: For egress traffic, Azure assigns a public IP address. This assignment can have a downstream impact on your external security gate.

- Azure Web Application Firewall. This service supports inbound filtering and only targets HTTP and HTTPS traffic.

  It offers basic security for common attacks, such as threats that the Open Worldwide Application Security Project (OWASP) identifies in the OWASP Top 10 document. Azure Web Application Firewall also provides other security features that are focused on layer 7, such as rate limiting, SQL-injection rules, and cross-site scripting.

  With Azure Web Application Firewall, TLS termination is required, because most checks are based on payloads.

  You can integrate Azure Web Application Firewall with routers, such as Azure Application Gateway or Azure Front Door. Azure Web Application Firewall implementations for those kinds of routers can vary.

Azure Firewall and Azure Web Application Firewall aren't mutually exclusive choices. For your edge security solution, various options are available. For examples, see Firewall and Application Gateway for virtual networks.

Network security groups

A network security group is a layer 3 and layer 4 firewall that you apply at the subnet or network interface card (NIC) level. Network security groups aren't created or applied by default.

Network security group rules act as a firewall to stop traffic that flows in and out at the perimeter of a subnet. A network security group has a default rule set that's overly permissive. For example, the default rules don't set a firewall from the egress perspective. For ingress, no inbound internet traffic is allowed.

To create rules, start with the default rule set:

- For **inbound** traffic, or ingress:
  - Virtual network traffic from direct, peered, and VPN gateway sources is allowed.
  - Azure Load Balancer health probes are allowed.
  - All other traffic is blocked.
- For **outbound** traffic, or egress:
  - Virtual network traffic to direct, peered, and VPN gateway destinations is allowed.
  - Traffic to the internet is allowed.
  - All other traffic is blocked.

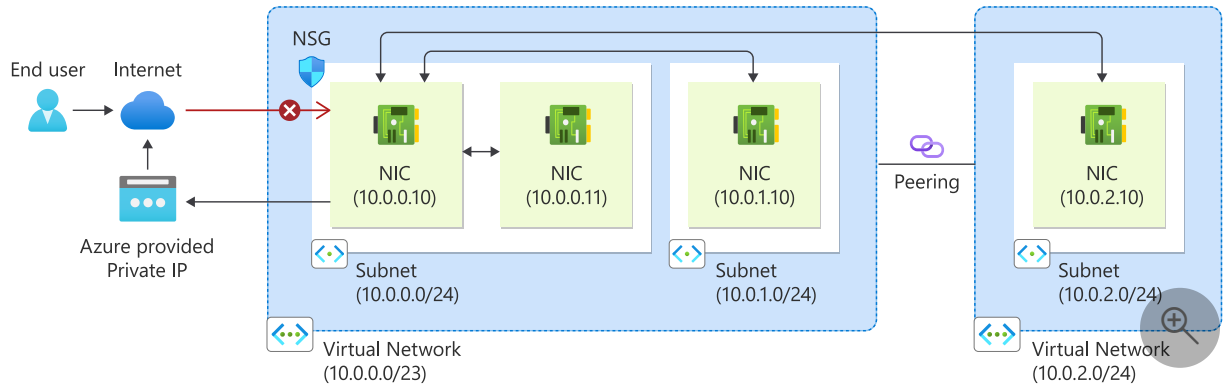
Then consider the following five factors:

- Protocol
- Source IP address
- Source port
- Destination IP address
- Destination port

The lack of support for FQDN limits network security group functionality. You need to provide specific IP address ranges for your workload, and they're hard to maintain.

But for Azure services, you can use service tags to summarize source and destination IP address ranges. A security benefit of service tags is that they're **opaque to the user, and the responsibility is offloaded to Azure**. You can also assign an application security group as a destination type to route traffic to. This type of named group contains resources that have similar inbound or outbound access needs.

>  **Risk**: Service tag ranges are very broad so that they accommodate the widest possible range of customers. Updates to service tags lag behind changes in the service.



In the preceding image, network security groups are applied at the NIC. Internet traffic and subnet-to-subnet traffic are denied. The network security groups are applied with the `VirtualNetwork` tag. So in this case, the subnets of peered networks have a direct line of sight. The broad definition of the `VirtualNetwork` tag can have a significant security impact.

When you use service tags, use regional versions when possible, such as `Storage.WestUS` instead of `Storage`. By taking this approach, you limit the scope to all endpoints in a particular region.

Some tags are exclusively for **inbound** or **outbound** traffic. Others are for *both* types. **Inbound** tags usually allow traffic from all hosting workloads, such as `AzureFrontDoor.Backend`, or from Azure to support service runtimes, such as `LogicAppsManagement`. Similarly, **outbound** tags allow traffic to all hosting workloads or from Azure to support service runtimes.

Scope the rules as much as possible. In the following example, the rule is set to specific values. Any other type of traffic is denied.

Information	**Example**
Protocol	Transmission Control Protocol (TCP), UDP
Source IP address	Allow ingress to the subnet from <source-IP-address-range>: 4575/UDP
Source port	Allow ingress to the subnet from <service-tag>: 443/TCP
Destination IP address	Allow egress from the subnet to <destination-IP-address-range>: 443/TCP
Destination port	Allow egress from the subnet to <service-tag>: 443/TCP

To summarize:

- **Be precise when you create rules**. Only allow traffic that's necessary for your application to function. Deny everything else. This approach limits the network line

of sight to network flows that are needed to support the operation of the workload. **Supporting more network flows than necessary leads to unnecessary attack vectors and extends the surface area.**

Restricting traffic doesn't imply that allowed flows are beyond the scope of an attack. Because network security groups work at layers 3 and 4 on the Open Systems Interconnection (OSI) stack, they only contain shape and direction information. For example, if your workload needs to allow DNS traffic to the internet, you would use a network security group of `Internet:53:UDP`. In this case, an attacker might be able to exfiltrate data through UDP on port 53 to some other service.

- Understand that network security groups can differ slightly from one another. It's easy to overlook the intent of the differences. **To have granular filtering, it's safer to create extra network security groups.** Set up at least one network security group.
  - Adding a network security group unlocks many diagnostics tools, such as flow logs and network traffic analytics.
  - Use Azure Policy to help control traffic in subnets that don't have network security groups.
- If a subnet supports network security groups, add a group, even if it's minimally impactful.

Azure service firewalls

Most Azure services offer a service-level firewall. This feature inspects ingress traffic to the service from specified classless inter-domain routing (CIDR) ranges. These firewalls offer benefits:

- They provide a **basic level of security**.
- There's a **tolerable performance impact**.
- Most services offer these firewalls at **no extra cost**.
- The firewalls emit logs through Azure diagnostics, which can be useful for analyzing access patterns.

But there are also security concerns associated with these firewalls, and there are limitations associated with providing parameters. For example, if you use Microsoft-hosted build agents, you have to open the IP address range for all Microsoft-hosted build agents. The range is then open to your build agent, other tenants, and adversaries who might abuse your service.

If you have access patterns for the service, which can be configured as service firewall rule sets, you should enable the service. You can use Azure Policy to enable it. Make sure you don't enable the trusted Azure services option if it isn't enabled by default. Doing so brings in all dependent services that are in the scope of the rules.

For more information, see the product documentation of individual Azure services.

Private endpoints

Private Link provides a way for you to give a PaaS instance a private IP address. The service is then unreachable over the internet. Private endpoints aren't supported for all SKUs.

Keep the following recommendations in mind when you use private endpoints:

- Configure services that are bound to virtual networks to **contact PaaS services through private endpoints**, even if those PaaS services also need to offer public access.
- Promote the use of **network security groups for private endpoints to restrict access** to private endpoint IP addresses.
- Always **use service firewalls when you use private endpoints**.
- When possible, if you have a service that's only accessible via private endpoints, remove the DNS configuration for its public endpoint.
- Consider runtime **line-of-sight concerns** when you implement private endpoints. But also consider **DevOps and monitoring concerns**.
- Use Azure Policy to **enforce resource configuration**.

>  **Tradeoff**: Service SKUs with private endpoints are expensive. Private endpoints can complicate operations because of network obscurity. You need to add self-hosted agents, jump boxes, a VPN, and other components to your architecture.

DNS management can be complex in common network topologies. You might have to introduce DNS forwarders and other components.

Virtual network injection

You can use the **virtual network injection process** to deploy some Azure services into your network. Examples of such services include Azure App Service, Functions, Azure API

Management, and Azure Spring Apps. This process **isolates the application** from the internet, systems in private networks, and other Azure services. Inbound and outbound traffic from the application is allowed or denied based on network rules.

Azure Bastion

You can use Azure Bastion to connect to a VM by using your browser and the Azure portal. Azure Bastion **enhances the security of RDP and SSH connections**. A typical use case includes connecting to a jump box in the same virtual network or a peered virtual network. Using Azure Bastion removes the need for the VM to have a public IP address.

Azure DDoS Protection

Every property in Azure is protected by Azure DDoS infrastructure protection at no extra cost and with no added configuration. The level of protection is basic, but the protection has high thresholds. It also doesn't provide telemetry or alerting, and it's workload-agnostic.

Higher-tiered SKUs of DDoS Protection are available but aren't free. The scale and capacity of the globally deployed Azure network offers protection against common network-layer attacks. Technologies like always-on traffic monitoring and real-time mitigation provide this capability.

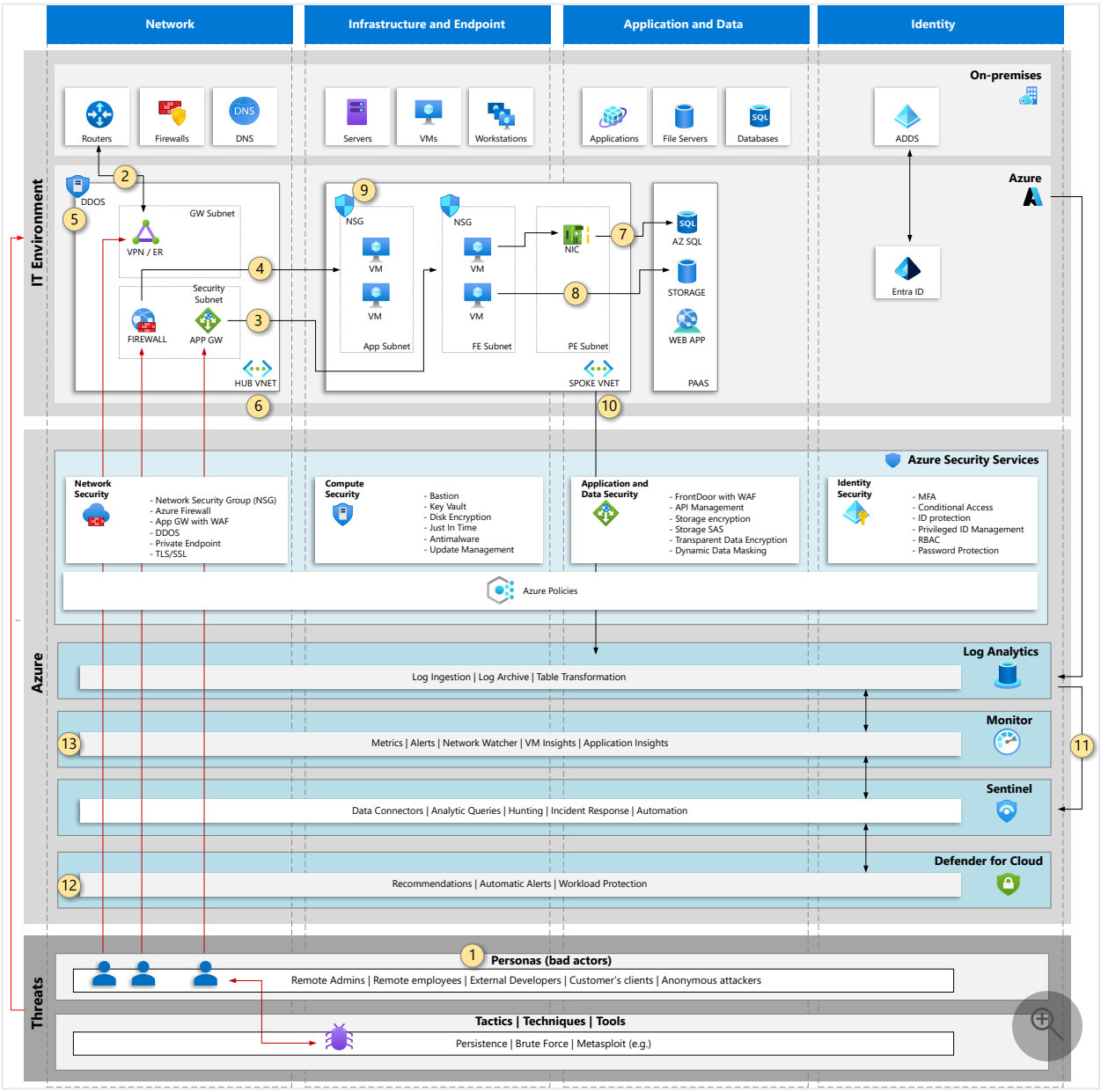
For more information, see Azure DDoS Protection overview.

Example

Here are some examples that demonstrate the use of network controls recommended in this article.

IT environment

This example builds on the Information Technology (IT) environment established in the security baseline (SE:01). This approach provides a broad understanding of network controls applied at various perimeters to restrict traffic.



1. **Network attack personas**. Several personas may be considered in a network attack, including Admins, employees, customer's clients and anonymous attackers.

2. **VPN access**. A bad actor might access the on-premises environment through a VPN or an Azure environment that's connected to the on-premises environment through a VPN. Configure with IPSec protocol to enable secure communication.

3. **Public access to the application**. Have a web application firewall (WAF) in front of the application to protect it on Layer 7 of the network OSI layer.

4. **Operator access**. Remote access through Layer 4 of network OSI layers must be secured. Consider using Azure Firewall with IDP/IDS features.

5. **DDoS protection**. Have DDoS protection for your entire VNet.

6. **Network topology**. A network topology such as hub-spoke, is more secure, and optimize costs. The hub network provides centralized firewall protection to all the

peered spokes.

7. **Private endpoints**: Consider adding publically exposed services into your private network by using private endpoints. These create a Network Card (NIC) in your private VNet and bind with the Azure service.

8. **TLS communication**. Protect data in transit by communicating over TLS.

9. **Network Security Group (NSG)**: Protect segments within a VNet with NSG, a free resource that filters TCP/UDP inbound and outbound communication considering IP and port ranges. Part of NSG is the Application Security Group (ASG) that allows you to create tags for traffic rules for easier management.

10. **Log Analytics**. Azure resources emit telemetry that's ingested in Log Analytics then used with a SIEM solution like Microsoft Sentinel for analysis.

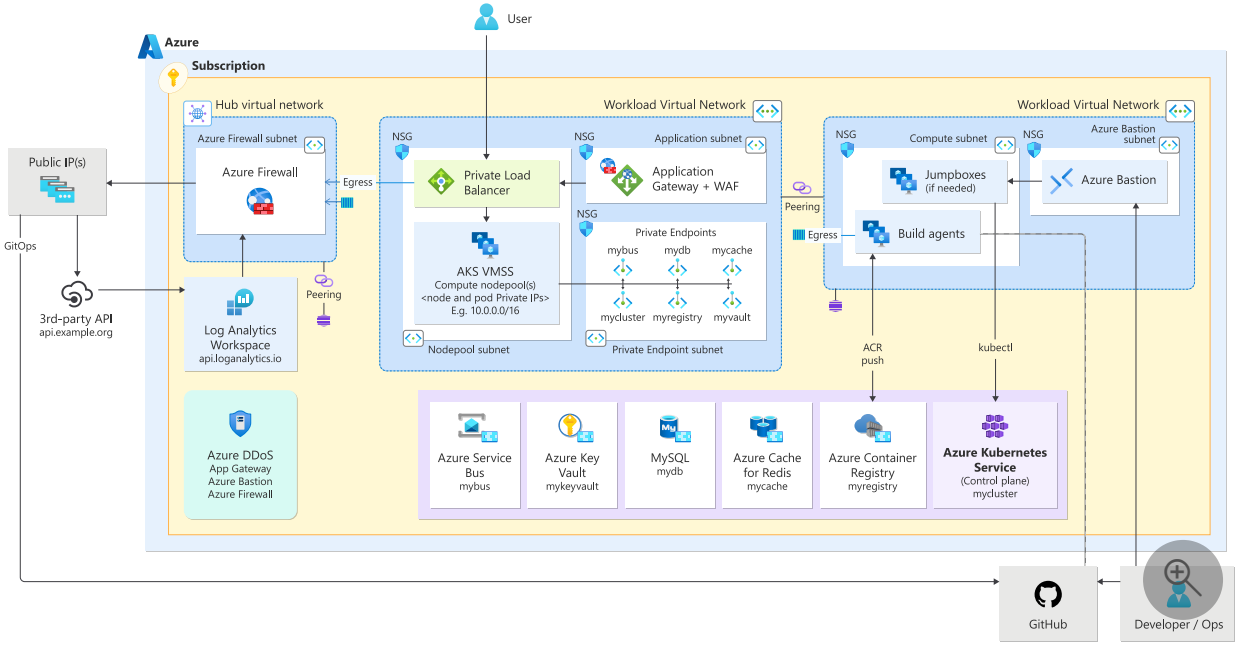
11. **Microsoft Sentinel Integration**. Log Analytics is integrated with Microsoft Sentinel and other solutions like Microsoft Defender for Cloud.

12. **Microsoft Defender for Cloud**. Microsoft Defender for Cloud delivers many workload protection solutions, including Network recommendations for your environment.

13. **Traffic Analytics**: Monitor your network controls with Traffic Analytics. This is configured through Network Watcher, part of Azure Monitor, and aggregates inbound and outbound hits in your subnets collected by NSG.

Architecture for a containerized workload

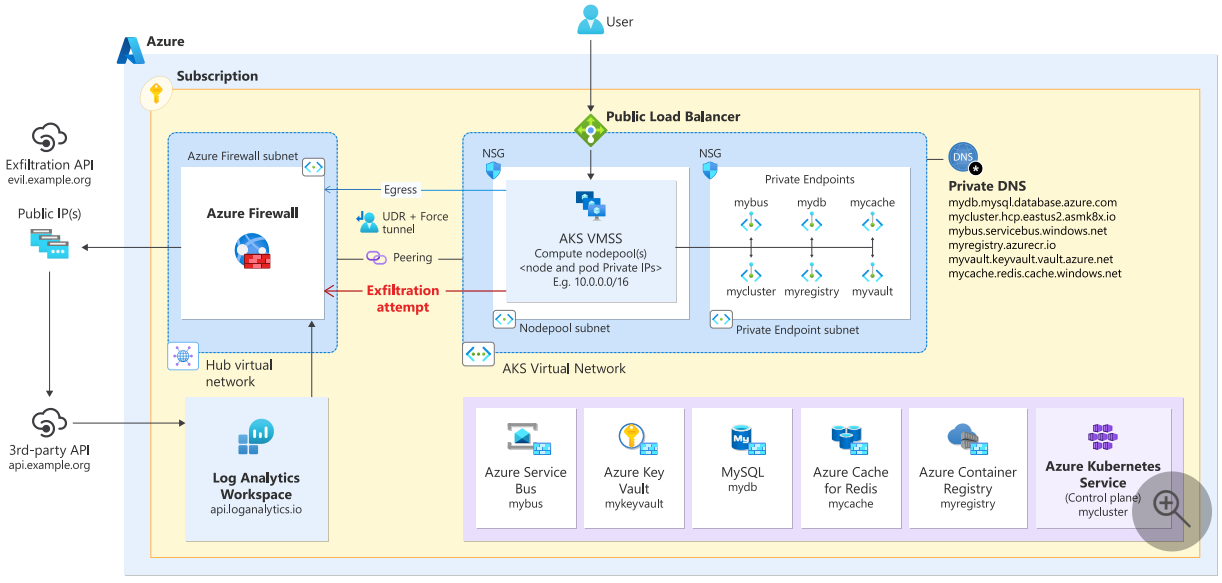
This example architecture combines the network controls that are described in this article. The example doesn't show the complete architecture. Instead, it focuses on ingress controls on the private cloud.



Application Gateway is a web traffic load balancer that you can use to manage traffic to your web applications. You deploy Application Gateway in a dedicated subnet that has network security group controls and web application firewall controls in place.

Communication with all PaaS services is conducted through **private endpoints**. All endpoints are placed in a dedicated subnet. DDoS Protection helps protect all public IP addresses that are configured for a basic or higher level of firewall protection.

Management traffic is restricted through Azure Bastion, which helps provide secure and seamless RDP and SSH connectivity to your VMs directly from the Azure portal over TLS. Build agents are placed in the virtual network so that they have a network view to workload resources such as compute resources, container registries, and databases. This approach helps provide a secure and isolated environment for your build agents, which boosts protection for your code and artifacts.



Network security groups at the subnet level of the compute resources restrict egress traffic. Forced tunneling is used to route all traffic through Azure Firewall. This approach helps provide a secure and isolated environment for your compute resources, which boosts protection for your data and applications.

Related links

- Recommendations for designing segmentation strategies
- Azure Virtual Network subnets
- Azure Virtual Network
- Azure Firewall
- Azure Web Application Firewall
- Firewall and Application Gateway for virtual networks
- Network security groups
- Service tags
- Azure Private Link
- Private endpoints
- Azure Bastion
- Azure DDoS Protection overview

Security checklist

Refer to the complete set of recommendations.

Security checklist

Recommendations for data encryption

Article • 02/05/2024

Applies to Well-Architected Framework Security checklist recommendation:

Expand table

SE:07	**Encrypt data by using modern industry-standard methods to guard confidentiality and integrity. Align encryption scope with data classifications; prioritize native platform encryption methods.**

If your data isn't protected, it can be maliciously modified, which leads to loss of integrity and confidentiality.

This guide describes the recommendations for encrypting and protecting your data. Encryption is the process of using cryptography algorithms to **make the data unreadable and lock the data with a key**. In the encrypted state, data can't be deciphered. It can only be decrypted by using a key that's paired with the encryption key.

Definitions

Expand table

Terms	Definition
Certificates	Digital files that hold the public keys for encryption or decryption.
Cipher suite	A set of algorithms that are used to encrypt and decrypt information to secure a network connection over Transport Layer Security (TLS).
Confidential computing	Confidential Computing is the protection of data in use by performing computation in a hardware-based, attested Trusted Execution Environment.
Decryption	The process in which encrypted data is unlocked with a secret code.
Double encryption	The process of encrypting data by using two or more independent layers of encryption.
Encryption	The process by which data is made unreadable and locked with a secret code.
Hashing	The process of transforming data to text or numbers with the intent of hiding information.
Keys	A secret code that's used to lock or unlock encrypted data.

Terms	Definition
Signature	An encrypted stamp of authentication on data.
Signing	The process of verifying data's authenticity by using a signature.
X.509	A standard that defines the format of public key certificates.

Key design strategies

Organizational mandates or regulatory requirements might enforce encryption mechanisms. For example, there might be a requirement that data must remain only in the selected region, and copies of the data are maintained in that region.

These requirements are often the base minimum. Strive for a higher level of protection. You're responsible for **preventing confidentiality leaks and tampering of sensitive data**, whether it's external user data or employee data.

Encryption scenarios

Encryption mechanisms likely need to secure the data in three stages:

- **Data at rest** is all information that's kept in storage objects.

  An example of securing data at rest is using BitLocker to encrypt data that's saved to storage on a disk.

- **Data in transit** is information that's transferred between components, locations, or programs.

  An example of securing data in transit is encrypting data with TLS so packets that move over public and private networks are secure.

- **Data in use** is data that's actively being worked on in memory.

  An example of securing data in use is encrypting with confidential computing to protect data as it's processed.

The preceding choices aren't mutually exclusive. They're often used together in the context of the entire solution. One stage might act as a compensating control. For example, you might need to isolate data to prevent tampering when data is read from memory.

Scope of encryption

Classify data by its purpose and sensitivity level to determine what data you need to encrypt. For data that should be encrypted, determine the required level of protection. Do you need end-to-end TLS encryption for all data in transit? For data at rest, which Azure features can meet your requirements? Do you need to double encrypt data at every storage point? How do you implement information protection?

It's important to balance your encryption decisions because there are significant tradeoffs.

> **Tradeoff**: Every encryption hop can introduce performance latency. Operational complexities can occur in relation to troubleshooting and observability. Recovery can be a challenge.
>
> Scope these tradeoffs. Anticipate tradeoffs for data that's classified as sensitive. Requirements might even determine the tradeoffs, for example if a certain type of data must be encrypted and stored within certain thresholds.

There are cases when encryption isn't possible because of technical limitations, investment, or other reasons. Ensure that those reasons are clear, valid, and documented.

Strong encryption mechanisms shouldn't be your only form of defense. Implement data theft prevention processes, proper testing methods, and anomaly detection.

For information about classification, see Recommendations on data classification.

Native encryption mechanisms

Most Azure services provide a base level of encryption. **Explore platform-provided encryption options**.

It's highly recommended that you don't disable platform capabilities to develop your own functionality. Platform encryption features use modern industry standards, are developed by experts, and are highly tested.

For rare occasions, if you need to replace the platform-provided encryption, evaluate the pros and cons and use industry-standard cryptographic algorithms.

Developers should use cryptography APIs that are built into the operating system rather than nonplatform cryptography libraries. For .NET, follow the .NET cryptography model.

Encryption keys

By default, Azure services use Microsoft-managed encryption keys to encrypt and decrypt data. Azure is responsible for key management.

You can opt for **customer-managed keys**. Azure still uses your keys, but you're accountable for key operations. **You have the flexibility to change keys** when you want. Decryption is a compelling reason to use customer-managed keys.

You should **pair strong encryption with strong decryption**. From a security perspective, protecting a decryption key is important because rotation is a common way to control the blast radius if a key is compromised. Monitor access to detect anomalous access and activities.

Store keys separate from encrypted data. This decoupling helps ensure that the compromise of one entity doesn't affect the other. If you use customer-managed keys, store them in a key store. Store highly sensitive data in a managed hardware security module (HSM).

Both stores are protected with identity-based access. This feature enables you to deny access, even to the platform.

Standard encryption algorithms

Use cryptography algorithms that are well-established and follow industry standards instead of creating custom implementations.

Industry standards for algorithms require encryption schemes to have a certain level of entropy. The entropy sources are injected during encryption. Entropy makes the algorithm strong and makes it difficult for an attacker to extract information. **Determine the tolerable thresholds of entropy**. Encryption procedures are processor-intensive. Find the right balance so that you're maximizing the compute cycles that are spent on the encryption, relative to the overall performance targets of the compute request.

>  **Tradeoff**: If you choose an algorithm that's highly complex or injects more than a reasonable amount of entropy, it degrades your system's performance.

Hashes and checksums

Typically, hashing is an error detection technique. You can also use hashing for security because it **detects changes to data that might be caused by tampering**. Hash functions are based on cryptography, but they don't use keys. Hash functions use algorithms to produce checksums. Checksums can compare data to verify the integrity of it.

Applications should use the SHA-2 family of hash algorithms, such as SHA-256, SHA-384, or SHA-512.

Data at rest

Classify and protect information storage objects in accordance with the internal and external compliance requirements. See the following recommendations:

- **Encrypt data by using native options** that are provided for storage services, data stores, and other resources that are used to persist data. Encrypt this data even if you store data in these storage services or resources only temporarily. Also encrypt your backup data to maintain the same level of security as the original source.

  For more information, see Data at rest protection.

- **Use double encryption**. If your business requirements call for higher assurance, you can perform double encryption. Encrypt data in two or more layers by using independent customer-managed keys. Store the data in a managed HSM. To read the data, you need access to both keys. If one key is compromised, the other key still protects the data. This technique aims to increase attacker costs.

  You can also use platform-provided encryption to double encrypt data. Platform-provided encryption protects the storage media at the infrastructure level, and you apply another layer of encryption at the data level. For example, a message broker service has platform-provided encryption via Microsoft-managed keys that protects the message pipe. This method allows you to encrypt the messages with customer-managed keys.

  Use more than one encryption key. Use a key encryption key (KEK) to protect your data encryption key (DEK).

- **Use identity-based access controls to control access to data**. Add network firewalls to provide an extra layer of security that blocks unexpected and unsafe access.

  For more information, see Recommendations for identity and access management.

- **Store keys in a managed HSM** that has least-privilege access control. Separate the data from the keys to the data.

- **Store limited amount of data** so that you only encrypt what's necessary. Your data shouldn't live longer than your encryption cycle. When data is no longer needed, delete the encrypted data without spending decryption cycles.

Data in transit

- **Use secure protocols for client-server communication**. Transport protocols have a built-in layer of security. TLS is the industry standard for exchanging data between client and server endpoints.

  Don't use versions lower than TLS 1.2. Migrate solutions to support TLS 1.2, and use this version by default. All Azure services support TLS 1.2 on public HTTPS endpoints.

  >  **Risk**: Older clients that don't support TLS 1.2 might not work properly if backward compatibility isn't supported.
  >
  > All website communication should use HTTPS, regardless of the sensitivity of the transferred data. During a client-server handshake, negotiate the use of the HTTP Strict Transport Security (HSTS) policy so that HTTPS transport is maintained and doesn't drop to HTTP during communication. This policy protects against man-in-the-middle attacks.
  >
  > Support for HSTS is for newer versions. You might break backward compatibility with older browsers.

  > **Note**
  >
  > You can also encrypt protocols to establish secure connections for databases. For example, Azure SQL Database supports the Tabular Data Stream (TDS) protocol, which integrates a TLS handshake.
  >
  > A cipher suite is a set of algorithms that are used to standardize the handshake between the client and the server. The ciphers ensure that the exchange is encrypted and authenticated. The choice of ciphers depends on the TLS version that the server uses. For some services, such as Azure Application Gateway, you can choose the version of TLS and the cipher suites that you want to support. Implement cipher suites that use the Advanced Encryption Standard (AES) as a symmetric block cipher. AES-128, AES-192, and AES-256 are acceptable.

- **Manage the lifecycle of certificates**. Certificates have a predetermined lifespan. Don't keep long-lived certificates, and don't let them expire on their own. Implement a process that renews certificates at an acceptable frequency. You can automate the process for renewals that occur at short intervals.

> ⓘ Note
>
> If you use **certificate pinning**, familiarize yourself with the agility and certificate management limitations.
>
> Your workflow shouldn't allow invalid certificates to be accepted in the environment. The certificate pinning process should validate certificates and enforce that validation check. You should monitor access logs to ensure that the signing key is used with proper permissions.
>
> If a key is compromised, the certificate must be revoked immediately. A certificate authority (CA) provides a certificate revocation list (CRL) that indicates the certificates that are invalidated before their expiration. Your validation check should account for CRLs.
>
>  **Tradeoff**: The certification validation process can be cumbersome and usually involves a CA. Determine the data that you must encrypt with certificates. For other types of communication, determine if you can implement localized compensating controls to add security.
>
> One way of localizing controls is with mutual TLS (mTLS). It establishes trust in both directions between the client and the server. Both the client and the server have their own certificates, and each certificate is authenticated with their public or private key pair. With mTLS, you're not dependent on the external CA. The tradeoff is the added complexity of managing two certificates.

- **Double encrypt VPN connections if needed**. Perform double encryption to add defense in depth to your VPN tunnel. When you use two VPN servers, you can hide the IP address between the servers, and also hide the IP address between the server and the destination. During this process, data in transit is also encrypted twice.

  >  **Tradeoff**: Compared to single VPN setups, double VPN setups are often more expensive, and connections are often slower.

- **Implement logging and monitoring processes**. Keep track of access sign-in resources that store information about clients, like their source IP, port, and protocol. Use this information to detect anomalies.

Data-in-use

For high security workloads, segmentation, isolation and least-priviledge are recommended design patterns.

In the context of in-use protection, hardware boundaries may require encryption of data while it's in use in the physical CPU and memory to ensure isolation of VMs, host management code and other components.

Encryption and decryption of data must only be done within those isolation boundaries.

More stringent security or regulatory requirements may also require hardware based, cryptographically signed evidence that data is being encrypted whilst in-use, this can be obtained through **attestation**.

Use of modern security and privacy measures is a common requirement for regulatory workloads. **Confidential computing** is one such technology that supports the requirement. Specific services in Azure offer the ability to protect data while it's being computed-upon. For more information, see Azure Facilitation: Azure Confidential Compute.

Consider the end-end lifecycle of data you are protecting data often moves through multiple systems in its lifetime, take care to ensure that all component parts of a solution can provide the required levels of protection, or ensure that your data management strategy provides appropriate segmentation or masking.

Azure facilitation

The following sections describe Azure services and features that you can use to encrypt your data.

Customer-managed keys

Store customer-managed keys in Azure Key Vault or in a Key Vault-managed HSM.

Key Vault treats the keys like any other secret. Azure role-based access controls (RBAC) access the keys via a permission model. This identity-based control must be used with Key Vault access policies.

For more information, see Provide access to Key Vault keys, certificates, and secrets by using RBAC.

Azure Key Vault Premium and Managed-HSM further enhances the offering by including confidential computing capabilites and Secure Key Release which supports a policy to ensure that that a key is only ever released to a workload that can cryptographically prove it is executing inside a Trusted Execution Environment (TEE).

Data-at-rest protection

- **Azure Storage** automatically encrypts your data with block ciphers when the data is persisted to a storage account. For Azure Blob Storage and Azure Queue Storage, Storage also provides client-side encryption via libraries.

  For more information, see Storage encryption.

- **Azure Virtual Machines** has disk files that serve as virtual storage volumes. You can encrypt the virtual disk files so the contents can't be accessed.

  Managed disks can be exported from the portal. Server-side encryption and encryption at host can protect data only after it's exported. However, you should protect data during the export process. You can use Azure Disk Encryption to protect and safeguard your data during the export process.

  Azure offers several encryption options for managed disks. For more information, see Overview of managed disk encryption options.

- **SQL Database** offers a transparent data encryption feature that's used to encrypt a database file at the page level.

Data-in-transit protection

With Key Vault, you can provision, manage, and deploy public and private Secure Sockets Layer (SSL) or TLS certificates. You can use the certificates with Azure and with your internal connected resources.

Data-in-use protection

Specific services in Azure offer the ability to protect data while it's being computed upon within the physical CPU and memory of a host using Azure confidential computing.

- **Confidential Virtual Machines** offer an entire virtual machine running inside a TEE, the memory and executing CPU contents of the virtual machine are encrypted offering a simple 'lift & shift' approach for moving unmodified applications with high security requirements to Azure. Each Azure confidential VM has its own

dedicated virtual Trust Platform Module (TPM). Encryption is performed while the operating system components securely boot.

- **Confidential AKS worker nodes, Confidential Containers on AKS or Confidential Containers on Azure Container Instances (ACI)** offer the ability to to run and manage unmodified containers inside a TEE which enables customers to benefit from in-use protection. Container offerings are built-upon Confidential Virtual Machines and benefit from the same protections.

- **Application Enclave** solutions are specially built applications taking advantage of specific CPU extensions offered by virtual machine SKUs that support Intel Software Guard Extensions (SGX), these offer a very granular Trusted Compute Base (TCB) but require applications to be specifically coded to take advantage of the features.

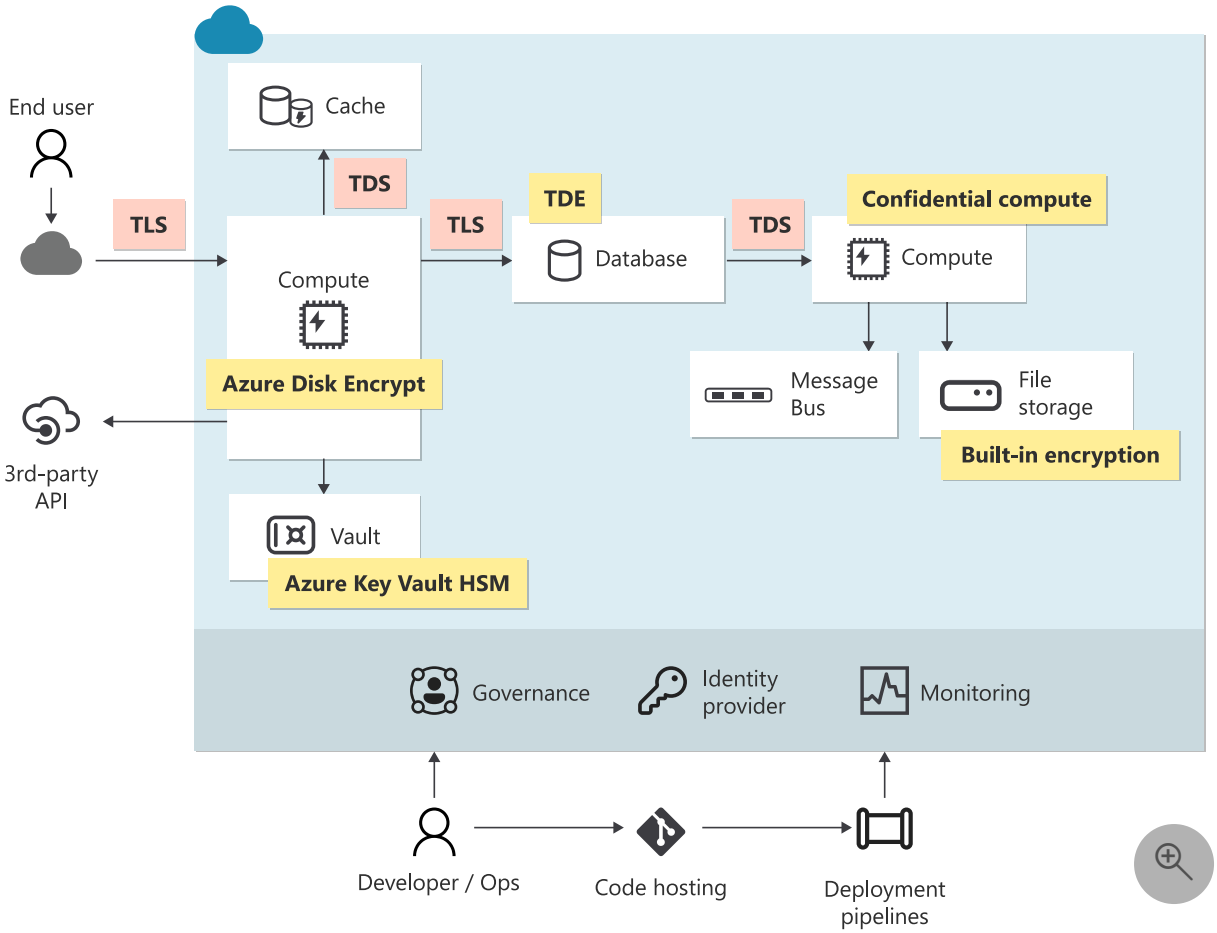
- **Secure Key Release** can be combined with these technologies to ensure that encrypted data is only ever decrypted inside a TEE which proves it provides the required level of protection through a process known as Attestation.

Secret management

You can use Key Vault to securely store and control access to tokens, passwords, certificates, API keys, and other secrets. Use Key Vault as a key and certificate management solution. Premium SKU supports HSMs.

Example

The following example shows encryption solutions that you can use to manage keys, certificates, and secrets.



Related links

- .NET cryptography model
- Azure Disk Encryption
- Storage encryption for data at rest
- Certificate pinning in Azure services
- Provide access to Key Vault keys, certificates, and secrets by using RBAC
- Overview of managed disk encryption options
- Transparent data encryption
- Trust Platform Module overview
- Azure confidential computing

Community links

- Key Vault overview

Security checklist

Refer to the complete set of recommendations.

Security checklist

Recommendations for hardening resources

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Security checklist recommendation:

SE:08	**Harden all workload components by reducing extraneous surface area and tightening configurations to increase attacker cost.**

This guide describes the recommendations for hardening resources by developing localized controls within a workload and maintaining them to withstand repeated attacks.

Security hardening is an intentional self-preservation exercise. The goal is to **reduce an attack surface** and **increase attackers' costs in other areas**, which limits opportunities for malicious actors to exploit vulnerabilities. To protect your workload, implement security best practices and configurations.

Security hardening is an ongoing process that requires continuous monitoring and adaptation to evolving threats and vulnerabilities.

Definitions

Term	Definition
Hardening	The practice of reducing an attack surface area by removing extraneous resources or adjusting configurations.
Privileged access workstation (PAW)	A dedicated and secure machine that you use to perform sensitive tasks, which reduces the risk of compromise.
Secure administrative workstation (SAW)	A specialized PAW that's used by critical impact accounts.
Surface area	A logical footprint of a workload that contains vulnerabilities.

Key design strategies

Security hardening is a highly localized exercise that **strengthens controls at the component level**, whether it's resources or processes. When you tighten the security of each component, it improves the aggregate security assurance of your workload.

Security hardening doesn't consider the functionality of the workload, and it doesn't detect threats or perform automated scanning. **Security hardening focuses on configuration tuning with an assume-breach and defense-in-depth mentality.** The goal is to make it difficult for an attacker to gain control of a system. Hardening shouldn't alter the intended utility of a workload or its operations.

The first step of the hardening process is to gather a comprehensive inventory of all hardware, software, and data assets. Keep your inventory records up to date by adding new assets and removing decommissioned assets. For all assets in your inventory, consider the following best practices:

- **Reduce the footprint**. Remove extraneous surface area or reduce the scope. **Eliminate easy targets**, or cheap and well-established attack vectors, such as unpatched software exploits and brute force attacks. Prior to the production deployment, you should clean identities, build components, and other nonrequired assets from the source tree.

- **Fine-tune configurations**. Evaluate and **tighten the remaining surface area**. When resources are hardened, tried and tested methods that attackers use are no longer successful. It forces attackers to acquire and use advanced or untested attack methods, which increases their costs.

- **Maintain defenses**. Maintain protective measures by performing **continuous threat detection** to help ensure that hardening efforts are dependable over time.

Also consider the following factors.

Trusted source. Part of the hardening exercise involves the software supply chain. This guidance assumes that **all components are obtained from trusted sources**. Your organization must approve software that's procured from third-party vendors. This approval applies to sources of the operating system, images, and other third-party tools. Without trusted resources, hardening can be an infinite drain of security assurances on untrusted sources.

For recommendations about security for your supply chain, see Recommendations for securing a development lifecycle.

Training. Hardening is a specialized skill. It's methodical and requires a high level of competency. You need to understand the functionality of a component and how changes affect the component. A team member must be able to discern the guidance that's from industry experts and the platform to distinguish it from guidance from uncertain sources. Educate your team members in creating a security-aware culture.

Ensure that your team is **proficient in security best practices, has awareness of potential threats, and learns from post-incident retrospectives**.

Documentation. Document and publish hardening requirements, decisions, and defined methods. For transparency, also **document exceptions or deviations** from those requirements.

Hardening can be cumbersome, but it's a crucial security exercise that you must document. Harden the core components first, and then expand to other areas, such as automated processes and human processes, to tighten up potential gaps. Be meticulous about changes. For example, a necessary step is to disable the default settings because changes to default values can't affect the stability of the system. Even if the replacement configuration is the same as the default, it must be defined. The following sections describe common targets for hardening. Evaluate key design areas of your workload and follow the key strategies to harden at a component level.

Networking

Divide the network into segments to isolate critical assets and sensitive data from less secure assets, which reduces lateral movements by attackers. In those segments, apply a *deny-by-default* approach. Only add access to the allowlist if it's justified.

Disable ports and protocols that aren't actively used. For example, on Azure App Service, if you don't need to deploy via FTP, you can disable it. Or if you perform management operations via an internal network, you can disable administrative access from the internet.

Remove or disable legacy protocols. Attackers exploit systems that use old versions. Use an Azure detection service to review logs and determine protocol usage. It might be difficult to remove protocols because it can disrupt the functionality of the system. Test all changes before implementation to mitigate the risk of operational interruption.

Treat public IP (PIP) addresses as high-risk assets because they're easy to access and have a broad worldwide reach. To reduce exposure, remove unnecessary internet access to the workload. Use shared public IP addresses that Microsoft services, like Azure Front Door, provide. These services are designed to be internet-facing, and they block access to disallowed protocols. Many such services perform initial checks on incoming requests at the network edge. With a dedicated PIP, you're responsible for managing its security aspects, allowing or blocking ports, and scanning incoming requests to ensure their validity.

For internet-facing applications, **restrict access by adding a layer-7** service that can filter invalid traffic. Explore native services that enforce distributed denial-of-service (DDoS)

protection, have web application firewalls, and provide protection at the edge before traffic reaches the application tier.

Domain Name System (DNS) hardening is another network security practice. To ensure that the DNS infrastructure is secure, we recommend that you **use trusted DNS resolvers**. To validate information from DNS resolvers and provide an extra layer of security, when possible, use a DNS security protocol for highly sensitive DNS zones. To prevent attacks such as DNS cache poisoning, DDoS attacks, and amplification attacks, explore other DNS-related security controls such as query rate limiting, response rate limiting, and DNS cookies.

Identity

Remove unused or default accounts. Disable unused authentication and authorization methods.

Disable legacy authentication methods because they're frequently attack vectors. Old protocols often lack attack-counter measures, such as account lockouts. Externalize your authentication requirements to your identity provider (IdP), such as Microsoft Entra ID.

Prefer federation over creating duplicate identities. If an identity is compromised, it's easier to revoke its access when it's centrally managed.

Understand platform capabilities for enhanced authentication and authorization. Harden access controls by taking advantage of multifactor authentication, passwordless authentication, Conditional Access, and other features that Microsoft Entra ID offers to verify identity. You can add extra protection around sign-in events and reduce the scope in which an attacker can make a request.

Use managed identities and workload identities with no credentials where possible. Credentials can be leaked. For more information, see Recommendations for protecting application secrets.

Use the least-privilege approach for your management processes. Remove unnecessary role assignments and perform regular Microsoft Entra access reviews. Use role assignment descriptions to keep a paper trail of justifications, which is crucial for audits.

Cloud resources

The preceding hardening recommendations for networking and identity apply to individual cloud services. For networking, pay special attention to **service-level firewalls**, and evaluate their inbound rules.

Discover and disable unused capabilities or features, such as unused data plane access and product features, that other components might cover. For example, App Service supports Kudu, which provides FTP deployments, remote debugging, and other features. If you don't need those features, turn them off.

Always **keep up with the Azure roadmap and the workload roadmap**. Apply patching and versioning updates that Azure services offer. Allow platform-provided updates, and subscribe to automated update channels.

>  **Risk**: Cloud resources often have requirements for allowances or must run in documented configurations to be considered *supported*. Some hardening techniques, such as aggressively blocking outbound traffic, can cause a service to fall outside a supported configuration, even if the service operates normally. Understand each cloud resource's runtime requirements from your platform to ensure that you maintain support for that resource.

Applications

Evaluate areas where your application might inadvertently leak information. For example, suppose you have an API that retrieves user information. A request might have a valid user ID, and your application returns a 403 error. But with an invalid customer ID, the request returns a 404 error. Then you're effectively leaking information about your user IDs.

There might be more subtle cases. For example, the response latency with a valid user ID is higher than an invalid customer ID.

Consider implementing application hardening in the following areas:

- **Input validation and sanitization**: Prevent injection attacks such as SQL injection and cross-site scripting (XSS) by validating and sanitizing all user inputs. Automate input sanitization by using input validation libraries and frameworks.
- **Session management**: Protect session identifiers and tokens from theft or session fixation attacks by using secure session management techniques. Implement session timeouts, and enforce reauthentication for sensitive actions.
- **Error management**: Implement custom error handling to minimize exposing sensitive information to attackers. Securely log errors and monitor these logs for suspicious activity.
- **HTTP security headers**: Mitigate common web vulnerabilities by utilizing security headers in HTTP responses, such as the Content Security Policy (CSP), X-Content-

Type-Options, and X-Frame-Options.

- **API security**: Secure your APIs with proper authentication and authorization mechanisms. To further enhance security, implement rate limiting, request validation, and access controls for API endpoints.

Follow secure coding practices when you develop and maintain applications. Regularly conduct code reviews and scan applications for vulnerabilities. For more information, see Recommendations for securing a development lifecycle.

Management operations

Also harden other non-runtime resources. For example, **reduce your build operations footprint** by taking an inventory of all assets and removing unused assets from your pipeline. Then, **pull in tasks that are published by trusted sources**, and only run tasks that are validated.

Determine if you need Microsoft-hosted or self-hosted build agents. **Self-hosted build agents need extra management and must be hardened**.

From an observability perspective, **implement a process for reviewing logs** for potential breaches. Regularly review and update access control rules based on access logs. Work with central teams to analyze security information event management (SIEM) and security orchestration automated response (SOAR) logs to detect anomalies.

Consider requiring PAWs or SAWs for privileged management operations. PAWs and SAWs are hardened physical devices that offer significant security advantages, but their implementation requires careful planning and management. For more information, see Securing devices as part of the privileged access story.

Azure facilitation

Microsoft Defender for Cloud offers several hardening capabilities:

- Server hardening
- Adaptive network hardening
- Docker host hardening

The Center for Internet Security (CIS) offers hardened images in Azure Marketplace.

You can use Azure VM Image Builder to build a repeatable process for hardened OS images. Common Base Linux-Mariner is a hardened Linux distribution that's developed

by Microsoft that follows security standards and industry certifications. You can use it with Azure infrastructure products to build workload implementations.

Example

The following procedure is an example of how to harden an operating system:

1. **Reduce the footprint**. Remove unnecessary components in an image. Install only what you need.

2. **Fine-tune configurations**. Disable unused accounts. The default configuration of operating systems has extra accounts that are linked to security groups. If you don't use those accounts, disable or remove them from the system. Extra identities are threat vectors that can be used to gain access to the server.

   Disable unnecessary access to the file system. Encrypt the file system and fine-tune access controls for identity and networking.

   Run only what's needed. Block applications and services that run by default. Approve only applications and services that are needed for workload functionality.

3. **Maintain defenses**. Regularly update operating system components with the latest security updates and patches to mitigate known vulnerabilities.

Organizational alignment

Cloud Adoption Framework for Azure provides guidance about creating centralized identity and access management functions. For more information, see Azure identity and access management design area.

Related links

- Adaptive network hardening
- Recommendations for protecting application secrets
- Recommendations for securing a development lifecycle
- Securing devices as part of the privileged access story
- Server hardening

Community links

CIS benchmarks

Security checklist

Refer to the complete set of recommendations.

Security checklist

Recommendations for protecting application secrets

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Security checklist recommendation:

> **SE:09** **Protect application secrets by hardening their storage and restricting access and manipulation and by auditing those actions. Run a reliable and regular rotation process that can improvise rotations for emergencies.**

This guide describes the recommendations for securing sensitive information in applications. Proper management of secrets is crucial for maintaining the security and integrity of your application, workload, and associated data. Improper handling of secrets can lead to data breaches, service disruption, regulatory violations, and other issues.

Credentials, such as API keys, Open Authorization (OAuth) tokens, and Secure Shell (SSH) keys are secrets. Some credentials, such as client-side OAuth tokens, can be dynamically created at runtime. Dynamic secrets still need to be safeguarded despite their temporary nature. Noncredential information, like certificates and digital signature keys, can also be sensitive. Compliance requirements might cause configuration settings that aren't typically considered secret to be treated as application secrets.

Definitions

Term	Definition
Certificates	Digital files that hold the public keys for encryption or decryption.
Credentials	Information that's used to verify the identity of the publisher or consumer in a communication channel.
Credential scanning	The process of validating source code to make sure secrets aren't included.
Encryption	The process by which data is made unreadable and locked with a secret code.
Key	A secret code that's used to lock or unlock encrypted data.
Least-privilege access	A Zero Trust principle that aims at minimizing a set of permissions to complete a job function.
Managed identity	An identity that's assigned to resources and managed by Azure.

Term	Definition
Nonsecret	Information that doesn't jeopardize the security posture of the workload if it's leaked.
Rotation	The process of regularly updating secrets so that, if they're compromised, they're available only for a limited time.
Secret	A confidential component of the system that facilitates communication between workload components. If leaked, secrets can cause a breach.
X.509	A standard that defines the format of public key certificates.

> ⓘ **Important**
>
> **Don't treat nonsecrets like secrets.** Secrets require operational rigor that's unnecessary for nonsecrets and that might result in extra costs.
>
> Application configuration settings, such as URLs for APIs that the application uses, are an example of nonsecrets. This information shouldn't be stored with the application code or application secrets. Consider using a dedicated configuration management system such as Azure App Configuration to manage these settings. For more information, see **What is Azure App Configuration?**.

Key design strategies

Your secret management strategy should minimize secrets as much as possible and integrate them into the environment by taking advantage of platform features. For example, if you use a managed identity for your application, access information isn't embedded in connection strings and it's safe to store the information in a configuration file. Consider the following areas of concern before storing and managing secrets:

- Created secrets should be kept in secure storage with strict access controls.
- Secret rotation is a proactive operation, whereas revocation is reactive.
- Only trusted identities should have access to secrets.
- You should maintain an audit trail to inspect and validate access to secrets.

Build a strategy around these points to help prevent identity theft, avoid repudiation, and minimize unnecessary exposure to information.

Safe practices for secret management

If possible, avoid creating secrets. Find ways to **delegate responsibility to the platform**. For example, use the platform's built-in managed identities to handle credentials. Fewer secrets result in reduced surface area and less time spent on secret management.

We recommend that keys have three distinct roles: user, administrator, and auditor. Role distinction helps to ensure that only trusted identities have access to secrets with the appropriate level of permission. Educate developers, administrators, and other relevant personnel about the importance of secret management and security best practices.

Preshared keys

You can control access by creating distinct keys for each consumer. For example, a client communicates with a third-party API using a preshared key. If another client needs to access the same API, they must use another key. Don't share keys even if two consumers have the same access patterns or roles. Consumer scopes might change over time, and you can't independently update permissions or distinguish usage patterns after a key is shared. Distinct access also makes revocation easier. If a consumer's key is compromised, it's easier to revoke or rotate that key without affecting other consumers.

This guidance applies to different environments. The same key shouldn't be used for both preproduction and production environments. If you're responsible for creating preshared keys, make sure you create multiple keys to support multiple clients.

For more information, see Recommendations for identity and access management.

Secret storage

Use a secret management system, like Azure Key Vault, to store secrets in a hardened environment, encrypt at-rest and in-transit, and audit access and changes to secrets. If you need to store application secrets, keep them outside the source code for easy rotation.

Certificates should only be stored in Key Vault or in the OS's certificate store. For example, storing an X.509 certificate in a PFX file or on a disk isn't recommended. If you need a higher level of security, choose systems that have hardware security module (HSM) capabilities instead of software-based secret stores.

> **Tradeoff**: HSM solutions are offered at a higher cost. You might also see an effect on application performance due to added layers of security.

A dedicated secret management system makes it easy to store, distribute, and control access to application secrets. Only authorized identities and services should have access to secret stores. Access to the system can be restricted via permissions. Always apply the least-privilege approach when assigning permissions.

You also need to control access at the secret level. Each secret should only have access to a single resource scope. Create isolation boundaries so that a component is only able to use secrets that it needs. If an isolated component is compromised, it can't gain control of other secrets and potentially the entire workload. One way to isolate secrets is to use multiple key vaults. There's no added costs for creating extra key vaults.

Implement auditing and monitoring for secret access. Log who accesses secrets and when to identify unauthorized or suspicious activity. For information about logging from a security perspective, see Recommendations on security monitoring and threat detection.

Secret rotation

Have a process in place that maintains secret hygiene. The longevity of a secret influences the management of that secret. To reduce attack vectors, secrets should be retired and replaced with new secrets as frequently as possible.

Handle OAuth access tokens carefully, taking into consideration their time to live. Consider if the exposure window needs to be adjusted to a shorter period. Refresh tokens must be stored securely with limited exposure to the application. Renewed certificates should also use a new key. For information about refresh tokens, see Secure OAuth 2.0 On-Behalf-Of refresh tokens.

Replace secrets after they reach their end of life, are no longer used by the workload, or if they've been compromised. Conversely, don't retire active secrets unless it's an emergency. You can determine a secret's status by viewing access logs. Secret rotation processes shouldn't affect the reliability or performance of the workload. Use strategies that build redundancy in secrets, consumers, and access methods for smooth rotation.

For more information on how Azure Storage handles rotation, see Manage account access keys.

Rotation processes should be automated and deployed without any human interaction. Storing secrets in a secret management store that natively supports rotation concepts can simplify this operational task.

Safe practices for using secrets

As a secret generator or operator, you should be able to distribute secrets in a safe manner. Many organizations use tools to securely share secrets both within the organization and externally to partners. In absence of a tool, have a process for properly handing off credentials to authorized recipients. Your disaster recovery plans should include secret recovery procedures. Have a process for situations where a key is compromised or leaked and needs to be regenerated on demand. Consider the following best practices for safety when using secrets:

Prevent hardcoding

Don't hard code secrets as static text in code artifacts such as application code, configuration files, and build-deployment pipelines. This high-risk practice makes the code vulnerable because secrets are exposed to everyone with read access.

You can avoid this situation by using managed identities to eliminate the need to store credentials. Your application uses its assigned identity to authenticate against other resources via the identity provider (IdP). Test in nonproduction environments with fake secrets during development to prevent accidental exposure of real secrets.

Use tools that periodically detect exposed secrets in your application code and build artifacts. You can add these tools as Git precommit hooks that scan for credentials before source code commits deploy. Review and sanitize application logs regularly to help ensure that no secrets are inadvertently recorded. You can also reinforce detection via peer reviews.

> ⓘ **Note**
>
> If the scanning tools discover a secret, that secret must be considered compromised. It should be revoked.

Respond to secret rotation

As a workload owner, you need to **understand the secret rotation plan and policies so that you can incorporate new secrets with minimal disruption to users.** When a secret is rotated, there might be a window when the old secret isn't valid, but the new secret hasn't been placed. During that window, the component that the application is trying to reach doesn't acknowledge requests. You can minimize these issues by building retry logic into the code. You can also use concurrent access patterns that allow you to have multiple credentials that can be safely changed without affecting each other.

Work with the operations team and be part of the change management process. You should let credential owners know when you decommission a part of the application that uses credentials that are no longer needed.

Integrate secret retrieval and configuration into your automated deployment pipeline. Secret retrieval helps to ensure secrets are automatically fetched during deployment. You can also use secret injection patterns to insert secrets into application code or configuration at runtime, which prevents secrets from being accidentally exposed to logs or version control.

Azure facilitation

Store secrets by using Key Vault. Store secrets in the Azure secret management system, Key Vault, Azure Managed HSM, and other locations. For more information, see How to choose the right key management solution.

Integrate identity-based access control. Microsoft Entra ID and managed identities help minimize the need for secrets. Microsoft Entra ID offers a highly secure and usable experience for access control with built-in mechanisms for handling key rotation, for anomalies, and more.

Use Azure role-based access control (RBAC) to assign permissions to users, groups, and applications at a certain scope.

Use an access model to control key vaults, permissions, and secrets. For more information, see Access model overview.

Implement secret exposure detection. Integrate processes in your workload that detect suspicious activity and periodically check for exposed keys in your application code. Some options include:

- Azure DevOps Credential Scanner task
- Defender for Cloud secret scanning
- Microsoft Defender for Key Vault
- GitHub Secret Scanner

Don't store keys and secrets for any environment type in application configuration files or continuous integration and continuous delivery (CI/CD) pipelines. Developers should use Visual Studio Connected Services or local-only files to access credentials.

Related links

- Access model overview
- Azure DevOps Credential Scanner task
- Configure the Microsoft Security DevOps Azure DevOps extension
- Configure GitHub Advanced Security for Azure DevOps
- Defender for Cloud secret scanning
- How to choose the right key management solution
- Manage account access keys
- Microsoft Defender for Key Vault
- Recommendations on security monitoring and threat detection
- Recommendations for identity and access management
- Secure OAuth 2.0 On-Behalf-Of refresh tokens for web services
- Visual Studio Connected Services

Community links

- GitHub secret scanner

Security checklist

Refer to the complete set of recommendations.

Security checklist

Recommendations for monitoring and threat detection

Article • 02/13/2024

Applies to this Azure Well-Architected Framework Security checklist recommendation:

Expand table

SE:10	**Implement a holistic monitoring strategy that relies on modern threat detection mechanisms that can be integrated with the platform. Mechanisms should reliably alert for triage and send signals into existing SecOps processes.**

This guide describes the recommendations for monitoring and threat detection. Monitoring is fundamentally a process of **getting information about events that have already occurred**. Security monitoring is a practice of capturing information at different altitudes of the workload (infrastructure, application, operations) to **gain awareness of suspicious activities**. The goal is to predict incidents and learn from past events. Monitoring data provides the basis of post-incident analysis of what occurred to help incident response and forensic investigations.

Monitoring is an Operational Excellence approach that's applied across all Well-Architected Framework pillars. This guide provides recommendations only from a security perspective. General concepts of monitoring, like code instrumentation, data collection, and analysis, are out of scope for this guide. For information about core monitoring concepts, see Recommendations for designing and building an observability framework.

Definitions

Expand table

Term	Definition
Audit logs	A record of activities in a system.
Security information and event management (SIEM)	An approach that uses built-in threat detection and intelligence capabilities based on data that's aggregated from multiple sources.
Threat detection	A strategy for detecting deviations from expected actions by using collected, analyzed, and correlated data.

Term	Definition
Threat intelligence	A strategy for interpreting threat detection data to detect suspicious activity or threats by examining patterns.
Threat prevention	Security controls that are placed in a workload at various altitudes to protect its assets.

Key design strategies

The main purpose of security monitoring is **threat detection**. The primary objective is to prevent potential security breaches and maintain a secure environment. However, it's equally important to recognize that not all threats can be preemptively blocked. In such instances, monitoring also serves as a mechanism to identify the cause of a security incident that has occurred despite the prevention efforts.

Monitoring can be approached from various perspectives:

- **Monitor at various altitudes.** Observing from **various altitudes** is the process of getting information about user flows, data access, identity, networking, and even the operating system. Each of these areas offers unique insights that can help you identify deviations from expected behaviors that are established against the security baseline. Conversely, continuously monitoring a system and applications over time can **help establish that baseline posture**. For example, you might typically see around 1,000 sign-in attempts in your identity system every hour. If your monitoring detects a spike of 50,000 sign-in attempts during a short period, an attacker might be trying to gain access to your system.

- **Monitor at various scopes of impact.** It's critical to **observe the application and the platform**. Assume an application user accidentally gets escalated privileges or a security breach occurs. If the user performs actions beyond their designated scope, the impact might be confined to actions that other users can perform.

  However, if an internal entity compromises a database, the extent of the potential damage is uncertain.

  If a compromise occurs on the Azure resource side, the impact could be global, affecting all entities that interact with the resource.

  The blast radius or impact scope could be significantly different, depending on which of these scenarios occurs.

- **Use specialized monitoring tools.** It's critical to invest in **specialized tools** that can continuously scan for anomalous behavior that might indicate an attack. Most of

these tools have **threat intelligence capabilities** that can perform predictive analysis based on a large volume of data and known threats. Most tools aren't stateless and incorporate a deep understanding of telemetry in a security context.

The tools need to be platform-integrated or at least platform-aware to get deep signals from the platform and make predictions with high fidelity. They must be able to generate alerts in a timely manner with enough information to conduct proper triage. Using too many diverse tools can lead to complexity.

- **Use monitoring for incident response.** Aggregated data, transformed into actionable intelligence, **enables swift and effective reactions** to incidents. Monitoring **helps with post-incident activities**. The goal is to collect enough data to analyze and understand what happened. The process of monitoring captures information on past events to enhance reactive capabilities and potentially predict future incidents.

The following sections provide recommended practices that incorporate the preceding monitoring perspectives.

Capture data to keep a trail of activities

The objective is to maintain a **comprehensive audit trail** of events that are significant from a security perspective. Logging is the most common way to capture access patterns. Logging must be performed for the application and the platform.

For an audit trail, you need to **establish the *what*, *when*, and *who* that's associated with actions**. You need to identify the specific timeframes when actions are performed. Make this assessment in your threat modeling. To counteract a repudiation threat, you should establish strong logging and auditing systems that result in a record of activities and transactions.

The following sections describe use cases for some common altitudes of a workload.

Application user flows

Your application should be designed to provide runtime visibility when events occur. **Identify critical points within your application and establish logging for these points.** For example, when a user logs into the application, capture the user's identity, source location, and other relevant information. It's important to acknowledge any escalation in user privileges, the actions performed by the user, and whether the user accessed sensitive information in a secure data store. Keep track of activities for the user and the user session.

To facilitate this tracking, code should be **instrumented via structured logging**. Doing so enables easy and uniform querying and filtering of the logs.

> **ⓘ Important**
>
> You need to enforce responsible logging to maintain the confidentiality and integrity of your system. Secrets and sensitive data must not appear in logs. Be aware of leaking personal data and other compliance requirements when you capture this log data.

Identity and access monitoring

Maintain a thorough **record of access patterns for the application and modifications to platform resources**. Have robust activity logs and threat detection mechanisms, particularly for identity-related activities, because attackers often attempt to manipulate identities to gain unauthorized access.

Implement comprehensive logging by **using all available data points**. For example, include the client IP address to differentiate between regular user activity and potential threats from unexpected locations. All logging events should be timestamped by the server.

Record all resource access activities, capturing who's doing what and when they're doing it. Instances of privilege escalation are a significant data point that should be logged. Actions related to account creation or deletion by the application must also be recorded. This recommendation extends to application secrets. Monitor who accesses secrets and when they're rotated.

Although logging successful actions is important, **recording failures is necessary from a security perspective**. Document any violations, like a user attempting an action but encountering an authorization failure, access attempts for nonexistent resources, and other actions that seem suspicious.

Network monitoring

By monitoring network packets and their sources, destinations, and structures, you gain visibility into access patterns at the network level.

Your segmentation design should **enable observation points at the boundaries** to monitor what crosses them and log that data. For example, monitor subnets that have

network security groups that generate flow logs. Also monitor firewall logs that show the flows that were allowed or denied.

There are access logs for inbound connection requests. These logs record the source IP addresses that initiate the requests, the type of request (GET, POST), and all other information that's part of the requests.

Capturing DNS flows is a significant requirement for many organizations. For instance, **DNS logs can help identify which user or device initiated a particular DNS query**. By correlating DNS activity with user/device authentication logs, you can track activities to individual clients. This responsibility often extends to the workload team, especially if they deploy anything that makes DNS requests part of their operation. DNS traffic analysis is a key aspect of platform security observability.

It's important to monitor unexpected DNS requests or DNS requests that are directed toward known command and control endpoints.

> **Tradeoff**: **Logging all network activities can result in a large amount of data.** Every request from layer 3 can be recorded in a flow log, including every transaction that crosses a subnet boundary. Unfortunately, it's not possible to capture only adverse events because they can only be identified after they occur. Make strategic decisions about the type of events to capture and how long to store them. If you're not careful, managing the data can be overwhelming. There's also a tradeoff on the cost of storing that data.

Because of the tradeoffs, you should consider whether the benefit of network monitoring of your workload is sufficient to justify the costs. If you have a web application solution with a high request volume and your system makes extensive use of managed Azure resources, the cost might outweigh the benefits. On the other hand, if you have a solution that's designed to use virtual machines with various ports and applications, it might be important to capture and analyze network logs.

Capture system changes

To maintain the integrity of your system, you should have an accurate and up-to-date record of system state. If there are changes, you can use this record to promptly address any issues that arise.

Build processes should also emit telemetry. Understanding the security context of events is key. Knowing what triggered the build process, who triggered it, and when it was triggered can provide valuable insights.

Track **when resources are created and when they're decommissioned**. This information must be extracted from the platform. This information provides valuable insights for resource management and accountability.

Monitor **drift in resource configuration**. Document any change to an existing resource. Also keep track of changes that don't complete as part of a rollout to a fleet of resources. Logs must capture the specifics of the change and the exact time it occurred.

Have a comprehensive view, from a patching perspective, of whether the system is up-to-date and secure. **Monitor routine update processes** to verify that they complete as planned. A security patching process that doesn't complete should be considered a vulnerability. You should also maintain an inventory that records the patch levels and any other required details.

Change detection also applies to the operating system. This involves tracking whether services are added or turned off. It also includes monitoring for the addition of new users to the system. There are tools that are designed to target an operating system. They help with context-less monitoring in the sense that they don't target the functionality of the workload. For example, file integrity monitoring is a critical tool that enables you to track changes in system files.

You should set up alerts for these changes, particularly if you don't expect them to occur often.

> **ⓘ Important**
>
> When you roll out to production, be sure that alerts are configured to catch anomalous activity that's detected on the application resources and build process.

In your test plans, **include the validation of logging and alerting** as prioritized test cases.

Store, aggregate, and analyze data

Data collected from these monitoring activities must be stored in data sinks where it can be thoroughly **examined, normalized, and correlated**. Security data should be persisted outside the system's own data stores. Monitoring sinks, whether they're localized or central, must outlive the data sources. The **sinks can't be ephemeral** because sinks are the source for intrusion detection systems.

Networking logs can be verbose and take up storage. **Explore different tiers in storage systems**. Logs can naturally transition to colder storage over time. This approach is

beneficial because older flow logs typically aren't used actively and are only needed on demand. This method ensures efficient storage management while also ensuring that you can access historical data when you need to.

The flows of your workload are typically a composite of multiple logging sources. Monitoring data must be **analyzed intelligently across all those sources**. For example, your firewall will only block traffic that reaches it. If you have a network security group that has already blocked certain traffic, that traffic isn't visible to the firewall. To reconstruct the sequence of events, you need to aggregate data from all components that are in flow and then aggregate data from all flows. This data is particularly useful in a post-incident response scenario when you're trying to understand what happened. Accurate timekeeping is essential. For security purposes, all systems need to use a network time source so that they're always in sync.

Centralized threat detection with correlated logs

You can use a system like security information and event management (SIEM) to **consolidate security data in a central location** where it can be correlated across various services. These systems have **built-in threat detection** mechanisms. They can **connect to external feeds** to obtain threat intelligence data. Microsoft, for example, publishes threat intelligence data that you can use. You can also buy threat intelligence feeds from other providers, like Anomali and FireEye. These feeds can provide valuable insights and enhance your security posture. For threat insights from Microsoft, see Security Insider.

A SIEM system can **generate alerts** based on correlated and normalized data. These alerts are a significant resource during an incident response process.

> **Tradeoff**: SIEM systems can be expensive, complex, and require specialized skills. However, if you don't have one, you might need to correlate data on your own. This can be a time-consuming and complex process.

SIEM systems are usually managed by an organization's central teams. If your organization doesn't have one, consider advocating for it. It could alleviate the burden of manual log analysis and correlation to allow more efficient and effective security management.

Some cost-effective options are provided by Microsoft. Many Microsoft Defender products provide the alerting functionality of a SIEM system, but without a data-aggregation feature.

By combining several smaller tools, you can emulate some functions of a SIEM system. However, you need to know that these makeshift solutions might not be able to

perform correlation analysis. These alternatives can be useful, but they might not fully replace the functionality of a dedicated SIEM system.

Detect abuse

Be proactive about threat detection and be vigilant for signs of abuse, like identity brute force attacks on an SSH component or an RDP endpoint. Although external threats might generate a lot of noise, especially if the application is exposed to the internet, **internal threats are often a greater concern**. An unexpected brute force attack from a trusted network source or an inadvertent misconfiguration, for instance, should be investigated immediately.

Keep up with your hardening practices. Monitoring isn't a substitute for proactively hardening your environment. A larger surface area is prone to more attacks. Tighten controls as much as practice. Detect and disable unused accounts, remove unused ports, and use a web application firewall, for example. For more information about hardening techniques, see Recommendations on security hardening.

Signature-based detection can inspect a system in detail. It involves looking for signs or correlations between activities that might indicate a potential attack. A detection mechanism might identify certain characteristics that are indicative of a specific type of attack. It might not always be possible to directly detect the command-and-control mechanism of an attack. However, there are often hints or patterns associated with a particular command-and-control process. For example, an attack might be indicated by a certain flow rate from a request perspective, or it might frequently access domains that have specific endings.

Detect **anomalous user access patterns** so that you can identify and investigate deviations from expected patterns. This involves comparing current user behavior with past behavior to spot anomalies. Although it might not be feasible to perform this task manually, you can use threat intelligence tools to do it. Invest in **User and Entity Behavior Analytics (UEBA) tools** that collect user behavior from monitoring data and analyze it. These tools can often perform predictive analysis that maps suspicious behaviors to potential types of attack.

Detect threats during pre-deployment and post-deployment stages. During the predeployment phase, incorporate vulnerability scanning into pipelines and take necessary actions based on the results. Post-deployment, continue to conduct vulnerability scanning. You can use tools like Microsoft Defender for Containers, which scans container images. Include the results in the collected data. For information about secure development practices, see Recommendations for using safe deployment practices.

Take advantage of platform-provided detection mechanisms and measures. For example, Azure Firewall can analyze traffic and block connections to untrusted destinations. Azure also provides ways to detect and protect against distributed denial-of-service (DDoS) attacks.

Azure facilitation

Azure Monitor provides observability across your entire environment. With no configuration, you automatically get platform metrics, activity logs, and diagnostics logs from most of your Azure resources. The activity logs provide detailed diagnostic and auditing information.

> **Note**
>
> Platform logs aren't available indefinitely. You need to keep them so that you can review them later for auditing purposes or offline analysis. Use Azure storage accounts for long-term/archival storage. In Azure Monitor, specify a retention period when you enable diagnostic settings for your resources.

Set up alerts based on predefined or custom metrics and logs to get notifications when specific security-related events or anomalies are detected.

For more information, see Azure Monitor documentation.

Microsoft Defender for Cloud provides built-in capabilities for threat detection. It operates on collected data and analyzes logs. Because it's aware of the types of logs generated, it can use built-in rules to make informed decisions. For example, it checks lists of potentially compromised IP addresses and generates alerts.

Enable built-in threat protection services for Azure resources. For example, enable Microsoft Defender for Azure resources, like virtual machines, databases, and containers, to detect and protect against known threats.

Defender for Cloud provides cloud workload protection platform (CWPP) capabilities for threat detection of all workload resources.

For more information, see What is Microsoft Defender for Cloud?.

Alerts generated by Defender can also feed into SIEM systems. **Microsoft Sentinel** is the native offering. It uses AI and machine learning to detect and respond to security threats in real time. It provides a centralized view of security data and facilitates proactive threat hunting and investigation.

For more information, see What is Microsoft Sentinel?.

Microsoft Sentinel can also use threat intelligence feeds from various sources. For more information, see Threat intelligence integration in Microsoft Sentinel.

Microsoft Sentinel can analyze user behavior from monitoring data. For more information, see Identify advanced threats with User and Entity Behavior Analytics (UEBA) in Microsoft Sentinel.

Defender and Microsoft Sentinel work together, despite some overlap in functionality. This collaboration enhances your overall security posture by helping to ensure comprehensive threat detection and response.

Take advantage of **Azure Business Continuity Center** to identify gaps in your business continuity estate and defend against threats like ransomware attacks, malicious activities, and rogue-administrator incidents. For more information, see What is Azure Business Continuity Center?.

Networking

Review all logs, including raw traffic, from your network devices.

- Security group logs. Review flow logs and diagnostic logs.
- Azure Network Watcher. Take advantage of the packet capture feature to set alerts and gain access to real-time performance information at the packet level.

  Packet capture tracks traffic in and out of virtual machines. You can use it to run proactive captures based on defined network anomalies, including information about network intrusions.

  For an example, see Monitor networks proactively with alerts and Azure Functions using Packet Capture.

Identity

Monitor identity-related risk events on potentially compromised identities and remediate those risks. Review the reported risk events in these ways:

- Use Microsoft Entra ID reporting. For more information, see What is Identity Protection? and Identity Protection.
- Use Identity Protection risk detection API members to get programmatic access to security detections via Microsoft Graph. For more information, see riskDetection

and riskyUser.

Microsoft Entra ID uses adaptive machine learning algorithms, heuristics, and known compromised credentials (user name and password pairs) to detect suspicious actions that are related to your user accounts. These user name and password pairs are surfaced by monitoring the public and dark web and by working with security researchers, law enforcement, security teams at Microsoft, and others.

Azure Pipelines

DevOps advocates change management of workloads via continuous integration and continuous delivery (CI/CD). Be sure to add security validation in the pipelines. Follow the guidance described in Securing Azure Pipelines.

Related links

- Recommendations for designing and creating an observability framework
- Security Insider
- Recommendations for hardening resources
- Recommendations for using safe deployment practices
- Azure Monitor documentation
- What is Microsoft Defender for Cloud?
- What is Microsoft Sentinel?
- Threat intelligence integration in Microsoft Sentinel
- Identify advanced threats with User and Entity Behavior Analytics (UEBA) in Microsoft Sentinel
- Tutorial: Log network traffic to and from a virtual machine using the Azure portal
- Packet capture
- Monitor networks proactively with alerts and Azure Functions using Packet Capture
- What is Identity Protection?
- Identity Protection
- riskDetection
- riskyUser
- Learn how to add continuous security validation to your CI/CD pipeline

Security checklist

Refer to the complete set of recommendations.

Security checklist

Feedback

Was this page helpful? Yes No

Recommendations for security testing

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Security checklist recommendation:

> **SE:11** **Establish a comprehensive testing regimen that combines approaches to prevent security issues, validate threat prevention implementations, and test threat detection mechanisms.**

Rigorous testing is the foundation of good security design. Testing is a tactical form of validation to make sure controls are working as intended. Testing is also a proactive way to detect vulnerabilities in the system.

Establish testing rigor through cadence and verification from multiple perspectives. You should include inside-out viewpoints that test platform and infrastructure and outside-in evaluations that test the system like an external attacker.

This guide provides recommendations for testing the security posture of your workload. Implement these testing methods to improve your workload's resistance to attacks and maintain confidentiality, integrity, and availability of resources.

Definitions

Term	Definition
Application security testing (AST)	A Microsoft Security Development Lifecycle (SDL) technique that uses white-box and black-box testing methodologies to check for security vulnerabilities in code.
Black-box testing	A testing methodology that validates the externally visible application behavior without knowledge of the internals of the system.
Blue team	A team that defends against the attacks of the red team in a war game exercise.
Penetration testing	A testing methodology that uses ethical hacking techniques to validate the security defenses of a system.
Red team	A team that plays the role of an adversary and attempts to hack the system in a war game exercise.
Security Development Lifecycle (SDL)	A set of practices provided by Microsoft that supports security assurance and compliance requirements.
Software development lifecycle (SDLC)	A multistage, systematic process for developing software systems.

Term	Definition
White-box testing	A testing methodology where the structure of the code is known to the practitioner.

Key design strategies

Testing is a nonnegotiable strategy, especially for security. It allows you to proactively discover and address security issues before they can be exploited and to verify that the security controls that you implemented are functioning as designed.

The scope of testing must include the application, infrastructure, and automated and human processes.

> **Note**
>
> This guidance makes a distinction between testing and incident response. Although testing is a detection mechanism that ideally fixes issues prior to production, it shouldn't be confused with the remediation or investigation that's done as part of incident response. The aspect of recovering from security incidents is described in **Incident Response recommendations**.

SDL includes several types of tests that catch vulnerabilities in code, verify runtime components, and use ethical hacking to test the security resilience of the system. SDL is a key shift-left activity. You should run tests like static code analysis and automated scanning of infrastructure as code (IaC) as early in the development process as possible.

Be involved in test planning. The workload team might not design the test cases. That task is often centralized in the enterprise or completed by external security experts. The workload team should be involved in that design process to ensure that security assurances integrate with the application's functionality.

Think like an attacker. Design your test cases with the assumption that the system has been attacked. That way, you can uncover the potential vulnerabilities and prioritize the tests accordingly.

Run tests in a structured manner and with a repeatable process. Build your testing rigor around cadence, types of tests, driving factors, and intended outcomes.

Use the right tool for the job. Use tools that are configured to work with the workload. If you don't have a tool, buy the tool. Don't build it. Security tools are highly specialized, and building your own tool might introduce risks. Take advantage of the expertise and

tools offered by central SecOps teams or by external means if the workload team doesn't have that expertise.

Set up separate environments. Tests can be classified as destructive or nondestructive. Nondestructive tests aren't invasive. They indicate there's a problem, but they don't alter functionality in order to remediate the problem. Destructive tests are invasive and might damage functionality by deleting data from a database.

Testing in production environments gives you the best information but causes the most disruption. You tend to do only nondestructive tests in production environments. Testing in nonproduction environments is typically less disruptive but might not accurately represent the production environment's configuration in ways that are important to security.

If you deploy by using IaC and automation, consider whether you can create an isolated clone of your production environment for testing. If you have a continuous process for routine tests, we recommend using a dedicated environment.

Always evaluate the test results. Testing is a wasted effort if the results aren't used to prioritize actions and make improvements upstream. Document the security guidelines, including best practices, that you uncover. Documentation that captures results and remediation plans educates the team about the various ways that attackers might try to breach security. Conduct regular security training for developers, admins, and testers.

When you design your test plans, think about the following questions:

- How often do you expect the test to run, and how does it affect your environment?
- What are the different test types that you should run?

How often do you expect the tests to run?

Test the workload regularly to make sure changes don't introduce security risks and that there aren't any regressions. The team must also be ready to respond to organizational security validations that might be conducted at any time. There are also tests that you can run in response to a security incident. The following sections provide recommendations on the frequency of tests.

Routine tests

Routine tests are conducted at a regular cadence, as part of your standard operating procedures and to meet compliance requirements. Various tests might be run at different cadences, but the key is that they're conducted periodically and on a schedule.

You should integrate these tests into your SDLC because they provide defense in depth at each stage. Diversify the test suite to verify assurances for identity, data storage and transmission, and communication channels. Conduct the same tests at different points in the lifecycle to ensure that there aren't any regressions. Routine tests help establish an initial benchmark. However that's just a starting point. As you uncover new issues at the same points of the lifecycle, you add new test cases. The tests also improve with repetition.

At each stage, these tests should validate code that's added or removed or configuration settings that have changed in order to detect the security impact of those changes. You should improve the tests' efficacy with automation, balanced with peer reviews.

Consider running security tests as part of an automated pipeline or scheduled test run. The sooner you discover security issues, the easier it is to find the code or configuration change that causes them.

Don't rely only on automated tests. Use manual testing to detect vulnerabilities that only human expertise can catch. Manual testing is good for exploratory use cases and finding unknown risks.

Improvised tests

Improvised tests provide point-in-time validation of security defenses. Security alerts that might affect the workload at that time trigger these tests. Organizational mandates might require a pause-and-test mindset to verify the effectiveness of defense strategies if the alert escalates to an emergency.

The benefit of improvised tests is preparedness for a real incident. These tests can be a forcing function to do user acceptance testing (UAT).

The security team might audit all workloads and run these tests as needed. As a workload owner, you need to facilitate and collaborate with security teams. Negotiate enough lead time with security teams so that you can prepare. Acknowledge and communicate to your team and stakeholders that these disruptions are necessary.

In other cases, you might be required to run tests and report the security state of the system against the potential threat.

>  **Tradeoff**: Because improvised tests are disruptive events, expect to reprioritize tasks, which may delay other planned work.

> **Risk**: There's risk of the unknown. Improvised tests might be one-time efforts without established processes or tools. But the predominant risk is the potential interruption of the rhythm of business. You need to evaluate those risks relative to the benefits.

Security incident tests

There are tests that detect the cause of a security incident at its source. These security gaps must be resolved to make sure the incident isn't repeated.

Incidents also improve test cases over time by uncovering existing gaps. The team should apply the lessons learned from the incident and routinely incorporate improvements.

What are the different types of tests?

Tests can be categorized by **technology** and by **testing methodologies**. Combine those categories and approaches within those categories to get complete coverage.

By adding multiple tests and types of tests, you can uncover:

- Gaps in security controls or compensating controls.
- Misconfigurations.
- Gaps in observability and detection methods.

A good threat modeling exercise can point to key areas to ensure test coverage and frequency. For recommendations on threat modeling, see Recommendations for securing a development lifecycle.

Most tests described in these sections can be run as routine tests. However, repeatability can incur costs in some cases and cause disruption. Consider those tradeoffs carefully.

Tests that validate the technology stack

Here are some examples of types of tests and their focus areas. This list isn't exhaustive. Test the entire stack, including the application stack, front end, back end, APIs, databases, and any external integrations.

- Data security: Test the effectiveness of data encryption and access controls to ensure data is properly protected from unauthorized access and tampering.

- Network and connectivity: Test your firewalls to ensure they only allow expected, allowed, and safe traffic to the workload.
- Application: Test source code through application security testing (AST) techniques to make sure that you follow secure coding practices and to catch runtime errors like memory corruption and privilege issues. For details, see these community links.
- Identity: Evaluate whether the role assignments and conditional checks work as intended.

Test methodology

There are many perspectives on testing methodologies. We recommend tests that enable threat hunting by simulating real-world attacks. They can identify potential threat actors, their techniques, and their exploits that pose a threat to the workload. Make the attacks as realistic as possible. Use all the potential threat vectors that you identify during threat modeling.

Here are some advantages of testing through real-world attacks:

- When you make these attacks a part of routine testing, you use an outside-in perspective to check the workload and make sure the defense can withstand an attack.
- Based on the lessons they learned, the team upgrades their knowledge and skill level. The team improves situational awareness and can self-assess their readiness to respond to incidents.

>  **Risk**: Testing in general can affect performance. There might be business continuity problems if destructive tests delete or corrupt data. There are also risks associated with information exposure; make sure to maintain the confidentiality of data. Ensure the integrity of data after you complete testing.

Some examples of simulated tests include black-box and white-box testing, penetration testing, and war game exercises.

Black-box and white-box testing

These test types offer two different perspectives. In black-box tests, the internals of the system aren't visible. In white-box tests, the tester has a good understanding of the application and even has access to code, logs, resource topology, and configurations for conducting the experiment.

> **Risk**: The difference between the two types is upfront cost. White-box testing can be expensive in terms of time taken to understand the system. In some cases, white-box testing requires you to purchase specialized tools. Black-box testing doesn't need ramp-up time, but it might not be as effective. You might need to put in extra effort to uncover issues. It's a time investment tradeoff.

Tests that simulate attacks through penetration testing

Security experts who aren't part of the organization's IT or application teams conduct penetration testing, or *pentesting*. They look at the system in the way that malicious actors scope an attack surface. Their goal is to find security gaps by gathering information, analyzing vulnerabilities, and reporting the results.

> **Tradeoff**: Penetration tests are improvised and can be expensive in terms of disruptions and monetary investment because pentesting is typically a paid offering by third-party practitioners.

> **Risk**: A pentesting exercise might affect the runtime environment and might disrupt the availability for normal traffic.

The practitioners might need access to sensitive data in the entire organization. Follow the rules of engagement to ensure that access isn't misused. See the resources listed in Related links.

Tests that simulate attacks through war game exercises

In this methodology of simulated attacks, there are two teams:

- The *red* team is the adversary attempting to model real-world attacks. If they're successful, you find gaps in your security design and evaluate the blast radius containment of their breaches.
- The *blue* team is the workload team that defends against the attacks. They test their ability to detect, respond, and remediate the attacks. They validate the defenses that have been implemented to protect workload resources.

If they're conducted as routine tests, war game exercises can provide ongoing visibility and assurance that your defenses work as designed. War game exercises can potentially test across levels within your workloads.

A popular choice to simulate realistic attack scenarios is the Microsoft Defender for Office 365 Attack simulation training.

For more information, see Insights and reports for Attack simulation training.

For information about red-team and blue-team setup, see Microsoft Cloud Red Teaming.

Azure facilitation

Microsoft Sentinel is a native control that combines security information event management (SIEM) and security orchestration automated response (SOAR) capabilities. It analyzes events and logs from various connected sources. Based on data sources and their alerts, Microsoft Sentinel creates incidents and performs threat analysis for early detection. Through intelligent analytics and queries, you can proactively hunt for security issues. If there's an incident, you can automate workflows. Also, with workbook templates, you can quickly gain insights through visualization.

For product documentation, see Hunting capabilities in Microsoft Sentinel.

Microsoft Defender for Cloud offers vulnerability scanning for various technology areas. For details, see Enable vulnerability scanning with Microsoft Defender Vulnerability Management - Microsoft Defender for Cloud.

The practice of DevSecOps integrates security testing as part of an ongoing and continuous improvement mindset. War game exercises are a common practice that's integrated into the rhythm of business at Microsoft. For more information, see Security in DevOps (DevSecOps).

Azure DevOps supports third-party tools that can be automated as part of the continuous integration/continuous deployment pipelines. For details, see Enable DevSecOps with Azure and GitHub - Azure DevOps.

Related links

Follow the rules of engagement to make sure that access isn't misused. For guidance about planning and executing simulated attacks, see the following articles:

- Penetration Testing Rules of Engagement
- Penetration testing

You can simulate denial of service (DoS) attacks in Azure. Be sure to follow the policies laid out in Azure DDoS Protection simulation testing.

Community links

Application security testing: Tools, types, and best practices - GitHub Resources describes the types of testing methodologies that can test the build-time and runtime defenses of the application.

Penetration Testing Execution Standard (PTES) provides guidelines about common scenarios and the activities required to establish a baseline.

OWASP Top Ten | OWASP Foundation provides security best practices for applications and test cases that cover common threats.

Security checklist

Refer to the complete set of recommendations.

Security checklist

Recommendations for security incident response

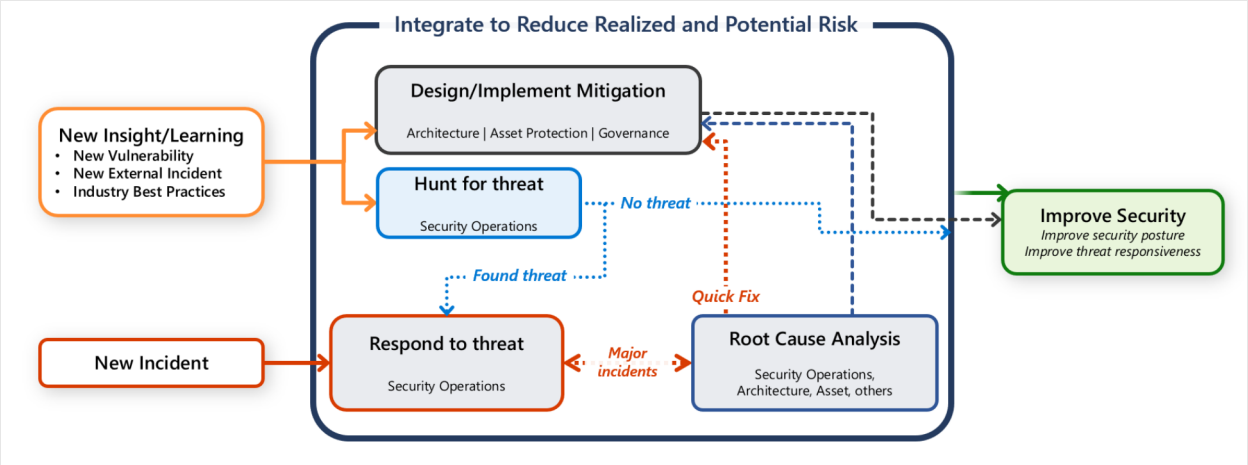
Article • 11/14/2023

Applies to Azure Well-Architected Framework Security checklist recommendation:

> **SE:12** **Define and test effective incident response procedures that cover a spectrum of incidents, from localized issues to disaster recovery. Clearly define which team or individual runs a procedure.**

This guide describes the recommendations for implementing a security incident response for a workload. If there's a security compromise to a system, a systematic incident response approach helps to reduce the time that it takes to identify, manage, and mitigate security incidents. These incidents can threaten the confidentiality, integrity, and availability of software systems and data.

Most enterprises have a central security operation team (also known as Security Operations Center (SOC), or SecOps). The responsibility of the security operation team is to rapidly detect, prioritize, and triage potential attacks. The team also monitors security-related telemetry data and investigates security breaches.



However, you also have a responsibility to protect your workload. It's important that any communication, investigation, and hunting activities are a collaborative effort between workload team and SecOps team.

This guide provides recommendations for you and your workload team to help you rapidly detect, triage, and investigate attacks.

Definitions

Term	Definition
Alert	A notification that contains information about an incident.
Alert fidelity	The accuracy of the data that determines an alert. High-fidelity alerts contain the security context that's needed to take immediate actions. Low-fidelity alerts lack information or contain noise.
False positive	An alert that indicates an incident that didn't happen.
Incident	An event that indicates unauthorized access to a system.
Incident response	A process that detects, responds to, and mitigates risks that are associated with an incident.
Triage	An incident response operation that analyzes security issues and prioritizes their mitigation.

Key design strategies

You and your team perform incident response operations when there's a signal or alert for a potential compromise. High-fidelity alerts contain ample security context that makes it easy for analysts to make decisions. High-fidelity alerts result in a low number of false positives. This guide assumes that an alerting system filters low-fidelity signals and focuses on high-fidelity alerts that might indicate a real incident.

Assign incident notification

Security alerts need to reach the appropriate people on your team and in your organization. Establish a designated point of contact on your workload team to receive incident notifications. These notifications should include as much information as possible about the resource that's compromised and the system. The alert must include the next steps, so your team can expedite actions.

We recommend that you log and manage incident notifications and actions by using specialized tooling that keeps an audit trail. By using standard tools, you can preserve evidence that might be required for potential legal investigations. Look for opportunities to implement automation that can send notifications based on the responsibilities of accountable parties. Keep a clear chain of communication and reporting during an incident.

Take advantage of security information event management (SIEM) solutions and security orchestration automated response (SOAR) solutions that your organization provides.

Alternatively, you can procure incident management tools and encourage your organization to standardize them for all workload teams.

Investigate with a triage team

The team member that receives an incident notification is responsible for setting up a triage process that involves the appropriate people based on the available data. The triage team, often called the *bridge team*, must agree on the mode and process of communication. Does this incident require asynchronous discussions or bridge calls? How should the team track and communicate the progress of investigations? Where can the team access incident assets?

Incident response is a crucial reason to keep documentation up to date, like the architectural layout of the system, information at a component level, privacy or security classification, owners, and key points of contact. If the information is inaccurate or outdated, the bridge team wastes valuable time trying to understand how the system works, who's responsible for each area, and what the effect of the event might be.

For further investigations, involve the appropriate people. You might include an incident manager, security officer, or workload-centric leads. To keep the triage focused, exclude people that are outside of the scope of the problem. Sometimes separate teams investigate the incident. There might be a team that initially investigates the issue and tries to mitigate the incident, and another specialized team that might perform forensics for a deep investigation to ascertain wide issues. You can quarantine the workload environment to enable the forensics team to do their investigations. In some cases, the same team might handle the entire investigation.

In the initial phase, the triage team is responsible for determining the potential vector and its effect on the confidentiality, integrity, and availability (also called the *CIA*) of the system.

Within the categories of CIA, assign an initial severity level that indicates the depth of the damage and the urgency of remediation. This level is expected to change over time as more information is discovered in the levels of triage.

In the discovery phase, it's important to determine an immediate course of action and communication plans. Are there any changes to the running state of the system? How can the attack be contained to stop further exploitation? Does the team need to send out internal or external communication, such as a responsible disclosure? Consider detection and response time. You might be legally obligated to report some types of breaches to a regulatory authority within a specific time period, which is often hours or days.

If you decide to shut down the system, the next steps lead to the workload's disaster recovery (DR) process.

If you don't shut down the system, determine how to remediate the incident without affecting the functionality of the system.

Recover from an incident

Treat a security incident like a disaster. If the remediation requires complete recovery, use proper DR mechanisms from a security perspective. The recovery process must prevent chances of recurrence. Otherwise, recovery from a corrupted backup reintroduces the issue. Redeploying a system with the same vulnerability leads to the same incident. Validate failover and failback steps and processes.

If the system remains functioning, assess the effect on the running parts of the system. Continue to monitor the system to ensure that other reliability and performance targets are met or readjusted by implementing proper degradation processes. Don't compromise privacy due to mitigation.

Diagnosis is an interactive process until the vector, and a potential fix and fallback, is identified. After diagnosis, the team works on remediation, which identifies and applies the required fix within an acceptable period.

Recovery metrics measure how long it takes to fix an issue. In the event of a shutdown, there might be an urgency regarding the remediation times. To stabilize the system, it takes time to apply fixes, patches, and tests, and deploy updates. Determine containment strategies to prevent further damage and the spread of the incident. Develop eradication procedures to completely remove the threat from the environment.

>  **Tradeoff**: There's a tradeoff between reliability targets and remediation times. During an incident, it's likely that you don't meet other nonfunctional or functional requirements. For example, you might need to disable parts of your system while you investigate the incident, or you might even need to take the entire system offline until you determine the scope of the incident. Business decision-makers need to explicitly decide what the acceptable targets are during the incident. Clearly specify the person that's accountable for that decision.

Learn from an incident

An incident uncovers gaps or vulnerable points in a design or implementation. It's an improvement opportunity that's driven by lessons in technical design aspects,

automation, product development processes that include testing, and the effectiveness of the incident response process. Maintain detailed incident records, including actions taken, timelines, and findings.

We highly recommended that you conduct structured post-incident reviews, such as root-cause analysis and retrospectives. Track and prioritize the outcome of those reviews, and consider using what you learn in future workload designs.

Improvement plans should include updates to security drills and testing, like business continuity and disaster recovery (BCDR) drills. Use security compromise as a scenario for performing a BCDR drill. Drills can validate how the documented processes work. There shouldn't be multiple incident response playbooks. Use a single source that you can adjust based on the size of the incident and how widespread or localized the effect is. Drills are based on hypothetical situations. Conduct drills in a low-risk environment, and include the learning phase in the drills.

Conduct post-incident reviews, or postmortems, to identify weaknesses in the response process and areas for improvement. Based on the lessons you learn from the incident, update the incident response plan (IRP) and the security controls.

Send the necessary communication

Implement a communication plan to notify users of a disruption and to inform internal stakeholders about the remediation and improvements. Other people in your organization need to be notified of any changes to the workload's security baseline to prevent future incidents.

Generate incident reports for internal use and, if necessary, for regulatory compliance or legal purposes. Also, adopt a standard format report (a document template with defined sections) that the SOC team uses for all incidents. Ensure that every incident has a report associated with it before you close the investigation.

Azure facilitation

Microsoft Sentinel is an SIEM and SOAR solution. It's a single solution for alert detection, threat visibility, proactive hunting, and threat response. For more information, see What's Microsoft Sentinel?

Ensure that the Azure enrollment portal includes administrator contact information so security operations can be notified directly via an internal process. For more information, see Update notification settings.

To learn more about establishing a designated point of contact that receives Azure incident notifications from Microsoft Defender for Cloud, see Configure email notifications for security alerts.

Organizational alignment

Cloud Adoption Framework for Azure provides guidance about incident response planning and security operations. For more information, see Security operations.

Related links

- Automatically create incidents from Microsoft security alerts
- Conduct end-to-end threat hunting by using the hunts feature
- Configure email notifications for security alerts
- Incident response overview
- Microsoft Azure incident readiness
- Navigate and investigate incidents in Microsoft Sentinel
- Security control: Incident response
- SOAR solutions in Microsoft Sentinel
- Training: Introduction to Azure incident readiness
- Update Azure portal notification settings
- What's an SOC?
- What's Microsoft Sentinel?

Security checklist

Refer to the complete set of recommendations.

Security checklist

Cost optimization quick links

Apply cost optimization guidance in your architecture to sustain and improve your return on investment (ROI).

Learn key points

QUICKSTART

Design principles

Checklist

Tradeoffs

Cost optimization patterns

Azure Well-Architected Review assessment

TRAINING

Cost optimization

VIDEO

What tradeoffs have you made to optimize for cost?

Review design principles

CONCEPT

Develop cost-management discipline

Design with a cost-efficiency mindset

Design for usage optimization

Design for rate optimization

Monitor and optimize over time

Set, measure, and protect financial targets

HOW-TO GUIDE

Create a culture of financial responsibility

Create a cost model

Collect and review cost data

Set spending guardrails

Take cost optimization actions

HOW-TO GUIDE

Get the best rates

Align usage to billing increments

Optimize component costs

Consolidate

Optimize

HOW-TO GUIDE

Environments

Flows

Data

Application code

Scaling

Personnel time

Explore related resources

REFERENCE

Azure Advisor: Cost recommendations

Azure Pricing Calculator

Microsoft Cost Management

Visualize cost reports

Cost Optimization design principles

05/29/2025

Architecture design is always driven by business goals and must **factor in return on investment (ROI) and financial constraints**. Typical questions to consider include:

- Do the allocated budgets enable you to meet your goals?
- What's the spending pattern for the application and its operations? What are priority areas?
- How will you maximize the investment in resources, by better utilization or by reduction?

A cost-optimized workload isn't necessarily a low-cost workload. There are significant tradeoffs. Tactical approaches are reactive and can reduce costs only in the short term. To achieve long-term financial responsibility, you need to **create a strategy with prioritization, continuous monitoring, and repeatable processes** that focuses on optimization.

The design principles are intended to provide optimization strategies that you need to consider when you design and implement your workload architecture. Start with the recommended approaches and **justify the benefits for a set of business requirements**. After you set your strategy, drive actions by using the Cost Optimization checklist as your next step.

As you prioritize business requirements to align with technology needs, expect the initial cost allocation to change. However, you should expect a series of tradeoffs in areas in which you want to optimize cost, such as security, scalability, resilience, and operability. If the cost of addressing the challenges in those areas is high and these principles aren't applied properly, you might make risky choices in favor of a cheaper solution, ultimately affecting your organization's business goals and reputation.

Develop cost-management discipline

Expand table

	**Build a team culture that has awareness of budget, expenses, reporting, and cost tracking.**

Cost optimization is conducted at various levels of the organization. It's important to understand how your workload cost is aligned with organizational FinOps practices. A view into the business units, resource organization, and centralized audit policies allows you to adopt a standardized financial system.

Expand table

Approach	Benefit
Develop a cost model. This fundamental exercise is a prerequisite to setting up a financial tracking system.	A cost model helps segment expenses and estimate and forecast the total cost of ownership, including infrastructure, support, and implementation. It enables you to identify cost drivers early and predict how any change, growth, or shrinkage will affect overall spending in your projected business model.
Have an effective but flexible accountability model that's governed and implemented with properly assigned roles and responsibilities.	Clear accountability helps enforce the functional expectations of each role (given a scope), drive clarity, and generate reports with transparency at desired levels. Proactive governance can help you avoid actions that might lead to unnecessary expenditure that's beyond the budget.
Estimate realistic budgets that cover all non-negotiable functional and nonfunctional requirements, personnel, and processes that provide for anticipated growth.	You'll be able to set financial boundaries and establish ways to check your spending against the allocated budget. You'll also get notifications when certain thresholds are exceeded, which prevents overspending at the tenant scope, resource scope, and other scopes that are applied to the budget.
For workloads governed by service-level agreements (SLAs), evaluate whether to allocate budget toward potential penalties or toward implementation efforts.	A well-implemented solution can help you avoid penalties altogether, making proactive investment. Setting aside a customer compensation budget is a pragmatic approach to reduce the risk of future liability. Collaborate with your product owner to negotiate realistic cost compensation budget.
Plan on training costs, hiring expenses, and the cost of infrastructure needed to augment skills as the workload matures.	Investing in staffing complements existing skills through full-time or vendor support.
Communicate cost implications of design changes that are driven by insights gained from production.	The organization is able to make practical budget adjustment based on production feedback, which should be considered as meaningful as numeric data.

Design with a cost-efficiency mindset

Expand table



Spend only on what you need to achieve the highest return on your investments.

Every architectural decision has direct and indirect financial implications. Understand the costs associated with build versus buy options, technology choices, the billing model and licensing,

training, operations, and so on.

Given a set of requirements, optimize, and make tradeoff decisions, in relation to costs, that still effectively address the cross-cutting concerns of the workload.

Expand table

Approach	Benefit
Establish a cost baseline, including the projected growth. Ensure design choices work within the allocated budget to meet the functional and nonfunctional requirements. Factor in expenses related to technology choices, automation, acquisition, training, and change management as part of that total cost.	Cost estimates enable you to forecast expenses against the budget and pinpoint key cost drivers. They also help reveal hidden costs that might otherwise go unnoticed, supporting a balanced approach that avoids overengineering. This process will also generate decision trees for technology options based on cost considerations. By eliminating high-cost alternatives that lack strong business justification, you can free up budget capacity to invest in higher-value opportunities. We don't recommend that you design beyond planned growth as it may result in a lower ROI.
Design and enforce cost guardrails in your architecture that keep the resources within the upper and lower limits.	Enforcement can prevent incidental or unapproved charges and ensure that only budgeted quantity of resources are provisioned.
Treat different SDLC environments differently, and deploy the right number of environments.	You can save money by understanding that not all environments need to simulate production. Nonproduction environments can have different features, SKUs, instance counts, and even logging. You also can save costs by creating preproduction environments on-demand and removing them when you no longer need them.

Design for usage optimization

Expand table



Maximize the use of resources and operations. Apply them to the negotiated functional and nonfunctional requirements of the solution.

Services and offerings provide various capabilities and pricing tiers. After you purchase a set of features, avoid underutilizing them. Find ways to maximize your investment in the tier.

Likewise, continuously evaluate billing models to find those that better align to your usage, based on current production workloads.

Expand table

Approach	Benefit
Take advantage of the full capabilities of your selected resource SKUs to meet performance, security, reliability, and operational goals.	You can maximize the use of what you paid for. Avoid selecting SKUs with features you don't need, as they can lead to unnecessary costs without added benefit.
Evaluate opportunities to dynamically adjust capacity, scaling up when demand increases and scaling down when it's no longer needed.	Without this approach, you may need to pre-provision more capacity than necessary. By contrast, dynamic scaling enables you to maintain a minimum baseline and expand only when required, aligning resource consumption with actual usage patterns.
Prioritize deployment of active-active models over active-passive models, as part of your recovery plan, if you already paid for the resources.	If your design defaults to using active-passive models, you might have idle resources that could otherwise be used. Converting to active-active might enable you to meet your load leveling and scale bursting requirements without overspending.
Prioritize the use of commitment-based discounted resources when developing new features, setting up additional environments, or optimizing for nonfunctional requirements.	Finding opportunities to use committed plans can significantly reduce the cost of implementing new functionality.
Make the most of your investment in a support plan. Keep allowance for training to ensure the team are using relevant tools and technologies.	Using your support plan to handle production problems or for proactive reviews will help you get your money's worth. Fully engage with your Microsoft support model.

Design for rate optimization

Expand table

Increase efficiency without redesigning, renegotiating, or sacrificing functional or nonfunctional requirements.

Take advantage of opportunities to optimize the utility and costs of your existing resources and operations. If you don't, you unnecessarily spend money without any added ROI.

Expand table

Approach	Benefit
Identify resources that have stable or predictable usage patterns over time. Optimize costs by prepurchasing these resources to take advantage of available discounts. Collaborate with your licensing team to influence future purchasing agreements and renewal strategies.	Microsoft offers discounted rates for predictable, long-term commitments to specific resources or resource categories. These resources incur lower costs during the usage period and can be amortized over time. By keeping your licensing team informed about current and projected resource investments, you can help them right-size commitments during agreement negotiations. In some cases, these forecasts can influence your organization's price sheet, benefiting not only your workload's cost efficiency but also other teams using the same technologies.
Explore alternatives that don't require additional licensing. Consider options like hybrid use and preproduction subscription pricing.	You'll be able to reduce licensing costs by taking advantage of options that give you usage rights to the same or comparable technologies at a lower cost.
Use consumption-based pricing when it's more cost effective.	You'll pay for what you use. This option might be more expensive than a fully utilized prepaid option. However, if you don't expect to fully utilize pre-purchased compute, pay-as-you-go might be a better choice.
Use fixed-price billing instead of consumption-based billing for a resource when its utilization is high and predictable and a comparable SKU or billing option is available.	When utilization is high and predictable, the fixed-price model usually costs less and often supports more features.
Where possible, co-locate usage with other workloads, resources, and teams to reduce financial and operational costs.	Shared resources are managed centrally and provisioned with higher capacity to support multiple workloads, allowing costs to be distributed across teams.
Deploy to lower-cost regions, provided there are no compromises to functional or nonfunctional requirements. Evaluate regional options for each environment individually. While production may require specific regions, consider leveraging more cost-effective regions for preproduction environments where feasible.	Use premium regions only where necessary can lead to significant savings. Also, savings from non-production environments can be reallocated to other priorities.
Prefer services that make it easier to achieve higher density.	As density increases, the amount of resources that you need to run a workload decreases. This decrease reduces cost per unit and the cost of management.

Approach	Benefit
Consider the potential tradeoffs, especially on security boundaries.	

Monitor and optimize over time

Expand table



Continuously right-size investment as your workload evolves with the ecosystem.

What was important yesterday might not be important today. As you learn through evaluation of production workloads, expect changes in architecture, business requirements, processes, and even team structure. Your software development lifecycle (SDLC) practices might need to evolve. External factors might also change, like the cloud platform, its resources, and your agreements.

You should carefully assess the impact of all changes on cost. Monitor changes and the ROI trend on a regular cadence, and evaluate whether you need to adjust functional and nonfunctional requirements.

Expand table

Approach	Benefit
Build capabilities in the system that capture and classify expense.	You'll be able to calculate the costs that reveal technical and business perspectives at different billing boundaries. You'll also be able to conduct regular reviews and drive showback and chargeback processes.
Implement cost alerts when spending approaches predefined budget thresholds. Regularly review and adjust these alerts to ensure they remain aligned with evolving usage patterns.	Proactive notifications help prevent budget overruns and support timely decision-making.
Continuously evaluate and adjust architecture design decisions around cost of resources, operations, and paid support.	Regular reviews of metrics, performance data, billing reports, and feature usage might lead to fine-tuning that can reduce costs. You might also be able to save some costs by evaluating the use of your support contract and right-sizing it.

Approach	Benefit
Decommission resources that are underutilized, unused, obsolete, or can be replaced with more efficient alternatives. Regularly delete unnecessary data.	By resizing or removing underutilized resources, or even changing SKUs, you can reduce costs. Shutting down unused resources and deleting data when you no longer need it reduces waste and frees up funds so you can invest them elsewhere.

Next steps

Cost Optimization checklist

Design review checklist for Cost Optimization

Article • 11/14/2023

This checklist presents a set of recommendations about cost optimization for your workload to help you achieve a high return on investment (ROI) based on the business value that your workload delivers. Cost optimization balances actual costs versus perceived value, team efficiency, focus, and effort, while meeting the defined functional and nonfunctional requirements of your workload.

Every workload has direct and indirect costs, and every workload is designed to deliver value. If you don't incorporate the recommendations in this article and consider the tradeoffs, your design might not make the best use of your time and money. Carefully consider the points covered in the following checklist to instill confidence in your design's success.

Cost optimization is a continuous process in which you optimize workload costs and align your workload with the broader governance discipline of cost management. What's important today might not be important tomorrow. Technology choices or options and features that your platform offers today might be different. Learn from production and nonproduction environments, be aware of platform changes, and apply your findings to your workload and your workload's dependencies.

Checklist

	Code	Recommendation
☐	CO:01	**Create a culture of financial responsibility.** Regularly train personnel so technical skills remain sharp. Foster creativity and spending accountability in the work environment. Invest in tooling and implementing automation.
☐	CO:02	**Create and maintain a cost model.** A cost model should estimate the initial cost, run rates, and ongoing costs. Negotiate a budget that covers a cost model and has a buffer for unplanned spending.
☐	CO:03	**Collect and review cost data.** Data collection should capture daily costs. In cost reports, include incurred costs (metered), prepaid costs (amortized), trends, and forecasts. Stakeholders should regularly review spending against the budget and cost model. Automate alerts to trigger notifications at key thresholds and detect anomalies to indicate deviations from trend baselines.
☐	CO:04	**Set spending guardrails.** Guardrails should include release gates, governance policies, resource limits, and access controls. Prioritize platform automation over

	Code	Recommendation
		manual processes.
☐	CO:05	**Get the best rates from providers.** You should find and use the best rates for cloud resources and licenses. Regularly review cost savings. Cost reviews should include regional pricing, pricing tiers, pricing models (consumption or commitment-based), license portability, corporate purchasing plans, and price sheets.
☐	CO:06	**Align usage to billing increments.** You should understand billing increments (meters) and align resource usage to those increments. Modify the service to align with billing increments, or modify resource usage to align with billing increments. Consider using a proof-of-concept to validate billing knowledge and design choices for major cost drivers and to reveal ways to align billing and resource usage.
☐	CO:07	**Optimize component costs.** Regularly remove or optimize legacy, unneeded, and underutilized workload components, including application features, platform features, and resources.
☐	CO:08	**Optimize environment costs.** Align spending to prioritize preproduction, production, operations, and disaster recovery environments. For each environment, consider the required availability, licensing, operating hours and conditions, and security. Nonproduction environments should emulate the production environment. Implement strategic tradeoffs into nonproduction environments.
☐	CO:09	**Optimize flow costs.** Align the cost of each flow with flow priority. When you prioritize flows, consider the features, functionality, and nonfunctional requirements of each flow. Optimizing flow spend often requires strategic compromises.
☐	CO:10	**Optimize data costs.** Data spending with data priority. Data optimization should include improvements to data management (tiering and retention), volume, replication, backups, file formats, and storage solutions.
☐	CO:11	**Optimize code costs.** Evaluate and modify code to meet functional and nonfunctional requirements with fewer or cheaper resources.
☐	CO:12	**Optimize scaling costs.** Evaluate alternative scaling within your scale units. Consider alternative scaling configurations, and align with the cost model. Considerations should include utilization against the inherit limits of every instance, resource, and scale unit boundary. Use strategies for controlling demand and supply.
☐	CO:13	**Optimize personnel time.** Align the time personnel spends on tasks with the priority of the task. The goal is to reduce the time spent on tasks without degrading the outcome. Optimization efforts should include minimizing noise, reducing build times, high fidelity debugging, and production mocking.
☐	CO:14	**Consolidate resources and responsibility.** Look within the workload for ways to consolidate resources and increase density. Outside the workload, use existing centralized resources and services that enable you to consolidate workload responsibilities.

Next steps

We recommend that you review the Cost Optimization tradeoffs to explore other concepts.

Cost Optimization tradeoffs

Cost Optimization tradeoffs

Article • 10/10/2024

When you design a workload to maximize return on investment (ROI) under financial constraints, you first need clearly defined functional and nonfunctional requirements. A work and effort prioritization strategy is essential. The foundation is a team that has a strong sense of financial responsibility. The team should have a strong understanding of available technologies and their billing models.

After you understand the ROI of a workload, you can start improving it. To improve the ROI, consider how decisions based on the Cost Optimization design principles and the recommendations in the design review checklist for Cost Optimization might influence the goals and optimizations of other Azure Well-Architected Framework pillars. For cost optimization, it's important to avoid focusing on a cheaper solution. Choices that focus only on minimizing spending can increase the risk of undermining your workload's business goals and reputation. This article describes example tradeoffs that a workload team might encounter when considering the target setting, design, and operations for cost optimization.

Cost Optimization tradeoffs with Reliability

The cost of a service disruption must be measured against the cost of preventing or recovering from one. If the cost of disruptions exceeds the cost of reliability design, you should invest more to prevent or mitigate disruptions. Conversely, the cost of the reliability efforts might be more than the cost of a disruption, including factors like compliance requirements and reputation. You should consider strategic divestment in reliability design only in this scenario.

> **Tradeoff: Reduced resiliency.** A workload incorporates resiliency measures to attempt to avoid and withstand specific types and quantities of malfunction.

- To save money, the workload team might underprovision a component or overconstrain its scaling, making the component more likely to fail during sudden spikes in demand.
- Consolidating workload resources (*increasing density*) for cost optimization makes individual components more likely to fail during spikes in demand and during maintenance operations like updates.

- Removing components that support resiliency design patterns, like a message bus, and creating a direct dependency reduces self-preservation capabilities.
- Saving money by reducing redundancy can limit a workload's ability to handle concurrent malfunctions.
- Using budget SKUs might limit the maximum service-level objective (SLO) that the workload can reach.
- Setting hard spending limits can prevent a workload from scaling to meet legitimate demand.
- Without reliability testing tools or tests, the reliability of a workload is unknown, and it's less likely to meet reliability targets.

>  **Tradeoff: Limited recovery strategy.** A workload that's reliable has a tested incident response and recovery plan for disaster scenarios.

- Reduced testing or drilling of a workload's disaster recovery plan might affect the speed and effectiveness of recovery operations.
- Creating or retaining fewer backups decreases possible recovery points and increases the chance of losing data.
- Choosing a less expensive support contract with technology partners might increase workload recovery time due to potential delays in technical assistance.

>  **Tradeoff: Increased complexity.** A workload that uses straightforward approaches and avoids unnecessary or overengineered complexity is generally easier to manage in terms of reliability.

- Using cost-optimization cloud patterns can add new components, like a content delivery network (CDN), or shift duties to edge and client devices that a workload must provide reliability targets for.
- Event-based scaling can be more complicated to tune and validate than resource-based scaling.
- Reducing data volume and tiering data through data lifecycle actions, possibly in conjunction with implementing aggregated data points before a lifecycle event, introduces reliability factors to consider in the workload.

- Using different regions to optimize cost can make management, networking, and monitoring more difficult.

Cost Optimization tradeoffs with Security

The cost of a compromise to confidentiality, integrity, and availability in a workload must always be balanced against the cost of the effort to prevent that compromise. A security incident can have a wide range of financial and legal impacts and harm a company's reputation. Investing in security is a risk mitigation activity. The cost of experiencing the risks must be balanced against the investment. As a rule, don't compromise on security to gain cost optimizations that are below the point of responsible and agreed upon risk mitigation. Optimizing security costs by rightsizing solutions is an important optimization practice, but be aware of tradeoffs like the following when doing so.

>  **Tradeoff: Reduced security controls.** Security controls are established across multiple layers, sometimes redundantly, to provide defense in depth.

One cost optimization tactic is to look for ways to remove components or processes that accrue unit or operational costs. Removing security components like the following examples for the sake of saving money impacts security. You need to carefully perform a risk analysis on this impact.

- Reducing or simplifying authentication and authorization techniques compromises the *verify explicitly* principle of zero-trust architecture. Examples of these simplifications include using a basic authentication scheme like preshared keys rather than investing time to learn industry OAuth approaches, or using simplified role-based access control assignments to reduce management overhead.

- Removing encryption in transit or at rest to reduce costs on certificates and their operational processes exposes data to potential integrity or confidentiality breaches.

- Removing or reducing security scanning or inspection tooling or security testing because of the associated cost and time investment can directly impact the confidentiality, integrity, or availability that the tooling and testing is intended to protect.

- Reducing the frequency of security patching because of the operational time invested in cataloging and performing the patching affects a workload's ability to address evolving threats.

- Removing network controls like firewalls might lead to a failure to block malicious inbound and outbound traffic.

>  **Tradeoff: Increased workload surface area.** The Security pillar prioritizes a reduced and contained surface area to minimize attack vectors and the management of security controls.

Cloud design patterns that optimize costs sometimes necessitate the introduction of additional components. These additional components increase the surface area of the workload. The components and the data within them must be secured, possibly in ways that aren't already used in the system. These components and data are often subject to compliance. Examples of patterns that can introduce components include:

- Using the Static Content Hosting pattern to offload data to a new CDN component.
- Using the Valet Key pattern to offload processing and secure resource access to client compute.
- Using the Queue-Based Load Leveling pattern to smooth costs by introducing a message bus.

>  **Tradeoff: Removed segmentation.** The Security pillar prioritizes strong segmentation to support the application of targeted security controls and to control the blast radius.

Sharing resources, for example in multitenancy situations or co-locating multiple applications on a shared application platform, is an approach for reducing costs by increasing density and reducing the management surface. This increased density can lead to security concerns like these:

- Lateral movement between components that share resources is easier. A security event that compromises the availability of the application platform host or an individual application also has a larger blast radius.
- Co-located resources might share a workload identity and have less meaningful audit trails in access logs.
- Network security controls must be broad enough to cover all co-located resources. This configuration potentially violates the principle of least privilege for some resources.

- Co-locating disparate applications or data on a shared host can lead to extending compliance requirements and security controls to applications or data that would otherwise be out of scope. This broadening of scope necessitates additional security scrutiny and auditing effort on the co-located components.

Cost Optimization tradeoffs with Operational Excellence

>  **Tradeoff: Compromised software development lifecycle (SDLC) capacities.** A workload's SDLC process provides rigor, consistency, specificity, and prioritization to change management in a workload.

- Reducing testing efforts to save time and the cost associated with test personnel, resources, and tooling can result in more bugs in production.
- Delaying paying back technical debt to focus personnel efforts on new features can lead to slower development cycles and overall reduced agility.
- Deprioritizing documentation to focus personnel efforts on product development can lead to longer onboarding time for new employees, impact the effectiveness of incident response, and compromise compliance requirements.
- A lack of investment in training leads to stagnated skills, reducing the team's ability to adopt newer technologies and practices.
- Removing automation tooling to save money can result in personnel spending more time on the tasks that are no longer automated. It also increases the risk of errors and inconsistencies.
- Reducing planning efforts, like scoping and activity prioritization, to cut expenses can increase the likelihood of rework due to vague specifications and poor implementation.
- Avoiding or reducing continuous improvement activities, like retrospectives and after-incident reports, to keep the workload team focused on delivery can create missed opportunities to optimize routine, unplanned, and emergency processes.

>  **Tradeoff: Reduced observability.** Observability is necessary to help ensure that a workload has meaningful alerting and successful incident response.

- Decreasing log and metric volume to save on storage and transfer costs reduces system observability and can lead to:
  - Fewer data points for creating alerts related to reliability, security, and performance.
  - Coverage gaps in incident response activities.
  - Limited observability into interactions or boundaries related to security or compliance.

- Cost optimization design patterns can add components to a workload, increasing its complexity. The workload monitoring strategy must include those new components. For example, some patterns might introduce flows that span multiple components or shift processes from the server to the client. These changes can increase the complexity of correlating and tracking information.

- Reduced investment in observability tooling and the maintenance of effective dashboards can decrease the ability to learn from production, validate design choices, and inform product design. This reduction can also hamper incident response activities and make it harder to meet the recovery time objective and SLO.

>  **Tradeoff: Deferred maintenance.** Workload teams are expected to keep code, tooling, software packages, and operating systems patched and up to date in a timely and orderly way.

- Letting maintenance contracts with tooling vendors expire can result in missed optimization features, bug resolutions, and security updates.

- Increasing the time between system patches to save time can lead to missed bug fixes or a lack of protection against evolving security threats.

Cost Optimization tradeoffs with Performance Efficiency

The Cost Optimization and Performance Efficiency pillars both prioritize maximizing a workload's value. Performance Efficiency emphasizes meeting performance targets without spending more than necessary. Cost Optimization emphasizes maximizing the value produced by a workload's resources without exceeding performance targets. As a result, Cost Optimization often improves Performance Efficiency. However, there are Performance Efficiency tradeoffs associated with Cost Optimization. These tradeoffs can

make it harder to reach performance targets and hinder ongoing performance optimization.

>  **Tradeoff: Underprovisioned or underscaled resources.** A performance-efficient workload has enough resources to serve demand but doesn't have excessive unused overhead, even when usage patterns fluctuate.

- Reducing costs by downsizing resources can deprive applications of resources. The application might not be able to handle significant usage pattern fluctuations.
- Limiting or delaying scaling to cap or reduce costs might result in insufficient supply to meet demand.
- Autoscale settings that scale down aggressively to reduce costs might leave a service unprepared for sudden spikes in demand or cause frequent scaling fluctuations (flapping).

>  **Tradeoff: Lack of optimization over time.** Evaluating the effects of changes in functionality, changes in usage patterns, new technologies, and different approaches on the workload is one way to try to increase efficiency.

- Limiting the focus on developing expertise in performance optimization in order to prioritize delivery can cause missed opportunities for improving resource usage efficiency.
- Removing access performance testing or monitoring tools increases the risk of undetected performance issues. It also limits the ability for a workload team to execute on measure/improve cycles.
- Neglecting areas prone to performance degradation, like data stores, can gradually deteriorate query performance and elevate overall system usage.

Related links

Explore the tradeoffs for the other pillars:

- Reliability tradeoffs
- Security tradeoffs
- Operational Excellence tradeoffs
- Performance Efficiency tradeoffs

Feedback

Was this page helpful? Yes No

Cloud design patterns that support cost optimization

Article • 10/10/2024

When you design workload architectures, you should use industry patterns that address common challenges. Patterns can help you make intentional tradeoffs within workloads and optimize for your desired outcome. They can also help mitigate risks that originate from specific problems, which can affect reliability, security, performance, and operations. If not mitigated, risks will eventually increase costs. These patterns are backed by real-world experience, are designed for cloud scale and operating models, and are inherently vendor agnostic. Using well-known patterns as a way to standardize your workload design is a component of operational excellence.

Many design patterns directly support one or more architecture pillars. Design patterns that support the Cost Optimization pillar align with implementing favorable billing models, reducing overprovisioning, changing scaling dimensions, and maximizing value during migrations.

Design patterns for cost optimization

The following table summarizes cloud design patterns that support the goals of cost optimization.

Expand table

Pattern	Summary
Claim Check	Separates data from the messaging flow, providing a way to separately retrieve the data related to a message. Messaging systems often impose limits on message size, and increased size limits is often a premium feature. Reducing the size of message bodies might enable you to use a cheaper messaging solution.
Competing Consumers	Applies distributed and concurrent processing to efficiently handle items in a queue. This pattern can help you optimize costs by enabling scaling that's based on queue depth, down to zero when the queue is empty. It can also optimize costs by enabling you to limit the maximum number of concurrent consumer instances.
Compute Resource Consolidation	Optimizes and consolidates compute resources by increasing density. This pattern combines multiple applications or components of a workload on a shared infrastructure. Doing so maximizes the utilization of computing resources by avoiding unused provisioned capacity via

Pattern	Summary
	aggregation of components or even whole workloads on a pooled infrastructure. Container orchestrators are a common example.
Gateway Offloading	Offloads request processing to a gateway device before and after forwarding the request to a backend node. Adding an offloading gateway into the request process enables you to redirect costs from resources that would be spent per-node into the gateway implementation. Costs in the centralized processing model are frequently lower than those of the distributed model.
Messaging Bridge	Provides an intermediary to enable communication between messaging systems that are otherwise incompatible because of protocol or format. This intermediary can increase the longevity of your existing system while still allowing interoperability with systems that use a different messaging or eventing technology.
Publisher/Subscriber	Decouples components of an architecture by replacing direct client-to-service or client-to-services communication with communication by using an intermediate message broker or event bus. This design can enable an event-driven approach in your architecture, which couples well with consumption-based billing to avoid overprovisioning.
Queue-Based Load Leveling	Controls the level of incoming requests or tasks by buffering them in a queue and letting the queue processor handle them at a controlled pace. Because load processing is decoupled from the request or task intake, you can use this approach to reduce the need to overprovision resources to handle peak load.
Sharding	Directs load to a specific logical destination to handle the specific request, enabling colocation for optimization. A system that implements shards often benefits from using multiple instances of less expensive compute or storage resources rather than a single more expensive resource. In many cases, this configuration can save you money.
Static Content Hosting	Optimizes the delivery of static content to workload clients by using a hosting platform that's designed for that purpose. Dynamic application hosts are usually more expensive than static hosts because dynamic hosts can run your coded business logic. Using an application platform to deliver static content isn't cost-effective.
Strangler Fig	Provides an approach for systematically replacing the components of a running system with new components, often during a migration or modernization of the system. The goal of this approach is to maximize the use of existing investments in the currently running system while modernizing incrementally. It enables you to perform high-ROI replacements before low-ROI replacements.
Throttling	Imposes limits on the rate or throughput of incoming requests to a resource or component. The limits can inform cost modeling and can

Pattern	Summary
	even be directly tied to the business model of your application. They also put clear upper bounds on utilization, which can be factored into resource sizing.
Valet Key	Grants security-restricted access to a resource without using an intermediary resource to proxy the access. This design offloads processing as an exclusive relationship between the client and the resource without adding a component to directly handle all client requests. The benefit is most dramatic when client requests are frequent or large enough to require significant proxy resources or the proxy does not add value as being part of the request.

Next steps

Review the cloud design patterns that support the other Azure Well-Architected Framework pillars:

- Cloud design patterns that support reliability
- Cloud design patterns that support security
- Cloud design patterns that support operational excellence
- Cloud design patterns that support performance efficiency

Feedback

Was this page helpful? Yes No

Recommendations for creating a culture of financial responsibility

Article • 11/14/2023

Applies to this Azure Well-Architected-Framework Cost Optimization checklist recommendation:

CO:01	**Create a culture of financial responsibility. Regularly train personnel so technical skills remain sharp. Foster creativity and spending accountability in the work environment. Invest in tooling and implementing automation.**

This guide describes the recommendations for creating a culture of financial responsibility in an organization. Creating a culture of financial responsibility is where the workload team is equipped and motivated to make prudent financial decisions. It drives the team to proactively seek out and implement strategies that enhance efficiency and reduce unnecessary expenses. Without this culture, there's often a disconnect between resource utilization and project goals. It can lead to budget overruns and diminished return on investment for the workload.

Definitions

Term	Definition
Financial responsibility	The sense of shared ownership of cost outcomes.
Microsoft learning partner	An organization that meets program requirements to teach training content developed by Microsoft and that employs Microsoft Certified Trainers to deliver content.

Key design strategies

Creating a culture of financial responsibility involves cost transparency, skill development, and clear communication. Be open about budgets and costs with the team. Hold regular workshops to teach cost-saving strategies. Invest in training for skills like budgeting and cloud cost management. Finally, set clear financial goals and encourage open communication for sharing ideas on cost optimization.

Make budgets and costs transparent

Making budgets and costs transparent requires openly communicating financial information about a workload or project with stakeholders. It helps cultivate an environment of trust and accountability, where everyone understands the financial implications of their actions and works collaboratively towards cost optimization. To make costs transparent, consider these strategies:

- *Involve everyone*: Ensure that everyone involved in the project or workload has access to budget information. Include team members, managers, and decision-makers. Sharing the budget information makes everyone feel accountable for their actions and decisions by fostering a sense of ownership.
- *Share budget details*: Consider sharing the allocated budget, cost breakdowns, and financial goals for the workload or project with stakeholders.
- *Share workload costs*: Strive for maximum transparency in the cost of the workload. Consider providing detailed information about the expenses for various aspects of the project, such as infrastructure costs, software licenses, and operational expenses. The more transparent the cost information is, the better it is for facilitating cost optimization efforts.

A benefit of sharing cost details is early detection of overspending. Transparent budgets provide a clear threshold for spending. If a team is approaching or exceeding a budget, it can take corrective actions early to avoid overspending.

Encourage continuous improvement

Encouraging a culture of continuous improvement is about fostering an environment where the workload team is empowered to explore and propose cost-saving measures. It requires open dialog, training, and team-building focused on cost optimization. With this mindset, the workload team is more likely to be proactive in identifying and implementing cost optimization strategies. To encourage continuous improvement, consider these suggestions:

- *Cost workshops*: Conduct workshops or training sessions to help leaders enhance their understanding of financial responsibility and its importance. These workshops can cover topics such as budgeting, cost management, and financial decision-making.

  Evaluate the current financial practices and culture in the organization. Identify areas that require improvement to determine the specific interventions that address those areas. Collaborate with subject matter experts or external consultants, if necessary, to design and develop engaging and informative workshop content.

- *Team-building activities*: Organize team-building activities that focus on financial responsibility. These activities can include group exercises, case studies, or simulations that encourage collaboration and problem-solving in financial matters.

  Foster a culture of knowledge sharing. Encourage the workload team to share its expertise and best practices with colleagues through internal forums, presentations, or mentoring programs.

- *Open dialog*: Create an environment where personnel feel comfortable speaking up about cost optimization ideas. Encourage open discussions, provide a platform for feedback, and welcome alternative suggestions. This environment helps create a culture of continuous improvement and innovation.

- *Empowerment:* Give the workload team the authority and responsibility to evaluate and propose the adoption of new technologies. The workload team should assess the suitability of new tools and technologies for their specific workload. They should consider factors such as compatibility, scalability, ease of integration, and potential cost savings.

Develop skills in-house

Developing skills in-house means investing in training the workload team. The goal is for the team to gain the necessary expertise to optimize costs within a workload.

This effort involves providing training programs, resources, and support to enhance the skills and capabilities of employees in the organization. It allows you to use the existing talent pool and empower employees to take ownership of cost optimization initiatives. Consider following these strategies:

- *Assess skills*: Evaluate the existing skills and knowledge of the workload team to identify areas that need improvement in terms of workload cost optimization.

- *Define training objectives*: Determine the specific skills and expertise that are required to optimize costs within the workload. Include topics such as cloud cost management, resource optimization techniques, budgeting, and data analysis.

- *Provide training and resources*: Offer training programs, workshops, and resources to enhance skills in the identified areas. Include both internal training sessions and external training programs provided by vendors or industry experts.

  Gather or create learning resources such as documentation, tutorials, videos, and case studies to support the training. These resources should provide step-by-step guidance on optimizing costs in the workload.

- *Give hands-on experience*: Provide opportunities for employees to apply their newly acquired skills in real-world scenarios. Assign them to cost optimization projects, or give them responsibilities that allow them to refine their skills.

  Create a workspace or sandbox where employees can explore, practice, and learn new concepts related to cost optimization. Allocate a budget for experimentation to encourage employees to think about the financial effects of their actions.

  Set aside time (a day or a week) for people to explore new technologies, build new tools, and express their creative thinking. Employees are closest to the challenges they face daily. They often find ways to optimize their time when they're given space to think about it.

- *Encourage continuous learning*: Encourage employees to attend conferences, webinars, and industry events, and to participate in online communities and forums. Arrange opportunities for employees to shadow and learn from experienced individuals who have a deep understanding of cost optimization practices. This practical experience can provide valuable insights and guidance.

Communicate financial expectations

Communicating financial expectations to a workload team involves conveying financial goals and establishing open channels for exchanging cost-saving ideas and knowledge. This process includes reshaping organizational values to emphasize financial responsibility, clear goals, and cross-functional communication. Consider these recommendations:

- *Reshape values*: Review and update values and mission statements to include financial responsibility. Ensure that these values are communicated and reinforced throughout the organization, and that they're aligned with the overall business objectives.

  Recognize and reward individuals and teams who demonstrate financial responsibility and contribute to cost optimization efforts. Use performance evaluations, incentives, or other recognition programs.

- *Establish expectations*: Set clear expectations and goals for all stakeholders who are aligned with the mission. Encourage accountability and responsibility for all actions taken. Establish and promote success metrics that are aligned with individual teams' goals.

- *Establish communication channels*: Use email groups, chat platforms, or dedicated collaboration tools as communication channels. Encourage team members to share

their ideas and insights about cost optimization. Promote diverse perspectives. Make sure everyone is aware of the channels and how to access them.

- *Extend communication channels*: Make the communication channels available to the broader organization or department to encourage cross-functional collaboration and knowledge sharing. Identify power users or people in the organization who can act as champions for cost optimization. These individuals can help drive the adoption of cost-saving practices and help others understand the importance of cost optimization.

  Use internal knowledge-sharing sessions, conferences, or industry forums. Employees can learn from others and contribute to the collective knowledge in the organization.

Azure facilitation

Making budgets and costs transparent: For your Azure related budgets and costs, you can create and management budgets and provide read access to make these details available to the broader team. You can also create daily reports and export cost data to make cost data accessible to stakeholders.

Encouraging continuous improvement: Azure regularly updates its services and introduces new features to improve efficiency and cost optimization. You can benefit from the latest advancements in cloud technology and take advantage of automation, advanced deployment strategies, and infrastructure as code (IaaS) practices.

Azure offers Visual Studio subscriptions with credits to try new features and learn new skills. Azure also provides role-based access control (RBAC), which allows read access to billing data through Microsoft Entra Privileged Identity Management.

Azure has a partner ecosystem that includes training partners who can provide expertise and guidance. Organizational leaders can talk to Microsoft and engage Cloud Solution Architects to get help and guidance in creating a culture of financial responsibility.

Developing skills in-house: Azure provides a wide range of learning resources. These resources include documentation, video series, training modules, tutorials, and learning paths. These resources help personnel enhance their skills and knowledge. Azure also offers certifications that validate expertise in Azure services and technologies.

Microsoft offers the Cloud Skills Challenge, which allows individuals to test and improve their Azure skills. It's a fun, free, and interactive skilling program that gives you access to Microsoft skilling resources for your specific solution area. Gain access to Microsoft

learning paths, virtual training days, and a virtual leaderboard to compete with peers in the industry.

Azure publishes blogs, announcements, and marketing materials that provide information on cost optimization and cost-optimized resources. They often contain information about upcoming or preview technology that you can experiment with.

Related links

- Find an Azure partner

Cost Optimization checklist

Refer to the complete set of recommendations.

Cost Optimization checklist

Recommendations for creating a cost model

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Cost Optimization checklist recommendation:

CO:02	**Create and maintain a cost model. A cost model should estimate the initial cost, run rates, and ongoing costs. Negotiate a budget that covers a cost model and has a buffer for unplanned spending.**

This guide describes the best practices for creating a cost model for your workload. A cost model is an estimate that predicts the combined costs of services and their associated expenses. It's foundational for expense forecasting and budget planning. A cost model provides scenario analysis, which allows you to assess the cost implications of potential workload changes. Without a cost model, you risk unforeseen expenses, budget overruns, and missed opportunities for cost optimization.

Definitions

Term	Definition
Chargeback	An accounting model in which you charge departments for their workload usage and receive payments from them.
Cost model	The estimated cost of a workload. It captures all the dimensions of billing, including operations.
Cost meter	A tracking mechanism that you use to measure the usage of resources over time. It tracks usage, such as compute hours, data transfer, and input-output operations. It emits the records that are used to calculate the bill for each resource based on its associated meter.
Showback	An accounting model in which you show departments the cost of their workload usage, and you don't receive payment from them.

Key design strategies

A cost model provides a projection of the workload cost based on existing data. The purpose of a cost model isn't to gain visibility into expenses or control them. The goal is to forecast the predicted expenses, considering all available known factors. From that prediction, you determine the best solution for your workload. The best solution aligns

spending to workload priorities. A cost model enables you to establish a workload budget, ensure alignment with this budget, and allocate funds for cloud resources.

Conduct workload assessment

Conducting a workload assessment involves systematically evaluating and analyzing the workload. A workload assessment helps identify workload characteristics that can inform cost optimization strategies, such as choosing the most suitable discount options based on usage patterns. You need to assess the workload characteristics to determine which available discounts are most suitable for your workload. For example, if your workload has predictable usage patterns, you might consider using a commitment-based model (reservations) to optimize costs. When you assess a workload, consider these recommendations:

- *Analyze key components*: Analyze the key components of your workload, including essential resources such as servers, databases, networks, and licenses. This identification allows for precise cost allocation within the model.
- *Understand characteristics*: Understand the workload's stability, predictability, and sensitivity to external factors like downtime and degraded performance. Such insights help anticipate fluctuating costs based on workload behavior.
- *Understand requirements*: Assess the specific requirements of your workload like performance, scalability, observability, backup, and disaster recovery. Recognizing these requirements ensures the model accounts for all potential expenses.
- *Understand supporting services.* Services that support observability, security, and governance incur costs and play a pivotal role in the operation of your workload. Observability solutions, such as monitoring tools and logging mechanisms, offer insights into workload usage and performance. Robust security measures, like encryption or access controls and regular security audits, safeguard your workload and ensure regulation compliance. Governance practices and policies ensure compliance and efficient resource utilization. Incorporate the expenses for these supporting services into your budget.

  If you include these often-forgotten factors in your budgeting early on, your cost modeling will be thorough, effective, and prevent future unexpected expenses.

Estimate workload costs

Estimating workload costs involves assessing all potential expenditures and savings linked to the workload. It encompasses direct vendor costs, operational maintenance

expenses, billing model choices, and potential savings from customer or enterprise agreements. By evaluating these factors, you can create a robust cost model, enabling precise forecasting and budgeting. To estimate workload costs, consider the following strategies:

Select the best billing model. A billing model determines how the cloud service provider charges for their services. Billing models include consumption-based (pay-as-you-go), commitment-based plans (reservations), and spot pricing. Identify the most suitable and cost-effective billing model by understanding the specific requirements and usage patterns of each model. Each billing model has advantages and disadvantages of cost structure and flexibility. For example, pay-as-you-go might provide flexibility but might be more expensive over time compared to commitment-base plans instances.

Use customer agreements. Cloud service providers offer customer agreements or enterprise agreements for customers. Some agreements offer discounts through available programs or allow you to use your existing licenses to save money. Implement these cost optimization strategies to maximize the value of your resources and reduce your overall expenses.

Estimate license costs. Calculate estimates for license costs to create an accurate cost model. To find the best deal, contact the software provider or the software reseller. If you're an existing customer, use existing licensing benefits and discounts.

Estimate service costs. Cloud service providers provide many services to support your workload. Choose services that help you meet your short-term and long-term cost objectives. For example, you might want to move an on-premises workload to the cloud with minimal changes to your workload. Choose a cloud service that supports your workload goals and provides the greatest return on your investment. Use the cloud platform's cost calculator to estimate your workload's resource costs. These tools help you understand the different cost meters in a resource and the billing model cost implications.

Consider the cost advantages and disadvantages of each service. Service-level objectives (SLOs) and platform features have cost implications. For example, downtime might cost your organization a considerable amount of money. If you invest more money into reaching higher SLOs, you can generate revenue by avoiding downtime and increasing customer satisfaction. Use built-in features as a cost-efficient alternative to building custom features that you need to develop and maintain.

Estimate resiliency costs. To estimate resiliency costs, consider factors such as infrastructure, maintenance, data replication, data storage, disaster recovery, and performance. Consider the specific requirements and goals of your application or system. It could include the required level of resiliency, the desired SLOs, and the

availability goals for each dependency on the critical path. The costs vary based on the cloud services and technologies that you choose.

Estimate operational costs. To estimate the cost of workload maintenance, consider the ongoing operational expenses for monitoring, testing, and maintenance of the infrastructure. These costs include monitoring the performance and health of the infrastructure. It should include monitoring tools and services to help track system metrics, detect issues, and ensure availability.

You should estimate the cost of regular testing activities such as load testing, security testing, and performance testing. These tests are essential for maintaining the integrity of your workload. Include the resources and tools that are required for testing the system's resilience, scalability, and security. You also need to include Regular maintenance tasks, such as applying software patches, updates, and security fixes, are necessary to keep the system up to date. Routine tasks like data backup, system optimization, and configuration management contribute to ongoing maintenance costs.

Develop the cost model

The cost model is an estimate of all costs associated with the workload. These costs include infrastructure, software licenses, personnel, maintenance, and support costs.

Align estimates to cost drivers

Cost drivers are specific factors or variables that influence the overall cost. It includes any factors that have a direct impact on the cost of resources, services, or operations within the workload. These drivers can include variables such as usage volume, the number of customers served, storage capacity.

Assign quantitative values to the identified cost drivers, such as estimating usage volumes or determining the number of customers or resources. Quantify the effect of each cost driver by using methods such as estimating usage volumes or determining the number of customers or resources. Based on the cost categories and drivers, establish mathematical models or formulas that relate the cost drivers to the associated costs. These models can include simple linear relationships or complex calculations, depending on the cost category.

Associate costs with business metrics

Associating costs with business metrics means linking workload expenses to specific business indicators, like cost per customer served or cost per transaction processed. This

practice provides a clearer understanding of how the workload consumes resources. It allows you to anticipate costs related to workload fluctuations and ensures efficient resource utilization based on demand. For example, if you expect the number of customers to grow, you can estimate how much it costs to support those customers.

You should emphasize clear visibility in the workload cost models. While it can make the model more intricate, it also allows for adaptability. Such a flexible cost model aids in scenario analysis, helping to predict expenses tied to workload or business shifts. To estimate the cost associated with each customer, divide the total workload cost by the number of customers. For a precise cost per customer, account for specific resources and services they utilize, like cloud services or software licenses.

Publish the cost model

Document the cost categories, drivers, and mathematical relationships that are used to calculate the costs. Create comprehensive and easily understood documentation for stakeholders. Ensure that the cost model is accessible to all relevant stakeholders. Publish the cost model in a format or on a platform that allows for seamless data exchange and enables efficient collaboration between stakeholders.

Set a budget

The cost model provides a foundation for negotiating your workload budget. The cost model is an estimate. The budget is a reality. Sometimes you have to negotiate to align the two. It's important that everyone understands how the workload supports business objectives. Present the cost model in alignment with business objectives to help clarify the value of the workload.

- *Share the cost model*: When you share the cost model with stakeholders, make sure the estimates are clear. Stakeholders should be able to see the cost distribution, cost variables, and optimization efforts.

- *Modify the cost model to fit the budget*: Stakeholders might not agree to the proposed budget and they might offer a budget that's less than the cost model. It's important that stakeholders know how the budget affects the workload. Create a second cost model that fits the budget and includes a buffer. Explain any functionality loss with the reduced budget.

  The resulting budget should be realistic, but include a buffer for predicted usage changes over the budget period. The cost model helps predict these changes. A budget should also include a small and reasonable buffer for unplanned overages that result from a mistake or an unplanned business change.

- *Set budget caps and quotas*: Define budget caps and quotas to control costs and limit spending. This practice ensures that you don't exceed the allocated budget for your workload. By setting budget caps and quotas, you can monitor and manage your spending effectively.
- *Set budget alerts*: Implement alerts for cost management. Set up alerts to notify you when spending reaches certain thresholds. This practice allows you to take immediate action and make necessary adjustments to stay within the budget. Monitor usage and set alerts to help identify trends, peak usage times, and opportunities for cost optimization.

Use the cost model

A cost model isn't just an analytical tool. It's a decision-making aid. Use the cost model for budgeting, scenario analysis, and resource optimization. To maximize the use of the cost model, consider these strategies:

- *Use the cost model for budgeting*: Use the cost model to project future expenses, allocate funds effectively, and avoid financial pitfalls. Regularly compare actual expenses against the budget and make adjustments if there are deviations.
- *Use cost model for scenario analysis*: Using the cost model for scenario analysis involves considering different scenarios and the associated costs with each one. Scenario analysis can help stakeholders understand the financial implications of business model changes, such as modifications to pricing, product offerings, or revenue streams. Scenario analysis also enables you to anticipate how changes in customer acquisition, retention, or churn rates might affect costs. You can forecast increased expenses and plan for scaling.
- *Use cost model for resource optimization*: Use the cost model to help identify areas where cloud resources are underutilized and make adjustments for significant cost savings. The cost model can also forecast the financial implications of scaling up resources in response to increased customer traffic or processing needs. It also helps compare the costs that are associated with cloud providers' billing models, which allows you to choose the most economical option.

Maintain the cost model

It's important to regularly update the cost model to reflect the latest data, business conditions, and any changes in the external environment. You should engage stakeholders, including product owners and the technical team, in discussions around the cost model to ensure its relevance and alignment with different teams' needs. Run

simulations and review the findings to inform decision-making. Educate all team members on how to use the cost model to foster a culture of data-driven decision-making. Consider the following recommendations:

Track resource usage. Monitor the usage of resources in your workload. Tracking resource usage is critical for adjusting cost models and identifying opportunities for cost optimization. Conduct utilization audits to identify underutilized resources and adjust cost estimates accordingly.

Generate and review forecasts. Utilize usage data to generate forecasts and project the cost of the workload. Update forecasts regularly and view them often. Investigate any forecasts that deviate significantly from the current cost model. When you find an issue, update the cost model accordingly. The definition of a significant deviation from the cost model is different for each workload. The deviation might be due to changes in workload usage patterns, resource requirements, or pricing changes. By using a forecast, you can foresee exceeding your budget and make changes to the budget or workload design.

Update the cost model. Review the cost model periodically to ensure that the workload receives the budget it needs. Use the metrics from the workload in production to inform budget reviews. The potential effect of services or technology changes can create the need for review. As services and technologies evolve, you might need to make changes to the workload design to optimize costs or take advantage of new features. Regular review ensures that the cost model remains aligned with the changing landscape. Review the cost model before and after workload design changes.

Update the cost model whenever you change services. Use the cloud platform's calculator to estimate the cost of the cloud resources that your workload needs. For new workloads, some of the cost variables, such as data transfer and storage amount, can be difficult to estimate. A business target can help you generate estimates. For example, to create a customer-based estimation, divide the daily revenue target ($100,000) by the average purchase per visitor ($100) to get the estimated number of daily visitors that you need to support (1,000).

Azure facilitation

Estimating workload costs and developing a cost model: The Azure pricing overview provides pricing information for all Azure services. It shows a comprehensive view of the costs that are associated with different Azure services.

The Azure pricing calculator is a tool that allows you to estimate the hourly or monthly costs of your workload. Input the services that you plan to use to generate an

estimate of the associated costs. This estimate helps you plan and budget for your Azure usage.

The total cost of ownership (TCO) calculator helps you estimate the cost savings of migrating your workload to Azure. It takes into account factors such as infrastructure, management, and labor costs to provide an estimate of the total cost of ownership. This estimate helps you make informed decisions about the financial aspects of your Azure migration.

Azure Hybrid Benefit is a program that cloud service providers, like Azure, offer. It allows customers to use their own licenses for certain software products on the cloud. Use your own license to take advantage of discounted pricing for using that software on the cloud platform. Sometimes Azure Hybrid Benefit is part of the customer agreement between the cloud service provider and the customer. This agreement outlines the terms and conditions for utilizing the benefit and the eligible software products that are covered under it.

When you extend your existing investments in software licenses to the cloud, you save on costs. Instead of paying the full price for using the software on the cloud, you benefit from the discounted pricing that Azure Hybrid Benefit offers.

Setting a budget: Azure provides tools that allow you to create and manage budgets. Budgets help you proactively inform others about their spending, manage costs, and monitor spending over time. You can set budget thresholds, receive alerts, and track expenses to ensure cost control and optimization.

Maintaining a cost model: Azure automatically provides cost forecasts, which enable you to plan and budget for your Azure usage. These forecasts help you understand the projected costs based on your current usage patterns and allow you to make proactive decisions to optimize costs.

Azure allows you to use tag inheritance to group and allocate costs. Tags are metadata that you can assign to Azure resources. With tag inheritance, you can track and manage costs for different teams or projects within your organization to help with cost allocation and analysis.

Related links

- Measure unit costs
- View and download Azure usage and charges

Cost Optimization checklist

Refer to the complete set of recommendations.

Cost Optimization checklist

Recommendations for collecting and reviewing cost data

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Cost Optimization checklist recommendation:

CO:03	**Collect and review cost data. Data collection should capture daily costs. In cost reports, include incurred costs (metered), prepaid costs (amortized), trends, and forecasts. Stakeholders should regularly review spending against the budget and cost model. Automate alerts to trigger notifications at key thresholds and detect anomalies to indicate deviations from trend baselines.**

This guide describes the recommendations for collecting and reviewing cost data for your workload. Gather cost data to paint a holistic picture of your workload and ensure spending is optimized. Data collection includes all indicators of cost optimization, like billing data, resource utilization, and usage patterns.

Collected data allows you to understand the cost of architecture decisions and business drivers like costs per user or unit. This data gives you a clearer understanding of where money goes and how to optimize spending. Failure to collect and review cost data can lead to budget overruns, no baseline for spending, and a lack of understanding of the financial health of your cloud workloads.

Definitions

Term	Definition
Billing boundary	The scope of what a bill includes.
Chargeback	An accounting model in which you charge departments for their workload usage and receive payments from them.
Resource utilization	The amount of resource capacity a workload uses.
Showback	An accounting model in which you show departments the cost of their workload usage, and you don't receive payment from them.

Key design strategies

Data collection is essential for identifying cost-saving opportunities, accounting (*showback* and *chargeback*), and for efficient resource usage. You must prioritize the collection and review of cost data from all relevant sources. You should centralize the collected data for streamlined analysis and regular review, assign resource owners, and automate alerts where possible.

Collect cost data

Effective cost management of cloud workloads requires a comprehensive grasp of associated expenses, from computing to network usage. Data collected provides a granular view of where and how resources are being consumed. It allows you to identify inefficiencies, make informed decisions about resource allocation, and ultimately optimize costs to ensure you're getting the best value for your investment.

Enable data collection. Data collection should include all sources of workload cost, such as compute, storage, network usage, and any other services or features the workload uses. The data should include invoiced and metered data. Invoiced data is *real*. It reflects actual billed expenses. Metered data is a predictive form of data based on the billing plans for services. While still valuable, daily slices of metered data are considered *good estimates* rather than precise figures. Recognizing the distinctions between invoiced and metered data in these components can provide more accuracy in financial planning and analysis.

Use all available collection methods. To collect cost data, use all available tools and methods at your disposal like service provider's cost monitoring and utilities to monitor workload expenses. While these tools typically offer detailed insights into cost breakdowns, usage trends, and optimization suggestions, they might not capture everything. Understand their default capabilities, like data dictionaries and taxonomies.

Design custom views if they're required for your specific workload. Beyond native tools, if your service provider offers APIs, tap into them to programmatically retrieve cost data. APIs facilitate automated cost reporting and seamlessly integrate with your existing management systems. Remember, the goal is to gather cost details from every possible source. Whether that means pulling data via an API, manually entering costs, or syncing with your financial systems, it's vital to ensure a centralized and comprehensive cost overview.

Centralize cost data. Centralized cost data allows for easier management and analysis of that data. It ensures you have a unified view, through a common data schema, of all workload costs and enables better cost optimization strategies. You need to combine usage data, and the data should flow into a central analytical sink. You can use a cost management tool provided by your cloud provider or integrate the data with third-party

cost management solutions. The goal is to have a low-cost solution that's easily accessible by authorized stakeholders and provides robust data analysis capabilities.

>  **Tradeoff**: Retaining cost data for longer periods enables historical analysis and trend identification. However, storing data can be costly. To minimize cost, store older data as aggregated data points without the granularity of newer data. Also, determine the best retention period based on your analysis needs.

Group data

Grouping data allows you to gain better insights to manage costs effectively. You can break down costs based on different dimensions, such as departments or projects, allowing you to accurately allocate costs to the respective stakeholders. Grouping data promotes transparency, accountability, and cost awareness.

Group cost data into meaningful categories such as resources, services, environments, regions, departments, projects, or teams. For example, understanding the cost breakdown at the resource and service level can help you make informed decisions about resource allocation, scaling, or even decommissioning. When you group cost data by environment, such as production, disaster recovery, or quality assurance, it can help you identify cost discrepancies and optimize resource usage based on the specific needs of each environment. When you group workload data, consider the following recommendations:

- *Collect usage and component data.* Collect detailed information about the usage and cost of each component in your workload. You can analyze costs from different angles and gain insights into the cost breakdown by capturing this data.

- *See different dimensions.* Break down your daily expenses by technical dimensions (for example, resource types or service categories), resource organization dimensions (for example, departments or teams), and business model dimensions (for example, projects or cost centers). This breakdown allows you to analyze costs based on the dimensions that matter the most to your specific problem or scenario.

- *Apply metadata.* Metadata can be used to group data and help generate meaningful cost reports. It enables you to identify areas of high cost and implement accountability measures or cost optimization strategies at the department or project level. Using metadata, you can design a mechanism to group costs in a way that reflects your application's core business model. For example, tagging resources with tenant identifiers instead of shared resources in a

multitenant solution. The ability to pivot cost data based on your application's pricing model can deliver key insights.

Generate cost reports

After collecting cost data, you need to use it to generate cost reports. Cost reports provide visibility into spending and facilitate the analysis of your workload spending patterns. You can identify areas where cost optimization is needed and make informed decisions to optimize your spending. Cost reports enable you to allocate costs to different teams, departments, or projects. This allocation helps in understanding distribution and facilitates accurate chargeback or showback.

Address common scenarios. When generating cost reports for workload costs, you want to be able to address common cost concerns. Gathering data in common concern areas helps ensure that the necessary data sets, such as costs, metrics, and usage, are interpreted cooperatively. Common concern areas include:

- *Granular costs*: Cost reports should provide information on the amount allocated per user and the cost per device.
- *Resource utilization*: Cost reports should help assess if current resources are fully utilized and identify potential savings.
- *Alternative solutions*: Cost reports should compare the costs and potential savings of transitioning to a new solution. They should also evaluate the feasibility of switching to a dynamic solution.
- *Return on investment*: Cost reports should help determine what percentage of revenue goes into system operation. If the system doesn't boost revenue, other ROI metrics should be measured.
- *Spending patterns*: Cost reports should analyze spending patterns to identify trends and patterns in costs over time. Spending patterns help in making informed decisions about cost optimization and budget planning.

Align to accounting standards. Cost reports should accommodate your internal accounting standards. Common systems are showback and chargeback. Showback is about visibility, and chargeback is about accountability.

- *Showback* refers to providing cost visibility throughout an organization without charging individual teams or departments for their cloud costs. You can use cost reports to generate showback statements that showcase the costs incurred by each team or department. For example, the marketing team utilized $15,000, while the

engineering department incurred costs of $25,000 for a combined workload expenditure of $40,000. Showback provides each department with a breakdown of costs, allowing each team to review and optimize their resource consumption. These reports provide transparency and enable stakeholders to understand their usage and associated costs.

- *Chargeback* involves billing internal teams or departments for their respective cloud costs based on their actual usage. Chargeback is dual-faceted. You can charge others and others can charge you based on resource consumption and services rendered. For example, your workload uses centralized security services. For one month, the security team billed you $10,000 for their services. But you charged the sales and marketing departments $7,000 and $8,000, respectively, for using your workload. All chargeback transactions, both credits and debits, are integrated into your centralized cost data sink. Chargeback ensures every expense is accounted for and incorporated into your organization's financial management. It provides a holistic view and promoting optimization of interdepartmental costs.

Provide comprehensive reports. Cost reports should include the cost of cloud services and vendors. The report should include costs incurred (invoiced), prepaid costs (amortized), trends, forecasts, credits, and cost variance. In both showback and chargeback systems, cost reports should include the following elements:

- *Incurred costs*: Incurred costs refer to the actual costs accrued based on metered usage. These costs are calculated based on the consumption of resources or services within a specific billing period.

- *Prepaid costs*: Prepaid costs are expenses paid in advance and are spread out over a specific period of time. These costs are typically amortized or allocated evenly over the duration of the prepaid period.

- *Trends*: Analyzing cost trends involves examining the historical data to identify patterns and changes in spending over time. This analysis helps you understand how costs fluctuate and identify any underlying factors.

- *Forecasts*: Cost forecasts predict future spending based on historical data and trends, allowing you to estimate future costs and plan accordingly. Forecasts can be generated using various techniques such as machine learning algorithms.

- *Credits*: Service providers often provide credits (free utilization) on services. Cost reports should include credit balances and usage to properly understand spending needs.

- *Cost variance*: Cost variance in a cost report refers to the difference between the actual costs incurred and the expected or budgeted costs. It helps you identify deviations from the planned costs and understand the reasons behind them.

Assign resource owners

Each cost item should have a directly responsible individual (DRI) as the *resource owner*. Assigning a resource owner to each cost item ensures clear accountability for the associated costs. It helps identify who is responsible for managing and optimizing the usage and cost of specific resources or services. Resource owners are important for:

- *Cost allocation*: Having a resource owner assigned to each cost item enables accurate cost allocation. Resource owners ensure cost attribution to the appropriate teams, departments, or projects, facilitating financial transparency and budget management.

- *Communication*: Assigning resource owners promotes effective communication and collaboration within a workload team and organization. It facilitates discussions about cost management, encourages sharing of best practices, and enables resource owners to work together to optimize costs collectively.

- *Decision-making*: Resource owners play a crucial role in decision-making related to resource provisioning, scaling, and optimization. They have the necessary insights and ownership to make informed decisions that align with business objectives and cost optimization goals. Resource owners can actively monitor and analyze the costs associated with their resources. They can identify cost-saving opportunities, optimize resource usage, and make decisions to control and reduce costs.

Review cost data

Regularly review spending against the budget and cost model with stakeholders. Regular reviews help in identifying cost trends, outliers, and areas for optimization. It's important to involve stakeholders such as finance teams, operations teams, and decision-makers in these reviews to drive cost optimization initiatives. Reviews ensure that costs are aligned with expectations and allow for adjustments if necessary. Monitor changes in usage patterns, adjust resource allocations as needed, and implement cost-saving measures based on ongoing analysis of cost data.

Analyze cost data

Review the cost data collected from your workload to gain insights into your spending patterns. Reviews can include analyzing resource utilization, identifying cost drivers, and understanding the distribution of costs across different components of your workload. You should also notice increases and decreases in costs, for example, in compute usage and network transfer costs. Look for areas where you can optimize costs without sacrificing performance or functionality. For example, identify underutilized resources, rightsizing instances, or cost-saving features provided by your cloud provider.

Review architectural choices

When examining the architectural decisions of your workload, it's essential to focus on cost implications. Utilizing alternative patterns or cloud-native offerings can lead to significant cost savings. Opting for platform as a service (PaaS) or software as a service (SaaS) over infrastructure as a service (IaaS) can be more economical. With PaaS, not only are infrastructure expenses part of the service's pricing, but the platform also simplifies the provisioning and management of these resources under a unified cost. For instance, deploying a lower tier virtual machine as a jump box might introduce extra costs for storage, server management, and public IP configuration. In contrast, PaaS handles these complexities, offering a consolidated cost that often encompasses enhanced security.

Automate cost alerts

Implementing automated alerts can trigger budget notifications at key thresholds. These alerts can be set up to notify stakeholders and DRIs when costs exceed predefined limits or when there are significant deviations from expected spending patterns. Budget alerts and forecast alerts are two different types of cost alerts used for automating cost alerts.

Use budget alerts. Budget alerts allow you to set a budget amount and define thresholds for actual costs. When the actual costs exceed the specified thresholds, budget alerts are triggered. These alerts help you monitor your spending and notify you when you're approaching or exceeding your budget. Budget alerts are based on the actual costs you accrued. Workload spend tends to vary. You should, at minimum, set alerts for the target budget at the anticipated costs (100 percent), ideal spend (90 percent), and less than ideal spend (110 percent).

Use forecast alerts. Forecast alerts provide advanced notification when your spending trends are likely to exceed your budget. These alerts are generated based on forecasted cost predictions. When the forecasted cost exceeds the set threshold, forecast alerts are triggered. Forecast alerts help you anticipate potential cost overruns so you can take

proactive measures to control your spending. You should set the forecast alert at 110 percent of the target budget.

Use anomaly detection. Anomaly detection helps identify unexpected or abnormal patterns in cost data, allowing you to detect and address cost anomalies promptly. Utilize anomaly detection techniques to identify deviations from trend baselines, such as unexpected spikes or drops in costs, and take appropriate action. You should tune anomaly detection to catch fluctuations that your spending guardrails can't or intermittently don't prevent.

Based on the analysis of cost anomalies, determine the necessary actions to address the situation. Action plans might involve optimizing resource utilization, resizing virtual machines, implementing Azure Policy controls, or adjusting budgets. It's important to align cost control measures with business values and obtain the necessary approvals for budget adjustments.

Implement automated processes to identify and address cost variances in real-time. Options include automatically scaling resources, automating shutdowns, or establishing workflows for investigation and mitigation of cost anomalies. Establish key performance indicators (KPIs) to measure the accuracy of cost forecasts, cost versus budget, the number of unexpected anomalies, and the time to react to forecast alerts. Regularly review forecasts, track variance, and ensure alignment with budget expectations.

>  **Risk**: Automating the collection and review of cost data can save time and effort. However, relying solely on automation might overlook certain cost optimization opportunities that require manual review and analysis. Finding the right balance between automation and manual review is crucial.

Azure facilitation

Collecting and grouping cost data: Azure provides services like Cost analysis and Azure Advisor that help track and analyze your Azure spending and usage. These services capture the necessary data to calculate costs accurately. Use Azure tags to group costs to align with different business units, engineering environments, and cost departments. Tags provide the visibility needed for businesses to manage and allocate costs across different groups.

Generating cost reports: Cost analysis offers customizable reports that provide insights into your incurred costs, prepaid costs, trends, and forecasts. These reports can be tailored to your specific requirements and provide a comprehensive view of your costs.

Reviewing cost data: Microsoft Power BI can help with collecting and reviewing cost data. Power BI provides a comprehensive solution for collecting, reviewing, and analyzing cost data. It enables you to gain insights, track trends, and optimize costs effectively. It integrates with Cost Management and allows you to import cost data into Power BI.

For smaller cost data sets, you can use Usage Details API to get programmatic retrieval of raw, unaggregated cost data that corresponds to your Azure bill.

Reviewing architecture design choices: Azure provides a wide range of PaaS resources. Here are some examples of when you might consider PaaS options:

Task	Use
Host a web server	Azure App Service instead of setting up IIS servers.
Indexing and querying heterogenous data	Azure Cognitive Search instead of ElasticSearch.
Host a database server	Azure offers many SQL and no-SQL options such as Azure SQL Database and Azure Cosmos DB.
Secure access to virtual machine	Azure Bastion instead of virtual machines as jump boxes.
Network security	Azure Firewall instead of virtual network appliances.

Automating alerts: Cost Management enables you to set up automated alerts and actions based on spending thresholds or budgets. These alerts can trigger notifications to stakeholders when costs exceed predefined limits or deviate from expected patterns. You should use Cost analysis to view and respond to cost anomalies. This feature can highlight unexpected spikes or drops in costs, allowing for timely investigation and action.

Related links

- Group and filter options in cost analysis and budgets
- Monitor usage and spending with cost alerts in Cost Management
- Azure billing and cost management budget scenario
- Group and allocate costs using tag inheritance.
- Allocate Azure costs

Cost Optimization checklist

Refer to the complete set of recommendations.

Cost Optimization checklist

Recommendations for setting spending guardrails

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Cost Optimization checklist recommendation:

CO:04	**Set spending guardrails. Guardrails should include release gates, governance policies, resource limits, and access controls. Prioritize platform automation over manual processes.**

This guide describes the recommendations for setting spending guardrails. Spending guardrails are measures to control and manage costs within a specified budget. They help prevent unexpected or excessive spending and promote cost-effective utilization of resources. Without spending guardrails, your workload costs might exceed your budget, leading to unplanned expenses that can strain your financial resources.

Definitions

Term	Definition
Governance policies	A set of rules that enforce compliance and enable auditing of workload resources.
Governance	A set of policies, processes, and controls that help ensure that the workload is managed effectively, securely, and in compliance with organizational and regulatory requirements.
Infrastructure as code (IaC)	A descriptive model for defining and deploying infrastructure, including networks, virtual machines, load balancers, and connection topologies.
Release gate	A condition or checkpoint in a release pipeline that must be satisfied before the deployment can proceed. A release gate helps to ensure that specific criteria are met before software is released.

Key design strategies

Set spending guardrails by implementing measures to control and manage your costs within a specified budget. These measures include governance policies, access controls, release gates, budget thresholds, and alerts. Automation reduces the risk of human error, improves efficiency, and assists the consistent application of spending guardrails. Prioritize platform automation over manual processes. Automation tools and services

the platform provides can streamline resource provisioning, configuration, and management.

Use governance policies

Governance policies can act as spending guardrails on various aspects of resources such as resource types, configurations, tags, location, and data management. Many cloud platforms have a service that automates the enforcement of governance policies. Use automated policies to control resource usage, enforce accountability, and eliminate spending on restricted resource types. Here are some of the policies you should consider enforcing:

- *Restricted resource types*: Policies can specify which types of resources are allowed or disallowed within an organization. For example, an organization might have a policy that restricts the use of certain expensive resource types to control costs.
- *Resource limits*: Set resource limits to controls costs and prevent overprovisioning. Include limits on the number of resources that can be provisioned, the size of resources, and the duration of resource usage in your policy. These limits can help you to prevent excessive spending and optimize resource utilization. For example, resource limits can minimize the effects of an unauthorized account breach related to crypto mining.
- *Defined resource configurations*: Policies can define specific configurations for resources. You can enforce settings on resources that promote cost optimization such as automatic scaling and data archiving
- *Restricted locations*: You can use policies to restrict the deployment of resources to specific regions or locations. Consider restricting locations to avoid costly data transfer fees and to maintain compliance with data sovereignty regulations.
- *Managed data*: Use policies to enforce data management practices that help optimize costs. For example, you can implement policies that require the use of lower-cost storage tiers for less frequently accessed data or policies that define expiration rules for data retention.
- *Enforced metadata*: Establish policies that mandate the use of specific metadata for better tracking and cost allocation. You can also use metadata in your automation or manual review. For example, use metadata to automate resources backups by using a backup tag. A consistent metadata policy helps to align costs with spending guardrails.

- *Limited idle resources*: Use policies to identify idle resources so you can delete or repurpose them. Consider setting policies that automatically shut down instances during the hours they're not in use.

>  **Risk**: If you implement automatic scaling, set a maximum scaling threshold based on testing. Maximum thresholds can help you avoid massive scaling spikes that cause cost overruns, but a threshold that's set too low might negatively affect performance. For more information, see Recommendations for optimizing scaling costs.

Configure access controls

Configure access controls to set restrictions that prevent overspending and to help ensure that only authorized individuals can consume resources. Access controls can help reduce the risk of accidental or unnecessary changes that negatively affect cost optimization. To implement access controls for cost optimization, follow these steps:

1. *Identify necessary control.* Identify the resources and services that need access controls.

2. *Define access policies.* Define access policies based on the principle of least-privilege access, granting users only the necessary permissions to perform their tasks. For example, some users might need only read access, while others might also require write or delete permissions.

3. *Implement authentication.* Implement authentication methods, like username/password, multifactor authentication, or integration with identity providers, to help ensure that only authorized users can access resources.

4. *Use role-based access control (RBAC).* Set up RBAC to assign roles and permissions to users based on their job responsibilities. Using RBAC can help you manage resource access effectively.

5. *Review and update controls.* Regularly review and update access controls to ensure that they align with the changing needs of the organization. Remove unnecessary access permissions and adjust access levels as needed.

Use release gates

Release gates are checkpoints or conditions that must be met before a release or deployment can proceed. Use release gates to help ensure that the release is cost-effective and aligns with optimization goals. Release gates offer a structured approach

to the identification and implementation of cost-saving measures. To implement release gates for workload cost optimization, consider the following steps:

1. *Establish release gate criteria.* Establish the conditions or criteria that must be met before resources are released or deployed. Include factors such as spending limits, resource utilization thresholds, or project milestones.

2. *Incorporate release gates.* Incorporate the release gates into the deployment pipeline. You can use automation tools or custom scripts to ensure that resource deployments are subject to the defined criteria.

3. *Monitor spending.* Continuously monitor spending and resource usage against the defined criteria. If the organization exceeds spending thresholds, the release gates should prevent further deployments until the issue is addressed.

Configure cost alerts

It's important to set alerts for budgets, cost anomalies, and commitment-based plan utilization to optimize costs. These alerts provide visibility into your cloud spending and enable proactive cost management. Be careful to manage notification recipients for alerts and keep the recipient list up to date with current responsibilities and access. Some alerts that you might create to optimize costs include:

- *Budget alerts*: Set alerts on budgets to track your spending against predefined thresholds. You can monitor your costs and receive notifications when you approach or exceed the budgeted amount by creating a monthly budget, billing account, or resource group. Budget alerts help you to stay informed on your spending and take preventative actions to control costs.

- *Cost anomaly alerts*: Anomaly alerts notify you about unexpected cost variations that might indicate inefficiencies or abnormal spending patterns. You can configure these alerts to identify anomalies in the actual or forecasted costs. Use cost anomaly alerts to investigate the underlying cause of a cost variation and take corrective actions when necessary.

- *Commitment-based plan utilization alerts*: Implement commitment-based plan utilization alerts to monitor your plan usage. If you have commitment-based plans, setting alerts on plan utilization can help you effectively manage and maximize the value of these commitments. You can configure these alerts to notify stakeholders if the utilization of commitment-based resources drops below a desired threshold. Optimize your commitment-based resources and ensure that you use the benefits of your commitments.

Use IaC

Infrastructure as code is the practice of managing and provisioning infrastructure resources by using code, typically in the form of configuration files. Implement this strategy to define and automate the deployment and configuration of infrastructure resources, such as virtual machines, networks, and storage, by using code-based templates.

IaC strategies provide a structured and repeatable approach to managing and controlling infrastructure resources. IaC can help you to deploy resources as-needed, delete resources without running them continuously, and optimize costs by ensuring you deploy and configure resources according to predefined rules. Follow these steps to use IaC for cost optimization:

1. *Create IaC templates.* Create a code-based template language to define your infrastructure resources and their configurations. These templates let you specify the desired state of your infrastructure resources in a declarative manner. Implement best practices for cost optimization in your infrastructure code. Consider right-sizing your resources by using reserved instances or savings plans. Use cost-effective storage options and apply resource metadata for cost allocation and tracking.

2. *Store templates.* Store IaC templates in a version control system to track changes and manage different versions. You can use version control to maintain a history of your infrastructure configurations and foster collaboration among team members.

3. *Use parameters.* Use parameters in your templates to make them reusable and configurable. By using parameters, you can easily customize your infrastructure deployments for different environments or scenarios.

4. *Use ephemeral environments.* Use ephemeral environments for development, testing, and staging purposes to optimize costs. Ephemeral environments should only be run when necessary. Create these environments by using IaC tools and delete the environment when you're finished.

5. *Use IaC tools.* Use IaC tools and frameworks to automate the deployment and configuration of your infrastructure resources. Use automation to consistently and reliably deploy resources according to your defined policies.

6. *Monitor deployed resources.* Regularly monitor your resources and their costs to ensure compliance with your spending policies. Use monitoring and alerting tools to identify any deviations from the defined guardrails and take corrective actions

as needed. Check for unused resources and delete them, preferably with automation.

Azure facilitation

Using governance policies: Use Azure Policy to define and enforce governance policies that align with your cost optimization goals. You can use Azure Policy to set rules on management groups, subscriptions, and resource groups. These policies can regulate resource provisioning, usage limits, and cost allocation. Use policies to promote rightsizing of resources, identify and eliminate idle or underutilized resources, and encourage the use of cost-effective services and architectures.

Azure allows you to set limits or quotas to prevent unexpected costs. You can define limits on the number of resources that can be provisioned, in addition to the size and duration of resource usage. Set these limits to help prevent overprovisioning and to control costs.

- *Identify underused or idle resources.* Use Azure Advisor to optimize and reduce your overall Azure costs by identifying idle and under-utilized resources. Receive cost recommendations from the *cost* section in the *advisor* dashboard.
- *Add resource metadata.* Use Azure governance to implement resource tagging and categorization. Tag resources using relevant metadata to track and allocate costs to different departments, projects, or cost centers. Visibility into cost attribution can help you identify areas of high spending, optimize resource allocation, and facilitate better cost management.

Configuring access controls: Use Azure RBAC to manage access to resources. You can use RBAC to grant permissions to users, groups, or applications based on their roles. Implement RBAC to help ensure that only authorized users have access to resources, reducing the risk of unauthorized resource usage and potential cost implications.

Using release gates: Use Azure Pipelines release management to define and enforce your release gates. You can set up manual or automated checkpoints to help ensure that you meet specific criteria, such as security checks, compliance requirements, and cost thresholds.

Using infrastructure as code. You can use Azure tools and services to deploy and manage infrastructure resources by using code. By using tools like Azure Resource Manager (ARM) templates, Azure Bicep, and Azure DevOps, you can define and deploy your infrastructure resources in a declarative manner. Azure has Bicep, Azure Resource Manager, and Terraform templates for every Azure resource.

Use Azure Pipelines or other continuous integration and continuous delivery (CI/CD) tools to automate the build, test, and deployment processes. Teams can use pipelines to define a series of steps and actions that run automatically whenever changes are made to the codebase. Automate these processes to reduce manual effort, ensure consistency, and accelerate the delivery of software.

Consider using lower-cost resources for your ephemeral or nonproduction environments to optimize costs. Azure provides various pricing tiers for resources. Azure DevTest Labs pricing and Azure Reservations are cost-saving methods that you can explore for ephemeral environments.

Git repositories, such as Azure Repos and GitHub, provide version control capabilities for managing code and infrastructure configurations. Teams and developers can use automated repositories to collaborate, track changes, and maintain a history of their codebase.

Azure Deployment Environments empowers development teams to quickly and easily create app infrastructure by using project-based templates that establish consistency and best practices while maximizing security. On-demand access to secure environments accelerates the stages of the software development lifecycle in a compliant and cost-efficient way.

The Azure Developer CLI is an open-source tool that accelerates the time it takes for you to get your application from a local development environment to Azure. The Azure Developer CLI offers developer-friendly commands that map to key stages in your workflow, whether you're working in the terminal, an integrated development environment (IDE), or CI/CD.

Configuring cost alerts: Use Microsoft Cost Management to optimize costs and enforce spending guardrails. You can use cost management features to set budgets and alerts, visualize cost information by using tools like Power BI, and analyze cost patterns and performance bottlenecks.

Organizational alignment

Central teams should use the Cloud Adoption Framework guidance to set up spending guardrails across the organization so workload teams understand what the central team can offer.

For more information, see Cost Management policy compliance processes and Develop cost governance policy statements. We encourage the organization to Adopt policy-driven guardrails for implementation.

Related links

- Assign access to Cost Management data
- Cost Management tools in Azure
- Create and manage budgets
- Identify anomalies and unexpected changes in cost
- Monitor usage and spending with cost alerts

Cost Optimization checklist

Refer to the complete set of recommendations.

Cost Optimization checklist

Recommendations for getting the best rates from providers

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Cost Optimization checklist recommendation:

CO:05	**Get the best rates from providers. You should find and use the best rates for cloud resources and licenses. Regularly review cost savings. Cost reviews should include regional pricing, pricing tiers, pricing models (consumption or commitment-based), license portability, corporate purchasing plans, and price sheets.**

This guide describes the recommendations for getting the best rates from providers for your workload. Getting the best rates is the practice of finding and securing the most cost-efficient pricing options for cloud and software resources without modifying architecture, resources, or functionality. By optimizing rates, you can reduce cloud costs without changing the workload. A small rate reduction on services you use a lot provides significant cost savings. Without rate optimization, you end up paying more for your resources, services, and licenses than necessary.

Definitions

Term	Definition
Consumption-based billing model	A pricing model where you're charged based on the actual usage of the service. Examples include the number of virtual machines deployed, the amount of storage used, and the amount of data transferred.
Commitment-based billing model	A pricing model where you reserve and pay for a specific amount of usage in advance and can often get a discounted rate compared to consumption pricing.
Rate	The unit price for using a service or license.

Key design strategies

Getting the best rates requires actively searching for the most cost-effective pricing models for all workload components. It takes into consideration price differences across regions, the benefits of different billing models, such as consumption (pay-as-you-go) versus commitment-based billing. It also involves software licenses and corporate discount plans.

To get the best rates on the resources and licenses in your workload, you should start with identifying and reducing costs in the most expensive areas. Evaluate the discounts available from providers, and choose the right discounts based on workload needs. Regularly check for discounts and reduce licensing fees where possible. Determine if it's more cost effective to build or buy new workload solutions.

Create an ordered list of expenses

Understanding the workload is the first step to finding and using the best rates on infrastructure, resources, licenses, and third-party services. It prepares you to make informed decisions and implement cost optimization strategies that are specific to the needs of the workload.

Here are actions that you can take to understand the workload rates:

- *Take inventory.* List all the components of your workload, including infrastructure, cloud resources, licenses, third-party services, and any other expenses related to the workload.
- *Understand spending.* Gain a clear understanding of the current spending for each item in the inventory list. Identify what you're paying for and where most of your expenses lie.
- *Create an ordered list of workload expenses.* List the most expensive components and work your way down to the least expensive. This exercise helps you prioritize your optimization efforts and focus on the areas that have the highest effects on cost.

Determine the right billing model

For your billing model, you choose between consumption (pay-as-you-go) and commitment-based billing models. You base the selection of consumption versus commitment-based pricing on the predictability, duration, and usage consistency of workload components. When you make this decision, you must collaborate with development and purchasing teams to evaluate resource needs, usage patterns, and potential cost optimization ideas.

Selecting the right billing model is crucial for cost-effectiveness. It helps align the workload with business objectives and get the best rates for the specific requirements of a workload. To determine the right billing model, consider the following strategies:

Understand the consumption-based billing model

Consumption-based billing model (pay-as-you-go) is a flexible pricing model that allows you to pay for services as you use them. Cost variables for consumption pricing include how long a resource is running. Service meters have various billing increments, such as per hour or per second. This model provides flexibility and cost control, because you pay for only what you consume.

The consumption-based billing model is best suited for the following scenarios:

- *Variable workload*: A variable workload has unpredictable spikes or seasonal variations in usage. Consumption-based billing allows you to scale resources up or down to meet the fluctuations in demand. It helps you to provide the required performance and not overpay during times of low usage.
- *Preproduction environments*: Consumption-based billing is preferred for development and test environments that are ephemeral. It offers the advantage of paying only during the project. Ensure that you provide resources aligned with the development effort. Resources cost less when development is scaled down.
- *Short-term projects*: Short-term projects often have specific resource requirements. Consumption-based billing allows you to pay for the resources only during the project.

>  **Tradeoff**: Many on-premises environments are always on and always available. Being intentional about services might lower rates, but you must account for some creation time and operational overhead.

Understand the commitment-based billing model

Commitment-based pricing allows you to reserve a specific amount for a specific duration and pay for it in advance. By reserving the usage up front, you can get a discounted rate compared to consumption-based billing.

The amount you save with commitment-based pricing depends on factors such as the duration of the reservation, the reserved capacity, and the service. Commitment-based pricing is best suited for the following scenarios:

- *Predictable workloads*: If your workload has a consistent usage pattern, you can commit to a certain capacity over time and get a significant discount over consumption-based billing. Those instances incur charges whether you use them or not.
- *Production environments*: Commitment-based billing is suitable for production environments where you have a good understanding of the workload's resource

needs.

- *Long-term projects*: Commitment-based billing can be cost-effective for projects that have long-term resource requirements, even if they aren't highly predictable.

Discuss options with the workload team

To ensure effective optimization of workload costs, the development team (or architect) and the purchasing team must work together. Combining their expertise enables you to identify opportunities to optimize costs and make informed decisions.

Here's a suggested process for collaborating on rate reduction efforts:

1. *Identify opportunities for cost optimization*: Together, the teams should identify potential areas for cost optimization, such as infrastructure, cloud resources, licenses, and third-party services. Consider factors like usage patterns, scalability, and workload requirements.

2. *Assess resource requirements*: Determine the resources needed to support the component or workload. Consider factors such as infrastructure, maintenance, and ongoing support. Understanding these requirements can help you gauge the long-term commitment involved.

3. *Evaluate options*: Assess your options for cost optimization, such as pay-as-you-go versus commitment-based plans. Evaluate the pros and cons of each option in terms of cost savings and effect on performance. Evaluate the performance tiers in each service and the pricing differences between them.

Determine component permanence

It's important to assess how long you need a particular component to determine if committing to a commitment-based plan makes sense. If the expected usage duration is less than a year, don't commit to a commitment-based plan. Consider the flexibility of pay-as-you-go options for shorter-term requirements.

To determine the duration of component usage, you can follow this process:

1. *Gather usage data*: Collect data on the historical usage of the component or workload. This data can include how long the component has been in operation and the frequency of usage.

2. *Analyze usage patterns*: Analyze the collected usage data to identify patterns and trends. Look for consistent usage over a specific period of time or recurring usage

patterns. This analysis helps you understand the typical duration of component usage.

3. *Consider future requirements*: Consider any future requirements or changes in your component or workload. Evaluate whether any upcoming changes might affect its usage duration.

4. *Assess business needs*: Evaluate the business needs and objectives associated with the component or workload. Consider factors such as project timelines, budget constraints, and the overall strategy of your organization.

   Anticipating future developments can help you assess the long-term commitment required and whether it aligns with your objectives. This assessment helps you determine the appropriate duration for component usage.

Determine usage consistency

When you're considering a commitment-based plan, commit to the maximum consistent usage of a component. By committing to the maximum consistent usage, you can maximize the potential savings and cost optimization. However, there are a few factors to consider:

- *Usage patterns*: Analyze the historical usage patterns of the component. If the usage is consistently high and stable, committing to the maximum consistent usage makes sense. But if the usage is highly variable or unpredictable, committing to the maximum consistent usage might not be feasible or cost-effective.

- *Flexibility and scalability*: Consider the flexibility and scalability of the component. If the component can easily scale up or down based on demand, it might be more suitable to opt for flexible pricing models that allow you to adjust resources dynamically. This way, you can align your costs with the actual usage of the component.

- *Engagement with the provider*: Communicate with the provider to gather information about its plans, roadmap, and commitment to the component or workload. This dialog provides valuable insights into the provider's long-term vision and commitment level.

- *Cost analysis*: Perform a cost analysis to assess whether the potential savings of committing to a higher usage level outweighs the risks of not fully utilizing the commitment.

Select the right commitment-based plan

Strategic usage of commitment-based plan can significantly minimize costs for applicable resources. It allows you to effectively plan and allocate resources. To select the right commitment-based plan, consider the following strategies:

- *Choose an appropriate commitment-based plan*: Select a commitment-based plan that covers the minimum capacity that the workload requires. Starting with the minimum commitment gives you flexibility while you still benefit from cost savings.

  Having a clear understanding of the workload's minimum capacity requirements before you commit to a commitment-based plan minimizes risk and ensures that you optimize your savings. However, there are exceptions. A commitment that requires minimal upfront costs has a lower risk. The lower the commitment risk, the quicker you can commit to a commitment-based plan. As the cost and risk of a commitment grow, you need to understand your minimum consistent usage for each component you're committing to.

- *Increment commitments*: As the capacity of your workload grows, gradually increase your commitments. Start small and scale up. Increment scaling up based on the workload's actual usage.

- *Renegotiate and consolidate*: Regularly renegotiate and normalize commitment-based plans to align their ending time. This alignment allows you to consolidate them into a single line item on your bill, so it's easier to manage and optimize costs.

- *Eliminate underutilization*: You need to evaluate and optimize commitment-based contracts to ensure they deliver their full potential value. Regularly review and analyze your charges and usage data. Understand the breakdown between actual cost and amortized costs and reconcile the data to ensure accurate billing.

  Monitor utilization. Keep an eye on how much you're using your commitment-based plans. Set up alerts to tell you if you're not using all of your reserved resources. Check how you're using them over time and get rid of any you're not using. Make sure you're using the right size of virtual machines to get the most out of your plan. You can also adjust the sizes to fit what you already paid for.

  Modify the commitment-based plan. Consider changing the scope of the reservation to share, allowing it to apply more broadly across your resources. It can help increase utilization and maximize savings. If you find underused commitment-based plans, try exchanging unused quantity or canceling and refunding plans.

Evaluate and commit to available discounts

Assess and analyze the potential discounts that can be applied to a specific workload. This process helps you identify opportunities for cost reduction and optimize the expenditure associated with the workload. It also helps you allocate resources more efficiently.

Try these tasks:

- *Ask about trial offers*: Use a provider's trial periods or negotiate free or reduced rates to execute proofs of concept. This approach allows you to try out the services or products with limited financial risk, so you can assess their suitability for your workload before you commit to a purchase. Remember to review the terms and conditions of any trial periods or negotiated agreements.

- *Review provider offerings*: Understand the discounts and pricing models that providers offer. Explore volume-based discounts, promotional offers, or discounts for long-term commitments. Discuss the available options that can meet the variability and flexibility requirements of your workload. Include information about different pricing models, scaling options, or commitment-based agreements.

- *Analyze usage and consumption*: Assess the workload's usage and consumption patterns to determine if the workload meets the eligibility criteria for specific discount programs. This analysis helps you identify the most suitable discounts for your workload.

- *Evaluate contract terms*: Review the terms and conditions of existing contracts or agreements to identify any potential discount options. Consider the duration of the commitment, renewal terms, and the possibility of negotiating better rates.

- *Communicate with providers*: Know the actual and anticipated usage of a workload when you discuss discounts. Let the providers know what environment the discussion is about. For example, you can often get discounts on preproduction environments. Ask providers to discuss available discount options, such as product bundling. Ask specific questions about discount programs, eligibility criteria, and any negotiation possibilities.

- *Understand reseller options*: Consider engaging with resellers who can provide extra insights into available discounts or offer alternative pricing models. Resellers might have access to specialized programs or discounts that can benefit your workload.

Committing to the right discount options is where you act on your evaluation. You're equipped with the available options. You communicated your needs and workload data

to the various providers. Now you need to lock in the discounted rates for a defined period, which can result in significant cost savings compared to pay-as-you-go pricing.

Decide whether to build or buy a solution

Building a solution in-house allows for granular control over the features and configuration. This control can help you eliminate unnecessary functionality and optimize rates. However, building a solution in-house requires significant upfront investment in development time and maintenance.

When you buy a solution, such as from a marketplace, it offers quicker deployment with potentially lower upfront costs. But buying a solution might involve ongoing subscription or licensing fees.

Here are key considerations when you're deciding whether to build or buy a solution:

- *Control and customization*: Assess the specific functionality that you need for your product or solution. Determine whether buying a solution meets your requirements or whether building allows for the customization and flexibility that provide better rates.

  Building a solution offers greater control over component selection and configuration. You can add customization to fit business needs and minimize unneeded features that might incur charges. Buying solutions provides preconfigured options with limited customization capabilities.

- *Time to market*: Assess the urgency and time constraints for deploying the workload component or solution. Building a solution in-house might take longer because of development and testing, whereas buying a solution allows for quicker deployment.

- *Technical expertise*: Building might require greater technical expertise to ensure proper configuration and maintenance over time. A custom solution requires effort up front and over time. Buying a solution is often more user-friendly and requires less technical knowledge.

- *Cost*: Evaluate the total cost of building a solution, including development resources, infrastructure, ongoing maintenance, and support. Compare the cost of building a solution with the cost of buying a solution. Include any support plans, licensing, or subscription fees. Buying a solution might provide more predictable pricing and potential discounts due to economies of scale.

- *Support and updates*: Consider the availability of support and updates for both building and buying. Assess the level of technical expertise required for each option and the ease of accessing support resources.

  Updates for custom solutions add costs by requiring separate environments, testing, and backups. For purchased solutions, research the reputation and track record of the marketplace providers. Consider factors such as provider reliability, customer reviews, and the provided level of support.

  Also consider the billing cycles. For example, subscription billing cycles are incentivized to maintain the quality of the solution over time. One-time purchases don't have the same cost incentive to maintain the solution.

Optimize licensing costs

Optimizing licensing costs means using various licensing programs and options to minimize expenses while maximizing value. This approach helps ensure you get the best rates from providers, preventing overpayment for software and services. It's important to review the license associated with its design, build, and deployment phases. This review should encompass tools used in its software development, security, monitoring, and design components. These licensing programs might include options for:

- *Hybrid use and bundling*: In addition to exploring licensing programs, consider using hybrid use and bundling options. These programs can provide extra cost savings by optimizing licensing for both on-premises and cloud environments.

- *Negotiations*: Don't hesitate to negotiate with your provider to secure better licensing terms. Negotiations can often lead to more favorable pricing and discounts.

- *Dev/test pricing*: Take advantage of dev/test pricing options that your provider offers. These programs typically provide discounted rates for nonproduction environments, so you can save costs during development and testing phases.

- *Volume discounts*: As your usage increases, you might become eligible for volume discounts. Cloud service providers often offer discounted rates based on the scale of usage, so it's important to monitor your usage and explore opportunities for cost optimization.

- *Existing enterprise agreements*: Check your existing enterprise agreements to see if any licensing benefits or cost-saving opportunities are available. Your procurement department or license reseller can provide valuable insights in this area.

Azure facilitation

Microsoft Cost Management: Azure provides tools and features for managing and optimizing costs, such as Microsoft Cost Management. These tools allow you to track and analyze your cloud spending, set budgets, get cost alerts, and access detailed cost reports.

Azure reservations and Azure savings plans: Reservations and savings plans allow you to commit to using specific resources for a term and get significant discounts on Azure services. Here are the details:

- Azure reservations help you save money by committing to one-year or three-year plans for multiple products. Committing allows you to get a discount on the resources that you use.

  Reservations can significantly reduce your resource costs from pay-as-you-go prices. Reservations provide a billing discount and don't affect the runtime state of your resources. After you purchase a reservation, the discount automatically applies to matching resources.

  You should use reserved instances when you don't expect certain services, products, and locations to change over time. We highly recommend that you begin with a reservation for optimal cost savings.

- An Azure savings plan for compute is a flexible pricing model. It provides savings off pay-as-you-go pricing when you commit to spending a fixed hourly amount on compute services for one or three years.

  Committing to a savings plan allows you to get discounts, up to the hourly commitment amount, on the resources that you use. Savings plan commitments are priced in US dollars for Microsoft Customer Agreement and Cloud Solution Provider customers, and in local currency for Enterprise Agreement customers. Savings plan discounts vary by meter and by commitment term (one year or three years), not commitment amount.

  Savings plans provide a billing discount and don't affect the runtime state of your resources. You should use Azure savings plans for more flexibility in covering diverse compute expenses by committing to specific hourly spending.

Eliminating unused reservations and savings plans: To eliminate unused reservations and savings plans, you can use the Microsoft Cost Management and Billing tools. They provide insights into your reservation and savings plan usage, allowing you to identify

any unused or underutilized commitments and make adjustments accordingly. Utilization can be viewed in the Azure portal under the Reservations section.

Azure dev/test: Azure dev/test is an offer that comes with Visual Studio subscription benefits. With this offer, you get some Azure monthly credits to try various Azure services at no cost. Credit amounts vary by subscription level. You can also benefit from discounted Azure dev/test rates for various Azure services, which enable cost-efficient development and testing.

Azure services: Many Azure services offer both consumption and commitment-based billing models. You can switch to better align to your usage, potentially without sacrificing functionality.

Azure Hybrid Benefit: With Azure Hybrid Benefit, you can reduce the overall cost of ownership by using your existing on-premises licenses to cover the cost of running resources in Azure. This benefit applies to both Windows and Linux virtual machines, along with SQL Server workloads. To take advantage of Azure Hybrid Benefit, you need to ensure that your licenses are eligible and meet the requirements.

License Mobility: Azure supports License Mobility. You can bring your own licenses for certain software products and apply them to Azure resources. This ability can help reduce licensing costs and simplify license management.

Licensing agreements: Microsoft offers commitment-based and transactional licensing options for organizations that want to purchase Microsoft cloud services subscriptions, on-premises software licenses, or Software Assurance. Use these offers for your workload as applicable. Microsoft offers various volume licensing programs and agreements based on your workload's needs, including:

- Microsoft Enterprise Agreement
- Microsoft Customer Agreement
- Microsoft Open Value program
- Microsoft Products and Services Agreement

For more information, see the Microsoft licensing resources.

Azure spot instances: Azure spot instances provide access to unused Azure compute capacity at discounted prices. By using spot instances, you can save money on workloads that are flexible and can handle interruptions.

Related links

- Azure reservations

- Azure savings plan
- Azure dev/test
- Azure Hybrid Benefit
- License Mobility
- Software Assurance
- Microsoft Enterprise Agreement
- Microsoft Customer Agreement
- Microsoft Open Value program
- Microsoft Products and Services Agreement
- Microsoft licensing resources

Cost Optimization checklist

Refer to the complete set of recommendations.

Cost Optimization checklist

Recommendations for aligning usage to billing increments

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Cost Optimization checklist recommendation:

CO:06	**Align usage to billing increments. You should understand billing increments (meters) and align resource usage to those increments. Modify the service to align with billing increments, or modify resource usage to align with billing increments. Consider using a proof of concept to validate billing knowledge and design choices for major cost drivers and to reveal ways to align billing and resource usage.**

This guide describes the recommendations for aligning resource usage to billing increments. Resources are billed at specific increments, such as per hour or per instance. To optimize costs, you need to align your usage to those increments. You must either adapt a resource to your workload usage or adapt your workload to the resource billing increments, also called *meters*. Implement the following guidance so you can ensure that your workload derives the maximum value from each resource. If you fail to align billing and design, you might incur unnecessary charges.

Definitions

Term	Definition
Billing increment	A usage amount that incurs a cost (*meter*), such as a unit of time, number of instances, or size of data.
Billing factor	The type of usage that incurs costs, such as time, storage amount, data transfer amount, or number of requests.

Key design strategies

Aligning resource usage to billing increments is about making sure that your resource consumption closely matches the intervals or quantities that you're charged for. For instance, if a service charges by the hour but you use it only for a fraction of that time, you can adjust operations to maximize the use of that hour.

To save money, ensure that you understand how you're billed for a service. You need to understand specific increments like hourly rates, per gigabyte charges, or per request costs. Adjust the service's configuration or how you consume the service to fit the billing

increments and ensure that you don't incur unnecessary costs. Evaluate your workload's specific needs and understand how you're billed for various resources. Based on your findings, adjust the usage or the resource to optimize costs.

Determine billing factors

Billing factors differ among services. Billing factors include the instance number, time, transaction rate, and transaction size. They also include availability zone, location, storage amount, ingress data, and egress data. Familiarize yourself with the pricing thresholds of the services that you use. You can align your usage to maximize the value of the resource and only run incur charges when necessary.

Here are some common billing factors:

- *Runtime:* The runtime refers to the duration that a resource actively runs or is utilized. Runtime is typically measured in hours, days, or months. The runtime helps you analyze the cost implications of resource usage over time. It's important for cost optimization because you can identify opportunities to minimize resource usage and associated costs.
- *Data transfer:* Data transfer refers to the movement of data into and out of a resource. Data transfer costs can vary based on the volume of data. Understand data transfer costs, so you can optimize data transfer patterns, select appropriate network configurations, and minimize costs associated with data movement.
- *Specialized services:* Specialized services are services or features that you use with other resources. These services can include specialized databases, AI services, or other advanced capabilities. Evaluate the cost implications of specialized services because they might have separate pricing models or incur extra charges.
- *Virtual CPUs (vCPUs):* The utilization of vCPUs within a resource is the vCPU usage. Resources such as virtual machines are often billed based on the number of vCPUs allocated to them. Monitor and optimize vCPU usage, so you can ensure efficient utilization of resources and minimize unnecessary costs.
- *Uptime guarantees:* Uptime guarantees refer to the service-level agreements (SLAs) that cloud providers offer on the availability and reliability of their services. Uptime guarantees aren't directly related to billing, but they're important to consider when you want to optimize costs. Higher uptime guarantees can coincide with higher costs. Evaluate the tradeoff between the cost and the service availability.

Determine billing increments

Billing increments determine how resource usage is measured and billed. For each billing factor, there's a billing increment. Familiarize yourself with the billing increments

of each service, so you can align resource usage to these billing increments.

Here are some common types of billing increments:

- *Time:** Resources are billed based on the duration of usage, such as per second, hour, or day.
- *Per request:* Some resources, particularly in serverless or event-driven architectures, are billed based on the number of requests or invocations. Minimize unnecessary requests and optimize the design of applications to reduce the number of billable requests.
- *Data transfer increments:* Data transfer costs are measured in increments, such as gigabytes (GB) or terabytes (TB).
- *Storage increments:* Storage costs are often measured in increments, such as GB or TB.

Map usage to billing increments

Mapping usage to billing increments is an exercise to identify where resource consumption doesn't align with the billing increments. This mapping involves analyzing resource usage against billing increments in each billing factor to spot inefficiencies. At this step, you're only identifying areas where usage and billing increment aren't aligned. Later, you implement the changes. Consider the following guidance when mapping usage to billing increments:

- *Create an inventory of resources*. List the resources in the workload, such as compute, storage, and networking.
- *Understand usage patterns*. Use monitoring tools or past-usage data to identify the resource consumption patterns for the workload. Note periods of high and low usage.
- *Use pricing calculators*. Input the information that you gather into an online pricing calculator to get a detailed breakdown of costs, segmented by billing factors and increments.
- *Analyze billing increments*. If the calculator provides billing granularity for each component, align your actual or expected usage with the billing increments (hourly, daily, or per request).
- *Simulate scenarios*. Use the pricing calculator to simulate usage scenarios in order to understand how resource usage affects costs.

Consider building a proof of concept (POC)

A proof of concept is a concrete way to validate your understanding of billing factors and billing increments. A POC helps you see the effects of design decisions on cost. It can help you refine your workload design to align with billing increments. A POC is important for leading cost drivers, such as the application platform and resources that scale.

If you're unsure about your billing knowledge or want to gain more confidence in understanding cost implications, a POC can provide a hands-on experience. You can validate your assumptions and test various scenarios to ensure that you have a clear understanding of the billing aspects. Consider the following guidance when you build a POC for cost optimization:

Define POC scope: Clearly define the scope of the POC, including the specific workload or application that you want to optimize for cost and the resources involved. Include factors such as usage time, usage patterns, per instance charges, data transfer, storage, compute, and any other cost-driving components. Consider billing increments when you delineate the scope to ensure that cost factors are thoroughly addressed.

Emulate production: Design the POC to emulate the production environment, ensuring realistic cost estimations. You should evaluate cost drivers, such as the effect of scalability, operational decisions (stopping and starting resources), and storage costs. Align the POC design with billing threshold knowledge to ensure that the simulated environment accurately reflects potential cost scenarios.

Limit POC duration: Limit the lifespan of the POC, so you can gather conclusive evidence, but you don't incur unnecessary costs. Extend the POC slightly beyond a billing threshold to guarantee a comprehensive understanding of costs. For instance, if a resource is billed hourly, the POC might run for just over an hour or however long it takes to capture how costs accrue at the threshold. After you have the corroborating evidence, you can confidently make decisions based on your findings. When the POC provides a clear picture of billing implications, use the findings to make informed financial decisions for the actual environment.

Align usage to maximize resource value

Aligning usage to maximize resource value involves implementing the changes identified in the mapping exercise to realign resource usage with billing increments. This step is about making adjustments to how resources are consumed. There are two primary options for aligning usage to billing increments:

Modify the service. Modifying the services means using different configurations, service tiers, or services to align the workload to billing thresholds. For example, your workload

might move 5 TB of data daily, but you're charged in 4-TB increments. You can find a different service tier or configuration, so you can transfer the data at a cheaper or faster rate.

Modify usage. Modifying usage is about redesigning the usage pattern workload to align with a billing increment. For example, you can compress the 5-TB data to 4 TB before transferring. You can also extend the usage to the billing increment. For example, if you need to transfer 2 TB of data each day, you can modify the schedule to transfer 4 TBs of data every other day.

If neither option is feasible, you need to accept the extra cost. Rework the budgets as needed if the extra cost isn't included in the budget.

>  **Risk**: Cost optimization decisions shouldn't compromise security requirements or compliance regulations. If you opt for cheaper solutions without adequate security measures, you can expose the workload to potential vulnerabilities.

Azure facilitation

Determining billing factors and increments: Azure has product pricing details for every Azure product. Search for the products in your workload and catalog the different billing factors and increments for each billing factor. You can also use the Azure pricing calculator to estimate the cost of different increments.

Mapping usage to billing increments: You can use your Azure bill to analyze resource usage patterns and identify areas of high consumption. You can view and download your Azure invoice. These features help you understand how resources are utilized, so you can make informed decisions about optimizing their usage and minimizing unnecessary costs.

You can get a quick overview of your invoiced usage and charges on the Subscriptions page in the Azure portal. It's important to understand the terms in your Azure usage and charges file.

Aligning usage to maximize value: Microsoft Cost Management and Billing and Azure Advisor provide optimization recommendations that are based on usage and cost data. These recommendations help you identify opportunities for cost savings. With this data, you can determine if resources are over-provisioned or underutilized, and right-size them to match the workload requirements. Right-sizing resources can help align to billing increments.

Product SKUs represent the service tiers in Azure products. Azure offers various SKUs within each service. Switching SKUs can help you align billing increments with usage patterns. You can use the Azure product pricing pages to compare the different tiers for each product.

With Azure, you can set up cost alerts and budgets. Cost alerts notify you when consumption reaches predefined thresholds, allowing you to proactively monitor spending. Budgets help you set limits and track the burn rate of your resources, which helps ensure cost control.

Next steps

- Recommendations for collecting and reviewing cost data
- Recommendations for optimizing data costs

Cost Optimization checklist

Refer to the complete set of recommendations.

Cost Optimization checklist

Recommendations for optimizing component costs

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Cost Optimization checklist recommendation:

CO:07	**Optimize component costs. Regularly remove or optimize legacy, unneeded, and underutilized workload components, including application features, platform features, and resources.**

This guide describes the recommendations for optimizing workload component costs. Optimizing component costs refers to the process of evaluating and improving the cost-efficiency of individual elements within a workload. It emphasizes the continuous review and potential removal or improvement of outdated, unnecessary, or rarely used components, such as application features, platform features, and resources. It also covers cost optimization of disaster recovery environments and how to avoid introducing unoptimized components. The guidance in this article applies to existing workloads that aren't in the design phase. Neglecting regular component optimization can lead to inflated costs, resource waste, and inefficient workloads that drain both time and money.

Definitions

Term	Definition
Application feature	A distinct capability within the application software that enables users to perform specific tasks or access specific information.
Platform feature	A specific functionality or capability provided by a platform. It can vary depending on the platform, but generally, platform features are designed to enhance the user experience, improve productivity, or enable specific tasks or actions.
Resource	A single entity or component that you can create, configure, and utilize within a cloud service provider.

Key design strategies

Optimizing workload components is about refining the various elements of a workload, including application features, platform capabilities, and resource. The goal is to ensure the workload uses all components efficiently and cost-effectively. Strategies include removing, modifying, and avoiding components that cause you to spend more than you

need. The component cost optimization process ensures you allocate resources to features and components that deliver the most value, avoiding unnecessary expenses.

Optimize application features

Optimizing application features is the process of either removing, reinvesting, or monetizing application features based on value. It ensures you allocate resources to application features that provide the most value to customers. Optimizing application features helps avoid investing in features that contribute to technical debt or don't yield enough return on investment.

Evaluate application feature value

To determine the value of a feature, consider its effects on the overall application and the value it provides to the customers. Some factors to consider include:

- *Customer needs*: Assess how well the feature meets the needs and expectations of customers. Customer feedback, surveys, and usage data can be valuable in understanding the perceived value.
- *Business goals*: Evaluate how the feature aligns with the strategic objectives of the business. Consider how features support revenue generation, customer satisfaction, or competitive advantage.
- *Effect on user experience*: Determine the effect the feature has on enhancing the user experience and improving usability or productivity.
- *Differentiation*: Assess whether the feature provides a unique selling point or competitive advantage compared to other applications in the market.

Evaluate application feature cost

It's essential that you understand the cost associated with each feature for effective resource allocation and optimization. Consider various aspects when evaluating costs, such as:

- *Development effort*: Assess the time, resources, and expertise required to develop and maintain the feature or surrounding features. Underutilized features often become a key source of technical debt.
- *Maintenance and support*: Consider the ongoing costs associated with maintaining and supporting the feature, including bug fixes, security updates, and troubleshooting.

- *Infrastructure and resource utilization*: Evaluate the effect of the feature on infrastructure requirements, including server resources, storage, and bandwidth.
- *Integration complexity*: Assess the complexity and cost of integrating the feature with other systems or third-party services.
- *Performance considerations*: Evaluate the effect of the feature on the application's performance, including scalability, response time, and resource usage.

Review application feature value with stakeholders

Review the value of application features with stakeholders by engaging key personnel, such as product managers, software developers, and business analysts, to assess the value of specific features on business objectives. This collaboration is essential for cost optimization as it provides insights into maintenance efforts and identifies features that might hinder productivity or detract from developing new, valuable features. Your development team can give you important information about how much work it takes to maintain certain features. Encourage them to speak up about features that might be more trouble than they're worth, especially if these features distract the team from creating new ones.

Determine the future of the feature

Based on your analysis and evaluation, determine the future of the application features. Remove, reinvest, or monetize any application feature that doesn't provide a return on investment:

- *Removal*: Consider the planned end of life of an application feature based on data. Reasons for feature removal might include low customer demand, high maintenance costs, complexity, or redundancy that’s not worth the effort to fix. Create a plan for the removal, which might involve refactoring the code, updating dependencies, or reorganizing the UI.

  >  **Risk**: You can inadvertently remove features that are critical for certain users or scenarios and might negatively affect performance, operations, and security in your application.

- *Reinvest*: Some application features might not add enough value in their current state but could add value if you reinvest in them. Reinvestment means reworking or promoting the application feature. Prioritize the identified improvements based on their value and feasibility. Determine the roadmap and timeline for

implementing the changes. Consider factors such as development resources, dependencies, and the potential effect on the application.

- *Monetize*: Turn application features into a revenue-generating opportunity via monetization. Sometimes features provide value to users but aren’t worth the current investment. Explore opportunities to monetize these features, such as offering them as separate paid add-ons or licensing them to other companies.

Optimize workload resources

Optimizing workload resources involves removing any resources that unused and optimizing any underutilized resources that the workload needs. This effort can save money, avoid waste, and ensure that the workload only uses the resources that add value.

Remove unused workload resources. Unused resources are deployed services your workload or operations processes don't use. These resources might be long-term idled, orphaned, or forgotten. They provide no return on investment, and you should remove them. Common causes of unused resources include:

- Alerts.
- Demo builds.
- Environment decommissioning.
- Feature decommissioning.
- IP addresses.
- Network firewalls.
- Proof of concept.
- Snapshots.
- Storage accounts.
- Temporary testing environments.
- Temporary triage environments.

To remove unused resources in a workload, consider these steps:

1. *Take an inventory*: Conduct a thorough inventory of all resources within the workload across environments.

2. *Find orphaned resources*: Resources can become orphaned when they're no longer needed or when their parent resources are removed. For example, you might remove a virtual machine, but its associated storage account isn't removed. Review your workload to identify unneeded or orphaned resources.

3. *Remove idle components*: There's typically a cost associated with a deployed resource. Even if the resource allows you to stop or reallocate, you might continue to pay for the resource. Consider removing idle resources. If you need the data, back it up first and then remove the resource. You're better off redeploying the resource and restoring the data than allowing the resource to remain idle.

Optimize underutilized resources. Underutilized resources represent wasted expenditure as you pay for resource capacity that isn't fully utilized. Identify and optimize these resources to reduce costs and allocate resources more effectively. To evaluate and optimize the cost of underutilized resources, follow these steps:

1. *Monitor resources*: Use tools to monitor how much CPU, memory, and storage you actually use. Choose the best plan that matches your needs based on this information.

2. *Analyze utilization*: Look at the data to find out which resources you don't use. Pay attention to the resources that have low usage over time or large differences in usage between busy and slow times.

3. *Right-sizing*: Check if there are too many resources allocated to features that aren't in use. If so, adjust their size to better match what you actually need.

4. *Automatic scaling*: Use automatic scaling to adjust the resources you use based on how busy you are. Make sure that you set a maximum scaling limit to avoid sudden spikes that can be costly and unnecessary.

After you make these adjustments, test to make sure everything still works as it should. Continuously monitor resource utilization and adjust resource allocation as workload demands change over time. Regularly review and optimize resource utilization to maintain cost efficiency and performance optimization.

Optimize disaster recovery resources. Optimizing disaster recovery environments is about ensuring the resources allocated for disaster recovery are used efficiently. A warm (active-passive) disaster recovery strategy is a common source of underutilization. In a warm disaster recovery strategy, one environment receives all the load while the other environment is idle until there's a disaster scenario. To optimize a disaster recovery environment, consider how a hot (active-active), cold (active-off), or active-redeploy approach can help avoid underutilized resources. Here's an overview of these three disaster recovery approaches:

- *Hot plans*: Both the primary and secondary environments serve traffic concurrently. Your workload can balance loads between these environments and respond to demands in real-time. Distributing the load between two active environments,

allows you to use cheaper resources, reduce single-point bottlenecks, and utilize capacities to the fullest. It can lead to reduced costs in terms of resource wastage or idling. A hot approach might demand more investment in synchronization and maintaining parity between the two environments.

- *Cold plans*: A cold disaster recovery model involves a standby environment that remains dormant until a disaster triggers the need for failover. Since the standby environment isn't actively running, costs related to compute, storage, and network operations are minimized. Your expenses revolve around storing backups, virtual machine (VM) images, or templates. Failover in the cold model can take longer because resources need to be booted up and data might need to be restored. Ensure that the recovery time aligns with your business's recovery time objectives (RTO) before committing to this approach.

- *Active-redeploy*: This strategy uses infrastructure as code. When a failover event occurs, you deploy the secondary environment, using predefined templates and scripts. With no predeployed compute resources in the disaster recovery environment, you save on the costs associated with maintaining idle resources. You only incur costs during the actual deployment in a failover scenario. Like the cold approach, this model might introduce longer recovery times, especially if the infrastructure's complexity is high. You should test and measure the recovery time to ensure it meets your recovery time objective.

Optimize platform features

Optimizing platform features involves eliminating or updating platform features, such as performance tiers and configuration settings, to optimize costs. It helps align spending with the requirements of the workload and avoids unnecessary expenses on unneeded features. Here are some tips to optimize the cost of platform features:

- *Know the capabilities of the things you purchase*: Before you can optimize, you need a clear inventory of the services and their features across your cloud platforms. Understand the features and functionalities of the platforms or services in your workload. Be aware of the specific tier you chose and the features each tier offers. For example, if you don't require autoscaling or advanced networking, a lower-tier plan might suffice.

- *Disable unused features*: Identify and disable platform features that cost money. You might have unneeded storage snapshots, unused disks, redundant security features, or underutilized networking capabilities.

- *Use the right versions*: Newer versions of a service can provide similar performance for the same price. For example, a virtual machine with newer hardware can often provide the same performance for less money.

- *Use the right configurations*: You might be paying for more availability or performance than you need. Eliminate availability or performance guarantees that the workload doesn't need.

- *Eliminate unneeded automation*: Evaluate your automation processes and eliminate any unused automation that might incur extra costs.

- *Eliminate tool redundancy*: Get rid of tools that you don't need or tools that provide the same function. Evaluate potential redundancy in the tools you use for building software, writing code, security, and monitoring. For example, if you use GitHub Actions to build your software, you don't need to buy another tool that builds software. Before you buy features or tools, check if there's already a tool in your workload that can do the job. Eliminate tool redundancy to avoid wasted money and make the most of what you already have.

Prevent unoptimized components

Preventing unoptimized components is about proactively ensuring components are essential and optimized before adding or modifying. The best way to get rid of waste is to avoid it in the first place. Use strategies that prevent unnecessary expenses by addressing inefficiencies at the root, ensuring a workload runs cost-effectively from the outset. To help prevent waste, consider these strategies:

- *Find the root cause before changing solutions*: Before you fix a problem, make sure you know what's actually causing it. For example, if your website is slow, don't immediately switch to a new system. First, figure out why it's slow. You might find out that the real issue is something else, like bad database queries. Fix the real problem to save time and money.

- *Apply metadata*: Apply metadata to organize and track resources. You can use metadata to categorize and group resources, making it easier to track, delete, and avoid orphaned resources. Create a consistent metadata strategy across resources. Consider adding owners, the anticipated resource duration (for example, `sunset-30d`), or other tags.

- *Document nonstandard changes*: Document any changes made to your infrastructure or configurations performed outside the normal control process of your workload to cut unexpected costs. For example, you might increase a

resource's scaling (up or out) capacities to meet a short-term demand or triage an issue but forget to scale it back down. Make a list of nonstandard changes and use it as a reminder to revert the changes when they're no longer necessary.

- *Keep things simple*: Simplify your infrastructure and minimize complexity to help reduce costs. Use only the necessary resources and services that meet your requirements.

Azure facilitation

Optimizing application features: You can use Azure Monitor and Application Insights to monitor the usage of your application and identify areas that are or aren't used. Based on the insights gathered, you can make informed decisions to remove or optimize unused or underutilized features.

Optimizing workload resources and platform features: Azure Advisor provides cost recommendations provides recommendations to identify and eliminate unused resources. You can use Advisor to analyze your resource usage and receive suggestions about resources to remove or scale down. The Cost Optimization workbook in Azure Advisor serves as a centralized hub for some of the most commonly used tools that can help you drive utilization and efficiency goals. It offers a range of recommendations, including Azure Advisor cost recommendations. It also helps identify idle resources and manage of improperly deallocated virtual machines.

Azure Monitor supports workbooks. With Azure Monitor workbooks, you can find or create a workbook that finds and reports orphaned resources across a defined scope. You can use Azure Automation to shut down virtual machines during periods of inactivity. Resource shutdowns help reduce costs by minimizing the use of idle resources.

You can use the autoscale feature in Azure to automatically scale your application based on predefined conditions, so you don't have to overprovision capacity. Automatic scaling can help you allocate resources efficiently and cost-effectively.

From a design perspective, Azure load balancers can distribute loads across availability zones and regions. These load balancers can help eliminate idle resources, for example, in disaster recovery approaches.

Related links

- Recommendations for continuous performance optimization
- Recommendations for defining reliability targets

- Recommendations for highly available multi-region design
- Recommendations for optimizing scaling and partitioning
- Recommendations for setting spending guardrails

Cost Optimization checklist

Refer to the complete set of recommendations.

Cost Optimization checklist

Recommendations for optimizing environment costs

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Cost Optimization checklist recommendation:

CO:08	**Optimize environment costs. Align spending to prioritize preproduction, production, operations, and disaster recovery environments. For each environment, consider the required availability, licensing, operating hours and conditions, and security. Nonproduction environments should emulate the production environment. Implement strategic tradeoffs into nonproduction environments.**

This guide describes the recommendations for cost optimizing workload environments. Each environment should be tailored for its specific purpose and optimized for cost effectiveness. It's important to make strategic tradeoffs and allocate resources where they matter the most, without compromising on critical components. By treating environments differently and optimizing them accordingly, you can achieve a balance between cost optimization and meeting the required objectives.

Definitions

Term	Definition
Recovery point objective (RPO)	The maximum acceptable duration of data loss during an incident.
Recovery time objective (RTO)	The maximum acceptable time that an application can be unavailable after an incident.
Service-level agreement (SLA)	A contractual agreement between the service provider and the service customer. The agreement defines the service-level objectives (SLOs). Failure to meet the agreement might have financial consequences for the service provider.

Key design strategies

The goal of optimizing environment costs is to find the right balance of value, cost, and risk for each environment, including production, preproduction, and disaster recovery (DR) environments. Customize each environment for its particular use to save money and efficiently use resources. Determine the benefits of each environment, like efficiency or customer satisfaction. You want to evaluate the return on investment (ROI) for the

environment, even if it doesn't make a direct profit. Spend more money on high-risk environments to reduce issues and save money on low-risk environments. Aim to balance value, cost, and risk in each environment.

Assess environment value

Assessing the value of each environment means understanding its broader effect on business, gauging user satisfaction, and determining how it aligns with overarching organizational goals. This assessment helps you make informed decisions about resource allocation and align cost with environmental priorities. The essence of value extends beyond how much revenue an environment generates. When evaluating an environment's value, you need to prioritize spending in a manner that resonates with the goals of the workload. To assess the value of each environment, consider the following factors:

- *Consider the user*: Consider who uses each environment and what they need from it. For example, customers use the production environment, which must be reliable and meet specific SLAs for performance and uptime.

  On the other hand, the development environment is mainly for the workload team, such as developers and testers. This environment doesn't have to meet customer-facing SLAs, but it should have the necessary tools and resources for the team to work effectively.

  When you understand the unique needs of users in each environment, you can better allocate resources and avoid extra costs. This avoidance helps ensure each environment is functional and cost effective.

- *Align with organizational measures of value*: Align your cost-cutting efforts with your organization's priorities, like profit or employee satisfaction. For each environment, understand how success is defined, so you can keep your actions on target. For example, if your organization focuses on profit maximization or employee satisfaction, align your spending decisions with those metrics.

Determine environment costs

Determining environment costs is about knowing the costs of infrastructure, services, licenses, and operational expenses in each workload environment. Cost management tools are key to gaining insights into spending patterns and trends across environments. To determine environment costs, consider these strategies:

- *Identify cost drivers*: Identify the key factors that drive costs in each environment. These factors can include resource utilization, storage usage, data retention, data transfer, and specific services.
- *Evaluate risks*: Assess the risks that are associated with spending decisions and their potential effect on the environment and business operations. Consider factors such as data security, compliance, performance, audits, and SLA requirements.
- *Monitor and adjust your spending*: Continuously monitor and analyze spending patterns, value delivery, and risk factors. Regularly review and adjust your spending optimization strategies as the needs of the environment and business evolve.

Optimize the production environment

Optimizing costs in the production environment involves implementing strategies to reduce unnecessary expenses and improve operational efficiencies. Focus on differentiating production deployments and meeting the needs of users. Here are recommendations for optimizing the production environment:

- *Differentiate regions*: Spend less on regions that serve fewer customers. For example, you should invest more in a region that serves 90 percent of your users than in a region that serves 10 percent of your users. Adjust your deployment strategy to meet the requirements of each region and user segment.
- *Differentiate scaling*: Implement horizontal and vertical scaling strategies. Scale resources efficiently to meet demand without over-provisioning.
- *Differentiate infrastructure*: Choose cost-effective hardware and infrastructure solutions that meet the required performance and scalability. Consider factors such as performance, cost, reliability, and scalability.
- *Tune tenant models*: Customize the environment based on the tenant model. For example, spend more on services and features for paid tenants and spend less for nonpaying tenants.

Optimize the DR environment

A DR environment refers to infrastructure and processes that a workload uses to recover after a disruptive event. Disruptive events include natural disasters, cyber attacks, and hardware failures. Balance the cost of maintaining a DR environment and the potential affects of a disruptive event. Consider the following strategies:

- *Evaluate the criticality of systems and data*: Assess the importance of systems and data to determine the required level of protection and resources for each component.
- *Determine RTOs and RPOs*: To help determine the design of the DR environment, define the acceptable downtime and data loss limits for each system or application.
- *Optimize a cold DR environment*: A cold DR environment has little or no infrastructure or running services. You can use infrastructure as code (IaC) to quickly deploy infrastructure during a disruptive event. Your backup and storage policies need to meet the RPOs and RTOs of the environment. Ensure that the amount and frequency of data backups isn't more robust than needed.

  >  **Tradeoff**: A cold DR environment is a cost-effective option, but you might have long recovery times.

- *Optimize a hot DR environment*: All infrastructure and services run in a hot DR environment. The data mirrors the primary site in real time. It provides near-instantaneous failover and minimal data loss if there's a disaster. Consider an active-active deployment to optimize costs.
- *Optimize a warm DR environment*: A warm DR approach is a middle ground between a cold DR environment and a hot DR environment. A warm environment is partially active and periodically syncs with the primary site. It offers a balance between cost and recovery time. However, it's the least cost-optimized approach. Consider a cold or hot approach to optimize costs.

Optimize preproduction environments

Optimizing preproduction environments involves strategically managing resources within development, testing, and staging areas to closely simulate production while reducing unnecessary costs. Preproduction environments don't require the full scale and availability of production environments. The most opportunities lie in tailoring these environments to specific testing and development needs without duplicating production exactly. Areas of cost reduction include using lower-cost resources, turning off unneeded services, and applying discounts offered for preproduction usage. Consider the following strategies to optimize preproduction environments:

Evaluate preproduction environments

Insufficient or improper allocation of preproduction environments might lead to over-provisioning or under-provisioning of resources. To evaluate your preproduction environments for your workload, consider the following guidance:

- *Understand the environment types*: Identify the types of preproduction environments, such as development, testing, and staging, that you need for your workload. Each environment should have a defined role and specific function to ensure efficient resource allocation.
- *Align with users' requirements*: Before you set up preproduction environments, understand the requirements and expectations of your users. Tailor the features and specifications based on their needs to avoid unnecessary expenses of features or resources.
- *Consolidate the environment*: Determine if you can combine environments without compromising their functionality. Combine environments that have functions that don't overlap. For instance, you can merge a user acceptance environment with a quality assurance environment. The functions are distinct, and one environment is usually idle when the other is in use.

  >  **Risk**: Be cautious when you combine environments to ensure that you don't introduce conflicts or compromise the testing or development processes.

The following table provides examples of common preproduction environments.

Preproduction environment example	**Description**
Development environment	Developers use this environment to write and test code. It provides a sandbox space, so developers can experiment, build, and integrate code changes.
Quality assurance environment	This environment is dedicated to quality assurance activities. It for testing to identify and fix bugs or issues before deploying to the production environment.
Security environment	This environment is for security testing. It's for ensuring an application is secure against threats and vulnerabilities.
User acceptance testing environment	In this environment, end users and stakeholders test an application to validate its functionality and ensure that it meets requirements and expectations.
Staging environment	This environment closely resembles the production environment. It's for final testing and validation before deploying to production.

Apply governance

Applying governance is about limiting deployment options in preproduction environments to control expenses and mitigate risks. In preproduction, you have flexibility to tailor configurations and deploy resources. The more the preproduction environment deviates from the production environment, the greater the potential risk. Use governance to constrain preproduction environments. Consider the following guidelines:

- *Limit performance tiers*: Evaluate the performance requirements of your preproduction environments. Choose performance tiers that balance cost and performance. A service often has different performance tiers, and some of these tiers are more suitable for testing. Some services have tiers that offer production-like features but don't come with an SLA. These services reduce costs but still provide the necessary functionality for testing and development.

- *Understand preproduction SKUs*: Some SKUs are designed for development environments. To optimize costs, evaluate services and tiers. Opt for low-performance tiers if the workload doesn't require high performance.

- *Control the number of instances and CPUs*: Determine the optimal number of instances and CPU resources that your preproduction environment needs based on workload demands. Avoid over-provisioning resources to minimize costs.

- *Limit retention and logging*: Define retention policies for logs and data in preproduction environments. Consider the necessary duration for retaining logs and data based on compliance requirements and cost considerations. Avoid excessive logging and retention to reduce storage costs.

- *Use a consistent CPU architecture*: Use the same CPU architecture in preproduction and production. For example, x86 applications don't run natively on Azure Resource Manager, and vice versa. Use the same CPU architecture as your production environment to ensure compatibility and minimize potential issues.

- *Use the same operating system*: Avoid changing the operating system (for example from Windows to Linux) or kernel in preproduction environments. Software built for Windows often doesn't run natively on Linux without a compatibility layer, and vice versa. The file systems and directory structures are different, which can cause application patching issues. Consistent environments help reduce the risk of compatibility issues and ensure smooth deployments.

- *Constrain scaling*: To optimize cost, you can constrain automation to mitigate runaway automation. For example, set a maximum scaling limit at three in the

development environment, and set it at 10 in the production environment. Constrain scaling to help control the resource usage and automation cost.

- *Turn off unneeded resources*: Turn off resources when they aren't actively used, such as during off hours and weekends. You can use automation tools or scripts to schedule the shutdown and startup of resources. Some vendors provide APIs that you can use to programmatically stop and start the resources. Consider using IaC to create ephemeral environments that you can remove when you no longer need them.

Balance similarity with production

It's often unnecessary and expensive for preproduction environments to mirror the production environment exactly. The goal is to ensure each preproduction environment is appropriately different from production to avoid unnecessary costs. However, when preproduction and production are different, there's a risk of deploying a bug into production. The more different these environments are, the more risk there is. Tailoring the preproduction environment to meet your needs can help you manage risks while optimizing cost. To balance the similarity with production, consider the following recommendations:

- *Avoid exact replicas*: Avoid making the preproduction environment an exact copy of production. It can unnecessarily increase costs. Create a preproduction environment that's cost-effective but enable you to uncover and address potential risks before deployment.

- *Avoid extreme deviations*: Avoid excessive deviation from production, like the use of different services. Different services might not accurately simulate real-world risks. Determine a risk threshold, and don't cross the threshold solely to save money.

- *Shorten runtimes*: Consider shortening the runtimes of processes in the preproduction stage to save money. Be cautious of new vulnerabilities that might arise, such as undetected memory leaks.

- *Review licenses*: Review the licensing plans for your security tools. If the number of nodes vary significantly between your production and preproduction setups, reassess your needs to fine-tune costs without compromising security.

Optimize development environments

Development environments are designed for development, testing, and debugging purposes. They have shorter lifecycles and are often created as needed and exist for a short time. Development environments typically have lower requirements for reliability, capacity, and security compared to other preproduction and production environments. They might have fewer capabilities and can accept lower resource utilization. To optimize your development environment:

- *Evaluate tooling*: Regularly assess the cost-effectiveness of your current tooling setup, including integrated development environments (IDEs), licenses, and related tools. Consider free or open-source alternatives that offer similar functionality without compromising quality. Continuously reevaluate the necessity and efficiency of these tools as the development landscape evolves.

- *Consider hardware*: Evaluate the cost and performance of your current hardware setups. Investing in better and more efficient hardware can enhance productivity and reduce long-term costs. Instead of frequent hardware replacements, consider upgrading existing systems to prolong their lifespan and improve performance.

- *Optimize the number of environments*: Analyze the advantages and disadvantages of individualized development environments versus a shared environment. Individual environments can mimic production setups, prevent interference among developers, and offer customized setups. However, scaling becomes more costly as the number of developers increases. Shared environments can save costs, but reliability concerns might arise if issues affect the entire development team simultaneously. Find the right balance based on cost, risk mitigation, efficiency, and developer satisfaction.

- *Regularly clean up*: Routinely clean up and optimize your development environment to avoid the accumulation of orphaned resources, unused data, and proof-of-concept experiments. Implement clean-up processes or automated tools to identify and remove unused resources. Keep only essential and active components. Regular clean-up helps reduce storage costs and ensures efficient resource utilization.

- *Implement sampled scaling*: Instead of scaling all components to their maximum capacity, consider a sampled approach in which you selectively scale vital components. This approach can be cost-effective while minimizing risks. Evaluate the risk-to-benefit ratio of not scaling certain elements and consider the potential effect on the environment.

- *Optimize data management*: Development environments might have low needs for data retention and backup frequency.

Consider endpoint emulation

You can optimize costs in a preproduction environment by using endpoint emulation or mock endpoints, particularly for expensive resources like GPUs. Identify components or services in your preproduction environment that are the most expensive or resource-intensive. Use mock endpoints to simulate the responses of these costly components without invoking them. To simulate API responses, you can use tools like WireMock, Postman's mock server, or custom implementations.

Emulation and mock endpoints help save costs, but you must ensure that they represent the production environment to a sufficient degree for testing. Strike a balance between accuracy and cost to help avoid future issues in production. For example, if GPUs are a major cost factor, consider GPU emulation for tasks that don't require real GPU processing power in preproduction stages. Emulation might not fully represent the performance or quirks of real GPUs, so use it when exact GPU behavior isn't critical for preproduction testing.

Azure facilitation

Determining and optimizing environment costs: Microsoft Cost Management is a suite of tools that help organizations monitor, allocate, and optimize the cost of their Microsoft Cloud workloads. Cost Management is available to anyone with access to a billing or resource management scope.

Azure Advisor is a tool that provides cost optimization recommendations, including identifying areas of virtual machine usage that need optimization. Use Advisor to help you make informed decisions and optimize costs in your Azure environment. Azure provides cost management tools and features that help prioritize spending. You can use these tools to track and analyze costs across environments, set budgets, and receive cost optimization recommendations.

Applying governance: With Azure Policy, you can limit resource types, SKUs, and instances by defining policy rules that enforce restrictions on the types of resources that you can deploy in your Azure environment. You can maintain control over the provisioned resources and ensure compliance with your organization's policies and best practices.

To limit resource types by using Azure Policy, you can define policy rules that specify the allowed resource types. Apply those rules to the relevant Azure subscriptions or resource groups. Azure Policy prevents users from deploying resources that aren't allowed.

Use Azure Resource Manager to define and manage resources in a declarative manner. You can tune resources that are allocated to each environment based on their specific requirements. Use templates and parameterize resource configurations to optimize costs.

Optimizing preproduction environments: Azure offers dev/test pricing options that provide discounted rates for nonproduction environments. You can allocate more resources and budget to critical production environments, which optimizes costs in nonproduction environments. You can also use the Azure licensing offer, Azure Hybrid Benefit.

You can use Azure API Management for API mocking. API Management acts as a facade to back-end services, which allows API providers to abstract API implementations and evolve back-end architecture without affecting API consumers.

Organizational alignment

Cloud Adoption Framework provides guidance to stress the importance of optimizing environment costs across the organization.

For more information, see Best practices for costing and sizing Azure resources.

Cost Optimization checklist

Refer to the complete set of recommendations.

Cost Optimization checklist

Recommendations for optimizing flow costs

Article • 12/20/2023

Applies to this Azure Well-Architected Framework Cost Optimization checklist recommendation:

Expand table

CO:09	**Optimize flow costs. Align the cost of each flow with flow priority. When you prioritize flows, consider the features, functionality, and nonfunctional requirements of each flow. Optimizing flow spend often requires strategic compromises.**

This guide describes the recommendations for optimizing the cost of each of the flows in your workload. Cost-optimizing the flows in a workload involves strategically allocating and managing resources to minimize expenses while maintaining performance. This optimization is crucial because it ensures efficient utilization of invested resources, reduces unnecessary expenditures, and improves the overall return on investment for the infrastructure. If you don't cost-optimize the flows in a workload, you risk overspending on resources, which results in inflated operational costs and diminished profitability.

Definitions

Expand table

Term	Definition
Decouple	The strategy of removing a flow from a resource that contains multiple flows and placing it into a separate resource.
Flow	In a workload, the sequence of actions that performs a specific function. A flow involves the movement of data and the running of processes between components of the workload.
System flow	The flow of information and processes within a system. The system automatically follows this flow to enable user flows or workload functionality.
User flow	The paths or sequences of actions that users take within an application or system.

Key design strategies

Invest more in high-priority flows than in lower-priority flows. Aligning flow priority and spending can involve decoupling flows that currently share the same resource. It can also involve combining flows that have similar requirements but are run on separate resources. For example, suppose you have a web application that includes multiple flows, like user registration, sign-in, and data processing. These flows run on a single server, even though they have different resource needs. To optimize both costs and performance, you might separate flows or combine flows:

- *Separate flows*. For example, you might decouple the user registration flow from the others and move it to a dedicated, lower-cost server. This flow is important but not resource-intensive, so it's a good candidate for a less expensive server.
- *Combine flows*. For example, you might combine the sign-in and data processing flows, which both have higher resource requirements, and run them together on a high-performance server. Combining these flows enables the server to efficiently handle the resource-intensive needs of both flows. It optimizes performance and costs.

In a workload, there can be different types of flows or paths that you need to consider. This guide focuses on the following flow types:

- *System flows*. Optimizing system flows involves streamlining the communication and interaction between system components, minimizing bottlenecks, and ensuring efficient resource utilization.
- *User flows*. Optimizing user flows involves improving the user experience, reducing friction points, and ensuring smooth navigation and interaction within the application or system.

Create an inventory of flows

A flow inventory is a comprehensive list and description of all the sequences of actions, data transitions, and system interactions within a workload. A flow inventory is the first step to ensuring investments align with the priority of flows. You should only optimize flows when you fully understand their purpose and dependencies. Here are steps for creating an inventory of workload flows:

1. *Document flows*. Start by documenting and listing all existing flows in your workload to get an understanding of the comprehensive state of the system. Include every sequence of actions, data transitions, and system interactions. Familiarize yourself with every component, like external services, databases, middleware, and third-party integrations. Additionally, track or estimate the volume of requests over time.

2. *Visualize flows*. To get a clearer perspective, represent your findings visually, possibly in flowcharts or diagrams. Visualizations help you see the interdependencies between components. Consider using a tool like Visio to help you with the visualizations.

3. *Categorize flows*. Bundle similar flows, taking into account attributes like their functionality (for example, authentication, data retrieval, and transaction processing), criticality to business, or the resources they use (CPU, memory, or bandwidth).

Prioritize flows

Flow prioritization is the process of classifying flows based on their influence on business outcomes, implications on user experience, and the resources they consume. Critical flows often require higher levels of availability, faster recovery times, and better performance to meet workload objectives. By prioritizing flows, you can better align spending to flow priority. To prioritize flows, consider the following steps:

- *Identify flow value*. When you optimize workload flow costs, you need to identify the flow that provides the most value. You don't want to spend more than a flow is worth. Instead of simply cutting costs, consider shifting costs to prioritize the more valuable flows. For example, your checkout flow is critical for business, but the purchase history isn't. You should allocate more resources and budget to the checkout flow.

  Low-priority flows have lower expectations for availability, recovery, and performance. You can reduce costs by using cheaper configurations to reduce performance, availability, or business continuity spending.

- *Consider flow metrics*. If you're struggling to prioritize your flows, consider the availability and recovery goals that you assigned to them. Critical flows often have high availability requirements and service-level agreements (SLAs). Flows associated with a lower RPO and RTO are more important than flows that have a higher RPO and RTO.

Optimize independent flows

Sometimes your flows are already running on different resources. In these cases, you can more easily evaluate and optimize spending. Evaluate the components and processes involved in each independent flow to determine whether there are ways to optimize or simplify them. To optimize independent flows, you can follow these steps:

- *Eliminate unnecessary components*. Remove any extraneous elements that don't contribute to the flow's core functionality, thereby reducing complexity and cost.
- *Redesign the flow*. Consider redesigning the flow's architecture to enhance its efficiency. You might change the sequence of operations, reduce latency, or improve data transfer speeds, for example.
- *Choose an appropriate performance tier*. Different flows might have varying demands in terms of processing speed, memory, or other resource metrics. Make sure to choose a resource tier that aligns well with each flow's specific requirements.
- *Adjust scaling settings*. If a flow experiences variable demand, consider implementing autoscaling to dynamically adjust resources according to real-time needs, thus optimizing costs.
- *Fine-tune configurations*. Fine-tune other settings, like networking or data storage options, to better align with the flow's performance and budget requirements.

Separate dissimilar flows

Separating dissimilar flows onto different resources is a process of allocating distinct tasks with varying computational needs to dedicated resources. Dissimilar flows are flows that have different attributes. These attributes can include computational requirements, data dependencies, I/O operations, latency sensitivity, security needs, and compliance requirements. It's often more cost efficient to run different types of flows on separate resources. Doing so enables precise resource allocation to each flow, which reduces unnecessary expenditures and ensures maximum efficiency.

Consider separating dissimilar flows that are currently combined. This separation boosts scalability, fault tolerance, and adaptability and also streamlines costs. By ensuring that each flow operates independently, you reduce interference risks and can allocate resources more cost-effectively based on each flow's priority. For example, assume that you colocate CRM (user flow) with a data engine (data flow). User traffic to the CRM system during office hours might slow down the data engine. When you decouple flows, the data engine can scale each component or service independently based on workload demand. This decoupling optimizes resource allocation and reduces costs.

Combine similar flows

Combining similar flows onto a single resource is a process of consolidating tasks or processes with comparable attributes and using shared resources for them. This strategy

eliminates redundancies and ensures more efficient use of resources, leading to significant cost savings. Similar types of flows share similar attributes. You might consider the same attributes that you look at when you separate dissimilar flows: computational requirements, data dependencies, I/O operations, latency sensitivity, security needs, and compliance requirements. Here are some examples where combining similar workload flows to use the same resource can lead to substantial savings:

- *Web servers*. Instead of dedicating separate web servers for each application, consider consolidating them, especially if their traffic isn't consistently high. A shared web server, paired with a reverse proxy, can effectively manage and route traffic to multiple applications.
- *API gateways*. Rather than maintaining individual API gateways for separate microservices or applications, you can use a centralized API gateway to streamline requests and direct them to the relevant service. Doing so makes management easier and also reduces costs.
- *Log processing*. Instead of having multiple applications or services that each operate their own log processing instances, consider directing them all to a shared log processing tool. This approach minimizes the number of active instances, which translates to direct cost savings.
- *Authentication services*. If multiple applications deploy their own distinct authentication mechanisms, redundancy is introduced. Integrating a single sign-on (SSO) solution or a communal authentication service reduces this duplication and optimizes resource usage, which reduces costs.

>  **Risk**: Don't mistake coincidence with design. Two flows that look similar don't necessarily serve the same purpose. You need to understand the function and design of each flow before merging or changing them. Misinterpreting a flow by focusing solely on its appearance can lead to unintended consequences and disrupt the service or process that it supports. If multiple flows serve the same function and there are no discernible differences in their design or intent, consider consolidating them.

Continuously monitor flows

The nature of flows and workloads can change over time, so you need to review flow spending to ensure that costs align with priorities. Evaluate the resource utilization of each flow by analyzing the compute, storage, and network usage associated with each flow. Identify any inefficiencies or areas where resources are underutilized. This analysis

helps you pinpoint opportunities for cost optimization. Here are some considerations to take into account when you review flow utilization:

- *Analyze usage patterns*. Analyze the usage patterns of the flows. Some flows might be more active during certain times of the day or month, while others might have a consistent load. By understanding these patterns, you can predict resource needs and adjust allocation to avoid bottlenecks and overprovisioning.

- *Monitor relevant metrics*. Determine the metrics that can help you assess the efficiency and cost-effectiveness of each flow. Consider CPU utilization, data transfer costs, transaction costs, and storage footprint. Use monitoring tools to gather detailed metrics about resource usage and performance.

- *Consider ongoing maintenance*. Consider the cost of maintenance, especially when you use infrastructure-as-a-service solutions like virtual machines. You need to account for activities like patching, upgrades, backups, monitoring, and security.

During your analysis, identify any inefficiencies or areas where resources aren't utilized effectively. Consider idle compute instances, unused data, and low network bandwidth. These inefficiencies can indicate opportunities for cost optimization.

Azure facilitation

Prioritizing, optimizing, and monitoring flows: The User Flow tool in Application Insights provides a visual representation of user navigation across your site's pages and features. This tool aids in identifying areas where users frequently leave, repeat actions, or follow specific paths. By comparing actual user behavior with your anticipated outcomes and objectives, you can identify critical flows. It also allows you to optimize potential issues such as high churn rates, repetitive actions, or design flaws. The tool also allows for custom property filtering through dimensions, offering a more tailored analysis.

Azure Monitor helps you gain insights into the performance and health of your applications. It provides monitoring and diagnostics capabilities. These capabilities enable you to identify performance bottlenecks, optimize resource utilization, and detect and troubleshoot issues that might affect costs.

Log Analytics is a tool that enables you to collect, analyze, and visualize log data from various sources. By using Log Analytics, you can gain insights into your application and infrastructure logs, identify trends, and optimize costs by managing usage and data retention. Consider colocating logs and using dedicated solutions instead of shared ones to better manage costs.

Related links

- User Flow tool
- Azure Monitor

Cost Optimization checklist

Refer to the complete set of recommendations.

Cost Optimization checklist

Recommendations for optimizing data costs

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Cost Optimization checklist recommendation:

CO:10	**Optimize data costs. Data spending with data priority. Data optimization should include improvements to data management (tiering and retention), volume, replication, backups, file formats, and storage solutions.**

This guide describes the recommendations for optimizing data costs for a workload. Optimizing data costs involves minimizing the expenses related to the storage and management of data according to its significance and access frequency. Appropriate data management can significantly reduce overhead costs and align spending with data utility. Neglecting to optimize data costs can lead to inflated expenses, inefficient resource allocation, and financial waste due to misaligned storage solutions and unnecessary data retention.

Definitions

Term	Definition
Data lifecycle management	The process of managing data throughout its entire lifecycle, from creation to deletion. This process involves organizing, storing, protecting, and archiving data based on its value and usage patterns.
Data redundancy	The practice of storing duplicate copies of data across multiple storage systems or locations. The purpose of data redundancy is to improve data availability and fault tolerance.
Data tiering	A storage strategy that involves categorizing data based on its access frequency and storing it on storage tiers accordingly.
Retention policy	The duration for which data should be retained before it can be deleted. It specifies the time period during which data must be preserved to meet legal, regulatory, or business requirements.

Key design strategies

Within a specific workload, you optimize data costs by reducing the expenses associated with storing and managing data. There are various strategies and best practices to minimize data storage and processing costs. The goal is to align data costs with data

priority. You need to assign cost tiers to types of data based on their importance or frequency of access.

The primary drivers for the cost of workload data are access frequency, access latency, and storage amount. The following guidance contains strategies for optimizing costs across these cost drivers.

Take an inventory of data

Before you can optimize the cost of your data, you need to generate an inventory of data. Examine data access and determine its importance within your workload and its operations. Identify which data is accessed frequently and which data is accessed less frequently. The following inventory actions can help you allocate storage resources effectively:

- *Collect data access information:* Conduct a data audit to identify and catalog all data stores. Determine the value of data sets based on their importance to business operations, return on investment, and frequency of use. Gather access logs, usage metrics, or analytics from your data storage solutions.
- *Identify data types:* Categorize data based on its type, such as personal data, financial data, intellectual property, or operational data. Understand the sensitivity and criticality of each data type.
- *Identify access patterns:* Identify the patterns in data access, such as daily, weekly, or monthly usage patterns. You should understand latency, file sizes, and data freshness requirements for that data.

Prioritize data

Data prioritization is the process of categorizing and assigning importance levels to types of data based on sensitivity and criticality. Data priority should align with the importance of the environment. For example, production data is more important than preproduction data.

Assess the importance of various types of data to your workload by using these steps:

1. *Define priority levels:* Establish priority levels for data (such as high, medium, and low) based on its value to the organization, regulatory requirements, and potential effect of data loss. The goal is to align data priority to the appropriate data solution.

2. *Assign labels:* Label each data set with its sensitivity and criticality. You can apply labels at the row, column, or file level, depending on the data structure and usage. For databases, you can use a special tool to label and relate the sensitivity and criticality of data to specific rows and columns. This approach provides granular control over the management and access of data.

Optimize data management

Data management is the process of storing, moving, and securing workload data. By optimizing data management, you can align spending to data priority and derive more value from your data. Consider the following strategies for data management.

Optimize data lifecycle management

It's important to manage data throughout its lifecycle. Stages of the lifecycle include data creation (or acquisition), storage, usage, sharing, retention, and disposal (deletion or archiving). The goal of data lifecycle management is to optimize data storage solutions while complying with relevant regulations and policies.

Data storage has three critical cost components:

- *Storage cost*: The expense associated with storing data, such as per gigabyte.
- *Transaction cost*: Costs linked to data operations, such as write operations, read operations, and data retrieval (per gigabyte). Reading and writing data might have different costs.
- *Latency cost*: The expense associated with the speed or delay in accessing the data.

The following considerations are foundational to data lifecycle management:

- *Use data tiering:* The goal of data tiering is to align access and retention with the most cost-effective storage tier. Storage tiers range from frequent/immediate access (hot) to infrequent/delayed access (cold).

  It costs more to use a tier that doesn't align with data access and retention needs. For example, data that your application accesses frequently should be in hot storage. Data that your application accesses infrequently should be in cold storage. Effectively managing these aspects helps ensure efficient data storage.

- *Consider compliance requirements:* Implementing data tiering requires careful consideration of compliance requirements and data governance policies. Compliance and legal requirements often drive data access and retention. Establish

data retention policies to ensure compliance with legal, regulatory, and business requirements.

- *Define data lifecycle policies*. Data lifecycle policies specify when and how data should be moved between storage tiers based on predefined criteria. These policies ensure that you keep data in the appropriate tier for the required duration. For example, a policy can state that data must be retained in the hot tier for 30 days, in the cool tier for 90 days, and in the archive tier for one year. Set the retention period based on factors such as legal requirements, industry regulations, or internal policies.

- *Use automation*: Retention policies can trigger the movement of data between tiers. You should automate policies by using platform features before you build any custom solution.

  When the retention period for a particular tier expires, the policy can automatically move the data to the next lower-cost tier. For example, when the retention period for the hot tier ends, the policy can move the data to the cool tier. The policy ensures that data is continuously optimized based on its access patterns and cost requirements.

>  **Tradeoff**: Managing data retention policies requires ongoing monitoring and maintenance. It can introduce more overhead for data management processes. It might also affect storage costs. Longer retention periods or the use of higher-cost storage tiers can increase storage expenses.

>  **Risk**: A poor implementation of data lifecycle management could lead to data loss or limited access to critical data. You should have proper backup and recovery mechanisms in place to mitigate the risk of data loss.

Optimize data segmentation

Optimizing data segmentation involves strategically organizing data into distinct segments and consolidate similar data types to efficiently allocate storage resources. It allows you to tailor allocation of storage resources to data priority.

To effectively optimize data segmentation, you categorize data by type and usage pattern. Then you place the data segments on the most-effective solution depending on their operational similarities and requirements. For example, you place data that requires high-performance storage on resources with faster retrieval time. Archival data uses a lower-cost resource with slower retrieval time.

This approach ensures that high-demand data uses faster storage for optimal performance and less accessed data uses cheaper storage. Similarly, when data types share usage patterns, you should group them together on a single resource to reduce overhead, simplify management, and improve data handling.

Minimize data transfer

Minimizing data transfer refers to the reduction of data movement across networks to decrease data transfer costs. It reduces the volume of data that the workload moves and lowers network usage fees. To minimize data transfer, consider the following recommendations:

- *Use the right location.* place data geographically closer to its users. Data proximity reduces network travel, which speeds up access and optimizes costs.
- *Use caching.* Consider the benefits of caching to minimize data transfer.
- *Use a content delivery network.* A content delivery network can store frequently read static data closer to users. It reduces data movement across the network and helps offload bandwidth usage.

Optimize security and compliance

Certain production data demands higher security and compliance requirements. These measures might impose extra costs related to data protection, encryption, backup, retention, and auditing.

You must ensure that changes in data storage solutions adhere to these requirements. Data that has lower security and compliance requirements often presents an opportunity to optimize cost.

Optimize data volume

Finding strategies to decrease the amount of data that you store can help reduce costs. By changing the accessibility of the data and implementing the following techniques, you can effectively optimize the volume of your stored data:

- *Capture less data*: Take a closer look at the data you're capturing. Determine if any of it's unnecessary for your purposes. Modify your process, settings, or configurations to capture only the essential data.

- *Compress data*: Compression saves money by reducing the size of data. It's most effective in write-once, read-never or read-rarely scenarios. It's more suitable for colder storage.

 **Tradeoff**: Both compression and decompression of data increase CPU time.

- *Delete unneeded data*: Implement policies to streamline the process of storing relevant information. Evaluate the retention period for backups and snapshots, and delete data that you no longer need. You might want to have a process that leads up to eventual data deletion, such as first archiving data and enabling a soft-delete period. Always consider recoverability before deleting data.

- *Deduplicate data*: Implement data deduplication techniques to eliminate redundant data. Deduplication reduces storage requirements by ensuring that you store only unique data blocks, so you save costs. Use hashing algorithms and comparison of data chunks. Regularly run deduplication processes to identify and eliminate duplicate data.

- *Optimize user behavior*: In workloads that collect user-generated data, educate users on the importance of efficient data storage. Encourage them to regularly review and delete unnecessary files and data. Implement storage quotas or pricing models that discourage excessive data storage.

Optimize data replication

Data replication involves creating multiple copies of data and storing them in other geographic locations or zones for reliability. Replication ensures that if one location or zone experiences a failure or outage, you can still access the data from the replicated copies in other locations.

This redundancy helps improve the availability and resilience of data. It minimizes the risk of data loss and downtime.

To optimize data replication for cost optimization, consider the following guidelines:

- *Evaluate data replication requirements*: Assess the specific needs of your workload and determine the level of data replication that it requires. Consider factors such as data criticality, recovery time objectives (RTOs), and recovery point objectives (RPOs).

- *Choose the right replication strategy*: Select a replication technology that aligns with your goals for cost optimization. Consider the service-level agreement (SLA) requirements for your workload.

  Evaluate options such as synchronous replication, asynchronous replication, or a combination of both. Base the decision on factors like data consistency requirements and network bandwidth considerations. Assess the level of

availability that you need for your workload, and evaluate the need for zonal versus regional redundancy.

- *Optimize network bandwidth*: Minimize the usage of network bandwidth by implementing compression and data deduplication techniques. These techniques can reduce the amount of data transferred during replication, which can save costs.
- *Monitor and optimize replication frequency*: Regularly review and adjust the replication frequency based on the changing needs of your workload. Fine-tuning the replication frequency can help optimize costs by reducing unnecessary replication overhead.

Optimize backups

A backup is a periodic snapshot or copy of data that you can create and store separately from the primary storage. If there's data corruption, accidental deletion, or system failure, you can use backups to restore the data to its previous state.

Here are some techniques for optimizing backups:

- *Data classification*: Classify your data based on its importance and prioritization for backup. Classification helps you to focus resources on backing up critical data while minimizing backup costs for data that's less important.
- *Incremental backups*: Instead of performing full backups every time, consider implementing incremental backups. Incremental backups capture only changes made since the last backup, which can reduce storage and network bandwidth requirements.

  >  **Tradeoff**: Incremental backups require more steps and time to restore data. You need to restore the full backup first and then apply each incremental backup in sequence until you reach the desired restore point.

- *Backup compression*: Enable compression during the backup process to reduce the size of backup files. Compressed backups require less storage space, so you can save costs.
- *Backup storage tiers*: Evaluate your backup retention policies and consider moving older backups to lower-cost storage tiers, such as cold storage or archive storage. Storing less frequently accessed backups in cost-effective storage options helps optimize costs.

- *Backup retention period*: Review and adjust the retention periods for your backups based on business requirements and compliance regulations. Maintaining backups for longer durations might lead to extra storage costs.

- *Backup frequency*: Analyze the backup frequency for various types of data. Adjust the backup schedule based on the frequency of data changes and the importance of the data. These practices help eliminate unnecessary backups and reduce storage costs.

Optimize file formats

File formats influence cost optimization by optimizing input/output (I/O) patterns and query patterns of your data. Some file formats cater to particular scenarios. Aligning the file format with your workload requirements can improve the workload's performance.

Here are considerations for common formats:

- *Avro*: The Avro file format is a good choice when you're dealing with write-heavy I/O patterns or when query patterns necessitate fetching multiple rows of records in their entirety. Avro's serialization and deserialization processes are efficient, so it's compatible with message buses like Kafka that produce a series of events and messages in quick succession.

- *Parquet and Optimized Row Columnar (ORC)*: The Parquet and ORC file formats excel in scenarios of read-heavy I/O patterns or when the query patterns focus on specific columns of the records.

  Both formats are columnar storage, which means that data is stored column by column rather than row by row. Columnar storage allows for improved compression and efficient read operations. Only the required columns need to be fetched, so you avoid unnecessary I/O for irrelevant data.

Optimize storage solutions

Evaluate and select the most appropriate storage methods and systems for your data. This effort might include switching databases, using different storage types, or adding caching mechanisms. Ease of management is another factor to consider when you're choosing a storage solution.

By tailoring storage solutions to the specific needs and characteristics of the data, you can achieve better cost-effectiveness while meeting performance and scalability demands. There are costs associated with switching databases or swapping services, but storing data in the wrong storage solution can cost you extra money.

Here are a few use cases:

- *Switching databases*: You could consider switching to a database system that better suits your needs. For instance, if you're using a relational database, you might explore the option of moving to a NoSQL database if your data is more document oriented or requires flexible schemas.

- *Moving from a relational database to a flat file store*: In some cases, storing data in flat files instead of a traditional relational database can provide advantages such as simplicity and cost-effectiveness. Flat files are well suited for certain types of data, such as log files or data that doesn't require complex querying. For example, you can store binary images in a SQL database, but it's more cost-effective to store them in a storage service that's specifically for handling binary data.

- *Moving from infrastructure as a service (IaaS) to platform as a service (PaaS)*: IaaS database solutions can be time-consuming and resource-intensive properties that divert a technical team's attention from core tasks. The growth in data volume and the challenges of manual scaling, backups, and infrastructure maintenance can make a PaaS solution more cost-effective and efficient.

- *Adding a cache*: To reduce resource usage on the main database server, consider using a cache solution for caching complex query results. Rightsizing the database server might help in optimizing the cost. With applicable use cases, consider using time to live (TTL) with the cached data to reduce the storage needs and reduce the cost.

- *Query-optimized versus data storage stores*: Query-optimized stores are designed for fast data retrieval and analysis. They focus on quick data ingestion and reads but not frequent updates. They're great for time-series data and rapid access to recent data, but not for heavy transactional tasks.

  Data storage stores handle large volumes of flexible data, especially unstructured or semistructured data. Although data storage stores can support analytics, complex tasks might need specialized databases. They're best for storing lots of variable data like logs or user-generated content in scenarios like NoSQL use cases.

Azure facilitation

Taking an inventory of data: Microsoft Purview is a family of data governance, risk, and compliance solutions that can help your organization govern, protect, and manage your entire data estate. Microsoft Purview solutions provide integrated coverage and help

address the recent increases in remote user connectivity, the fragmentation of data across organizations, and the blurring of traditional IT management roles.

Optimizing data management: Azure Storage and Azure Data Lake Storage have different data access tiers. They also offer data lifecycle management policies that automate data tiering and retention.

You can use a rule-based policy to transition blob data to the appropriate access tiers or to expire data at the end of its lifecycle. This policy allows you to transition blobs from cool (or cold) to hot immediately when they're accessed, to optimize for performance.

Optimizing backups: The Azure Backup service provides multiple capabilities to streamline your backups. It offers features such as native database backup and storage backup through disk snapshots. It supports virtual machine backup, long-term retention, and backup management.

Here are some of the service's features:

- *Monitoring*: You can use Backup center as a single pane of glass to monitor your jobs and backup inventory on a day-to-day basis. Backup center provides an interface to Backup reports, which use Azure Monitor Logs and Azure workbooks.

- *Reports*: Backup reports offer the following capabilities:
  - Allocate and forecast consumed cloud storage.
  - Audit backups and restores.
  - Identify key trends at various levels of granularity.
  - Gain visibility and insights into cost optimization opportunities for your backups.

- *Reserved capacity*: Azure Backup Storage reserved capacity offers you a discount on capacity for backup data stored for the vault-standard tier when you commit to a reservation for either one year or three years. A reservation provides a fixed amount of backup storage capacity for the term of the reservation.

- *Archive tier*: You can use Azure Backup to store backup data, including long-term retention (LTR) backup data, according to the retention needs that your organization's compliance rules define. In most cases, the older backup data is rarely accessed and is stored only for compliance needs. Azure Backup supports the backup of LTR points in the archive tier, in addition to snapshots and the standard tier.

Optimizing storage solutions: Azure has many storage solutions. They offer various features and capabilities to help optimize costs based on your specific requirements. Azure has guidance to help you choose the right data store.

To choose the most suitable storage solution and configuration, it's important to assess your data access patterns, retention needs, and performance requirements. Regularly monitoring and optimizing your storage usage by using tools like Azure Advisor can help you further optimize costs.

Organizational alignment

The Cloud Adoption Framework provides guidance for optimizing data costs for organizational analytics platforms.

For more information, see Data lifecycle management.

Related links

- Recommendations for consolidation
- Microsoft Purview
- Data access tiers
- Data lifecycle management policies
- Azure Backup Storage
- Archive tier
- Choose the right data store
- Data lifecycle management

Cost Optimization checklist

Refer to the complete set of recommendations.

Cost Optimization checklist

Recommendations for optimizing code costs

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Cost Optimization checklist recommendation:

CO:11	**Optimize code costs. Evaluate and modify code to meet functional and nonfunctional requirements with fewer or cheaper resources.**

This guide describes the recommendations for optimizing code costs. Code optimization is the process of improving the efficiency, performance, and cost-effectiveness of application code. Effective code optimization involves making changes to the code to reduce resource consumption, minimize execution time, and improve overall performance.

By optimizing code, you can identify and eliminate inefficiencies that might lead to increased resource consumption and higher costs. You can reduce processing time, memory usage, and network overhead, which can lead to applications that are faster and more responsive. Improved performance enhances the user experience and enables your system to handle larger workloads efficiently.

Definitions

Term	Definition
Code instrumentation	The practice of adding code snippets or libraries to code that collect data and monitor code performance during runtime.
Concurrency	The execution of multiple processes at the same time.
Data serialization	The process of converting data objects into a format that can be stored or transmitted and then reconstructing them back to their original form when needed.
Hot paths	Critical or frequently run sections of a program that require high performance and low latency.

Key design strategies

Cost optimizing code means improving code to achieve the same functionality with fewer per-instance resources, such as CPU cycles, memory, and storage. By reducing

resource consumption, you can save money when applications handle large volumes of data or experience high traffic loads.

Code improvements are most effective when you're following other cost optimization efforts around scaling, rightsizing, redundancy, and throttling. After you take care of these foundational elements, you can consider code optimization.

You might not know if you have inefficient code. Serverless, autoscale, and reliability features can mask code inefficiencies. The following strategies can help you identify and fix application code that costs more than it should.

Instrument your code

Instrumenting code is the practice of adding code snippets or libraries that collect data and monitor code performance during runtime. It allows developers to gather information about key metrics, such as resource consumption (CPU or memory usage) and execution time. By instrumenting code, developers can gain insights into code hot paths, identify performance bottlenecks, and optimize the code for better efficiency and cost-effectiveness.

In an ideal environment, you should do code analysis early in the software development lifecycle. The earlier you catch a code problem, the cheaper it's to fix.

Automate as much of this code analysis as possible. Use dynamic and static tools for code analysis to reduce the manual effort. However, keep in mind that this testing is still a simulation of production. Production provides the clearest understanding of code optimization.

>  **Tradeoff**: Code monitoring tools are likely to increase costs.

Identify and optimize hot paths

By instrumenting your code, you can measure the resource consumption of code paths. These measurements help you identify hot paths. Hot paths have a significant effect on performance and resource usage. They're critical or frequently run sections of a program that require high performance and low latency.

To identify hot paths, consider these tasks:

- *Analyze runtime data*: Collect and analyze runtime data to identify areas of the code that consume significant resources, such as CPU, memory, or I/O operations.

Look for patterns or sections of code that are frequently run or take a long time to complete.

- *Measure performance*: Use profiling tools or performance testing frameworks to measure the execution time and resource consumption of code paths. This measurement helps identify bottlenecks and areas for improvement.

- *Consider business logic and user effect*: Evaluate the importance of code paths based on their relevance to the application's functionality or critical business operations. Determine which code paths are crucial for delivering value to users or meeting performance requirements.

  Review performance recommendations that are specific to the programming language you're working with. Evaluate your code against these recommendations to identify areas for improvement. Remove any unnecessary operations within the code path that might affect performance.

- *Remove unnecessary function calls*: Review your code. Identify any functions that aren't essential for the desired functionality and might negatively affect performance. For example, if a function call performs a validation that happened earlier in the code, you can remove that unnecessary function call.

- *Minimize logging operations*: Logging can be helpful for debugging and analysis, but excessive logging can affect performance. Evaluate the necessity of each logging operation and remove any unnecessary logging calls that aren't critical for performance analysis.

- *Optimize loops and conditionals*: Analyze loops and conditionals in your code. Identify any unnecessary iterations or conditions that you can eliminate. Simplifying and optimizing these structures can improve the performance of your code.

- *Reduce unnecessary data processing*: Review your code for any unnecessary data-processing operations, such as redundant calculations or transformations. Eliminate these unnecessary operations to improve the efficiency of your code.

- *Minimize network requests*: If your code makes network requests, minimize the number of requests and optimize their usage. Batch requests when possible and avoid unnecessary round trips to improve performance.

- *Minimize allocations*: Identify areas where excessive memory allocation occurs. Optimize the code by reducing unnecessary allocations and reusing existing resources when possible.

By minimizing allocations, you can improve memory efficiency and overall performance. Use the appropriate memory management and garbage collection strategies for your programming language.

- *Reduce data structure size*: Assess the size of your data structures, such as classes, and identify areas where reduction is possible. Review the data requirements and eliminate any unnecessary fields or properties. Optimize memory usage by selecting appropriate data types and packing data efficiently.

- *Assess cross-cutting implementations*: Consider the effects of cross-cutting implementations, such as middleware or token checks. Assess whether they're negatively affecting performance.

>  **Tradeoff**: Optimizing code and hot paths requires developer expertise in identifying code inefficiencies. These highly skilled individuals might need to spend time on other tasks.

Evaluate the use of concurrency

Evaluating the use of concurrency involves assessing whether asynchronous processing, multithreading, or multiprocessing can maximize resource utilization and reduce expenses. By using asynchronous processing, multithreading, or multiprocessing, you can handle more tasks with the same resources. However, it's crucial to ensure proper implementation to avoid more overhead and to maintain cost-effectiveness.

To evaluate whether using concurrency is a good fit, you can follow these guidelines:

- *Asynchronous processing*: Asynchronous processing allows nonblocking execution. For example, you can start a process and then pause it to allow a second process to finish.

  Determine the code components or operations that you can run asynchronously. Identify the programming language or framework that you're using and understand the asynchronous programming model that it supports, such as `async`/`await` in .NET or promises in JavaScript.

  Restructure your code to use asynchronous programming constructs by enabling nonblocking execution of tasks. Decouple long-running or I/O-intensive operations from the main execution thread by using asynchronous methods or callbacks. Use asynchronous APIs or libraries that your programming language or framework provides to handle asynchronous workflows.

- *Multithreading*: In multithreading, you run multiple threads of a single process concurrently.

  Identify sections of your code that you can run concurrently and independently. Read documentation or guidelines that are specific to the programming language or framework you're using for multithreading best practices. Create multiple threads or thread pools to handle parallel execution of tasks.

  Implement synchronization mechanisms, such as locks, mutexes, or semaphores, to ensure thread safety and prevent race conditions when code accesses shared resources. Consider using higher-level abstractions, like thread pools or task-based parallelism libraries, to streamline the management of multiple threads and simplify concurrency control.

- *Multiprocessing*: Multiprocessing can have processes run in parallel. It can provide better utilization of multiple CPU cores than multithreading.

  Determine whether the workload or operations in your code lend themselves to parallel processing. Identify the programming language or framework that you're using and explore its multiprocessing capabilities. For example, consider the multiprocessing module in Python or parallel streams in Java. Design your code to split the workload into multiple independent tasks that can be processed concurrently.

  Use multiprocessing APIs or libraries to create and manage parallel processes. Distribute the workload among these APIs or libraries. To enable coordination and data sharing among multiple processes, implement communication mechanisms like interprocess communication (IPC), shared memory, or message passing, depending on your programming language or framework.

Use the right SDKs

For cost optimization, select SDKs that are designed to optimize resource usage and improve performance. It's important to evaluate the features and capabilities of each SDK. Consider its compatibility with your programming language and development environment.

Here's guidance to help choose the best SDKs for your workload:

- *Conduct performance testing*: Compare the resource usage and performance of SDKs through performance testing. Choose the SDK that best meets your needs in terms of resource optimization and performance improvement. Integrate the

chosen SDK into your codebase by following the provided documentation and guidelines.

- *Monitor resource usage and optimize code*: Monitor resource usage with the implemented SDK. Gather insights from monitoring and analysis to optimize your code.

Choose the right operating system

Most coding languages can run on various operating systems, so it's important to evaluate your operating system against cheaper alternatives. If an alternative operating system supports the same or similar functionality at less cost, it's worth considering. By choosing a cheaper operating system, you can potentially reduce the cost of licensing fees and infrastructure costs.

The right operating system can contribute to overall cost optimization for your workload. To choose the right operating system for your workload, try these activities:

- *Evaluate your requirements*: Understand the specific needs of your workload, including the coding languages and frameworks that you're using. Consider any dependencies or integrations with other systems.

- *Consider compatibility*: Ensure the operating system you choose is compatible with your coding languages, frameworks, and any third-party libraries or tools you use. Check the documentation and community support for the operating system to ensure that it has good compatibility with your technology stack.

- *Assess functionality*: Determine if the alternative operating system supports the same or similar functionality as your current operating system. Evaluate whether it provides the necessary features and capabilities that your workload requires.

- *Compare costs*: Compare the costs associated with operating systems. Consider factors such as licensing fees, support costs, and infrastructure requirements. Look for cheaper alternatives that can meet your workload's requirements without compromising functionality.

- *Consider performance and optimization*: Evaluate the performance and optimization capabilities of the alternative operating system. Look for benchmarks, case studies, or performance comparisons to understand how it performs in real-world scenarios.

- *Review security and stability*: Assess the security and stability of the alternative operating system. Look for security updates, patches, and community support to

ensure that the operating system is actively maintained and is secure and stable overall.

- *Consider vendor support*: Evaluate the level of vendor support that's available for the alternative operating system. Check if there are official support channels, documentation, and a community of users who can provide assistance if you need it.

Optimize network traversal

Optimizing network traversal is about minimizing network traffic between workload components. Data transfer often has an associated cost. By minimizing network traffic, you can reduce the amount of data that needs to be transferred while lowering costs.

Analyze your workload and identify any unnecessary data transfers between components. Avoid transferring redundant or duplicate data, and transmit only essential information. For example, if a component repeatedly requests the same data from another component, it's a candidate for optimization. You can refactor your code to reduce unnecessary calls or to batch requests, minimizing the data transferred. Applications might send entire objects or data structures when only a few fields are needed. By optimizing the code to send only the required data, you minimize the size of each data transfer.

Optimize network protocols

Network protocols play a crucial role in the efficiency of network communication. By optimizing network protocols, you can improve the overall efficiency of data transfer and reduce resource consumption.

Consider these suggestions:

- *Choose efficient protocols*: Select protocols that are known for their efficiency in terms of data transfer speed and minimizing overhead. For example, consider using protocols like HTTP/2 over HTTP/1.1. These protocols are designed to improve performance by reducing latency and optimizing data transfer. Use libraries and frameworks in your application to use these protocols.

- *Support compression*: Implement compression mechanisms in your network protocols to reduce the size of data being transferred. Compression can significantly reduce the amount of data transmitted over the network, leading to improved performance and reduced bandwidth usage. Server-side compression is typically enabled in the application code or server configuration.

- *Utilize connection pooling*: Connection pooling allows for the reuse of established network connections to reduce the overhead of establishing new connections for each request. Connection pooling can improve the efficiency of network communication by avoiding the overhead of connection setup and teardown. Choose a connection pooling library or framework and configure to meet workload needs.

- *Implement other optimizations*: Explore other optimizations that are specific to your workload and network environment. For example, you can use content caching, load balancing, and traffic shaping to further optimize network traversal.

Minimize network overhead

Minimize the amount of network traffic and data transfer between components of your workload. By reducing network overhead, you can lower costs associated with data egress and ingress and improve overall network performance.

Consider these techniques:

- *Reduce redundant requests*: Analyze the code to identify any duplicate or unnecessary requests. Instead of making multiple requests for the same data, you can modify the code to retrieve the data once and reuse it as needed.

- *Optimize data size*: Review the data being transmitted between components or systems, and look for opportunities to minimize its size. Consider techniques such as compressing the data before transmission or using more efficient data formats. By reducing the data size, you can decrease network bandwidth usage and improve overall efficiency.

- *Batch requests*: If applicable, consider batching multiple smaller requests into a single larger request. Batching reduces the overhead of establishing multiple connections and decreases the overall data transmission.

- *Use data serialization*: Data serialization is the process of converting complex data structures or objects into a format that can be easily transmitted over a network or stored in a persistent storage system. This strategy involves representing the data in a standardized format, so the data can be efficiently transmitted, processed, and reconstructed at the receiving end.

  Select a serialization format that's compact, fast, and suitable for your workload's requirements.

Serialization format	Description
Protocol Buffers (protobuf)	A binary serialization format that offers efficient encoding and decoding of structured data. It uses typed definition files to define message structures.
MessagePack	A binary serialization format for compact transmission over the wire. It supports various data types and provides fast serialization and deserialization performance.
JavaScript Object Notation (JSON)	A widely used data serialization format that's human-readable and easy to work with. JSON is text based and has broad cross-platform support.
Binary JSON (BSON)	A binary serialization format that's similar to JSON but designed for efficient serialization and deserialization. BSON includes extra data types that aren't available in JSON.

Depending on the serialization format, you need to implement logic to serialize objects or data structures into the chosen format and deserialize them back into their original form. You can implement this logic by using libraries or frameworks that provide serialization capabilities for the format.

Optimize data access

Optimizing data access refers to streamlining the patterns and techniques for retrieving and storing data, to minimize unnecessary operations. When you optimize data access, you can save costs by reducing resource usage, reducing data retrieval, and improving the efficiency of data processing. Consider techniques such as data caching, efficient data querying, and data compression.

Use caching mechanisms

Caching involves storing frequently accessed data closer to the components that require it. This technique reduces the need for network traversal by serving the data from the cache instead of fetching it over the network.

Consider these caching mechanisms:

- *Use an external cache*: One popular caching solution is a content delivery network. It helps minimize latency and reduce network traversal by caching static content closer to consumers.

- *Tune caching parameters*: Configure caching parameters, such as time to live (TTL), to optimize the benefit of caching while minimizing potential drawbacks. Setting an appropriate TTL ensures that cached data remains fresh and relevant.

- *Use in-memory caching*: In addition to external caching solutions, consider implementing in-memory caching in your application. In-memory caching can help utilize idle compute resources and increase the compute density of allocated resources.

Optimize database traffic

You can enhance the efficiency of application communication to the database. Here are some key considerations and techniques for optimizing database traffic:

- *Create indexes*: Indexing is the process of creating data structures that improve the speed of data retrieval. By creating indexes on frequently queried columns, you can significantly reduce the time it takes to run queries. For example, if you have a table of users with a column for usernames, you can create an index on the username column to speed up queries that search for specific usernames.

  Identify the most frequently accessed columns and create indexes on those columns to speed up data retrieval. Regularly analyze and optimize the existing indexes to ensure that they're still effective. Avoid over-indexing because it can negatively affect insert and update operations.

- *Optimize queries*: Design efficient queries by considering the specific data requirements and minimizing unnecessary data retrieval. Start by using appropriate join types (for example, inner join and left join), based on the relationship between tables. Use query optimization techniques such as query hints, query plan analysis, and query rewriting to improve performance.

- *Cache query results*: You can store the results of frequently run queries in memory or a cache. Subsequent executions of the same query can then be served from the cache, which eliminates the need for expensive database operations.

- *Use an object-relational mapping (ORM) framework:* Use ORM features such as lazy loading, caching, and batch processing to optimize data retrieval and minimize database round trips. Use ORM frameworks such as Entity Framework for C# or Hibernate for Java.

- *Optimize stored procedures*: Analyze and optimize the logic and performance of stored procedures. The goal is to avoid unnecessary computations or redundant

queries in stored procedures. Optimize the use of temporary tables, variables, and cursors to minimize resource consumption.

Organize data

Organizing data for efficient access and retrieval involves structuring and storing data in a way that maximizes performance and minimizes resource consumption. It can improve query response times, reduce data transfer costs, and optimize storage utilization.

Here are some techniques for organizing data efficiently:

- *Partition*: Partitioning involves dividing a large dataset into smaller, more manageable subsets called partitions. You can store each partition separately to allow for parallel processing and improved query performance. For example, you can partition data based on a specific range of values or by distributing data across servers. This technique can enhance scalability, reduce contention, and optimize resource utilization.

- *Shard*: Sharding is a technique of horizontally dividing data across multiple database instances or servers. Each shard contains a subset of the data, and queries can be processed in parallel across these shards. Sharding can improve query performance by distributing the workload and reducing the amount of data that each query accesses.

- *Compress*: Data compression involves reducing the size of data to minimize storage requirements and improve the efficiency of data transfer. Because compressed data takes up less disk space, it allows for savings in storage costs. Compressed data can also be transferred more quickly over networks and reduce data transfer costs.

For example, consider a scenario where you have a large dataset of customer information. By partitioning the data based on customer regions or demographics, you can distribute the workload across multiple servers and improve query performance. You can also compress the data to reduce storage costs and improve the efficiency of data transfer.

Optimize architecture

Evaluate your workload architecture to identify opportunities for resource optimization. The goal is to use the right services for the right job.

To reach this goal, you might need to redesign parts of the architecture to use fewer resources. Consider serverless or managed services, and optimize resource allocation. By

optimizing your architecture, you can meet the functional and nonfunctional requirements while consuming fewer per-instance resources.

Use design patterns

Design patterns are reusable solutions that help developers solve recurring design problems. They provide a structured approach to designing code that's efficient, maintainable, and scalable.

Design patterns help optimize the use of system resources by providing guidelines for efficient resource allocation and management. For example, the Circuit Breaker pattern helps prevent unnecessary resource consumption by providing a mechanism to handle and recover from failures in a controlled manner.

Design patterns can help cost optimize code in the following ways:

- *Reduced development time*: Design patterns provide proven solutions to common design problems, which can save development time. By following established patterns, developers can avoid repetitive work and focus on implementing the specific requirements of their applications.
- *Improved maintainability*: Design patterns promote modular and structured code that's easier to understand, modify, and maintain. They can lead to cost savings in terms of reduced debugging and maintenance efforts.
- *Scalability and performance*: Design patterns help in designing scalable and performant systems. Patterns like the Cache-Aside pattern can improve performance by caching frequently accessed data to reduce the need for expensive computations or external calls.

To implement design patterns, developers need to understand the principles and guidelines of each pattern and apply them in the code. Consider identifying the appropriate pattern for a problem, understanding its structure and components, and integrating the pattern into the overall design.

Various resources are available, such as documentation, tutorials, and sample code. These resources can help developers learn and implement design patterns effectively.

Change configurations

Regularly review and update your workload configuration to ensure that it aligns with your current requirements. Consider adjusting resource sizing and configuration settings

based on workload demands. By optimizing configurations, you can effectively allocate resources and avoid overprovisioning to save costs.

Refactor architecture

Evaluate your workload architecture and identify opportunities for refactoring or redesigning components to optimize resource consumption. Consider techniques such as adopting a microservices architecture, implementing the Circuit Breaker pattern, and using serverless computing. By optimizing your architecture, you can achieve better resource utilization and cost efficiency.

Modify resource sizes

Continuously monitor and analyze the resource utilization of your workload. Based on the observed patterns and trends, adjust resource sizing and configuration settings to optimize resource consumption.

Consider rightsizing virtual machines, adjusting memory allocation, and optimizing storage capacity. By rightsizing resources, you can avoid unnecessary costs associated with underutilization or overprovisioning.

>  **Tradeoff**: Reworking code and architecture might not fit with current project schedules and could lead to schedule and cost slippage.

Azure facilitation

Instrumenting code: Azure provides monitoring and logging tools like Azure Monitor, Application Insights, and Log Analytics. You can use these tools to track and analyze the performance and behavior of your code in real time.

Identifying hot and optimize paths: Application Insights and Application Insights Profiler help identify and optimize the hot paths in your code by analyzing execution times and resource usage. You can minimize unnecessary memory allocations and optimize memory usage with Profiler.

Using the right SDKs: Azure offers SDKs in multiple programming languages, optimized for performance and ease of use. These SDKs provide prebuilt functions and libraries that interact with Azure services to reduce the need for custom implementation.

Optimizing network traversal: Various Azure services support high-speed network protocols like HTTP/2 and QUIC for efficient communication between services and

applications.

Azure services, such as Azure Database for PostgreSQL - Flexible Server, support connection pooling.

Azure supports batch processing in various services, so you can group multiple operations together and run them in a single request. Batch processing can significantly improve efficiency and reduce network overhead.

Regarding data serialization, Azure supports various serialization formats, including JSON and XML. Choose the appropriate serialization format based on data size, performance requirements, and interoperability needs.

Optimizing data access: Azure provides caching services like Azure Cache for Redis. You can use caching to store frequently accessed data closer to the application, which results in faster retrieval and reduced back-end load.

- *Indexing and query optimization:* Azure services like Azure SQL Database and Azure Cosmos DB provide indexing capabilities to optimize query performance. By choosing the right indexing strategy and optimizing queries, you can improve the overall efficiency of data retrieval.
- *Object-relational mapping (ORM):* Azure supports ORM frameworks like Entity Framework. These frameworks simplify data access and mapping between object-oriented code and relational or NoSQL databases.
- *Optimizing stored procedures:* You can use Azure services like Azure SQL Database to create and optimize stored procedures. Stored procedures can enhance performance by reducing network round trips and precompiling SQL statements.
- *Partitioning and sharding:* Azure offers partitioning and sharding capabilities in services like Azure Cosmos DB and Azure SQL Database. You can use partitioning to distribute data across multiple nodes for scalability and performance optimization.
- *Compressing data:* Azure services support data compression techniques like GZIP and DEFLATE.

Optimizing architecture: Azure provides architectural guidance and design patterns for designing scalable, resilient, and performant applications. For more information, see Design patterns.

Related links

- Azure Monitor
- Application Insights
- Log Analytics
- Application Insights Profiler
- Connection pooling
- Azure Database for PostgreSQL - Flexible Server connection pooling
- Azure SQL Database index tuning
- Azure Cosmos DB indexing policies
- Azure Cosmos DB partitioning
- Azure SQL Database partitioning

Cost Optimization checklist

Refer to the complete set of recommendations.

Cost Optimization checklist

Recommendations for optimizing scaling costs

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Cost Optimization checklist recommendation:

> **CO:12** **Optimize scaling costs. Evaluate alternative scaling within your scale units. Consider alternative scaling configurations, and align with the cost model. Considerations should include utilization against the inherit limits of every instance, resource, and scale unit boundary. Use strategies for controlling demand and supply.**

This guide provides recommendations for optimizing scaling costs. Cost optimizing scaling is the process of removing inefficiencies in workload scaling. The goal is to reduce scaling costs while still meeting all nonfunctional requirements. Spending less to get the same result. Optimizing scaling allows you to avoid unnecessary expenses, overprovisioning, and waste. It also helps prevent unexpected spikes in costs by controlling demand and capping supply. Inefficient scaling practices can lead to increased workload and operational costs and negatively affect the overall financial health of the workload.

Definitions

Term	Definition
Autoscaling	A scaling approach that automatically adds or removes resources when a set of conditions is met.
Cost metrics	Numeric data related to workload cost.
Scale down	A vertical scaling strategy that shifts to a lower SKU to provide less resources to the workload.
Scale in	A horizontal scaling strategy that removes instances to provide less resources to the workload.
Scale out	A horizontal scaling strategy that adds instances to provide more resources to the workload.
Scale unit	A group of resources that scale proportionately together.
Scale up	A vertical scaling strategy that shifts to a higher SKU to provide more resources to the workload.

Term	Definition
Stock keeping unit (SKU)	A service tier for an Azure service.
Usage data	Usage data is either direct information (real) or indirect/representative information (proxy) about how much a task, service, or application is being used.

Key design strategies

The goal of cost optimizing scaling is to scale up and out at the last responsible moment and to scale down and in as soon as it's practical. To optimize scaling for your workload, you can evaluate alternative scaling options within the scale units and align them with the cost model. A scale unit represents a specific grouping of resources that can be scaled independently or together. You should design scale units to handle a specific amount of load, and they can comprise multiple instances, servers, or other resources. You need to evaluate the cost effectiveness of your workload scale units and model alternates.

If you don't use scaling, see guidance on scaling the workload. You need to figure out if your application can scale. Stateless applications are easier to scale because they can handle multiple requests at the same time. Also, evaluate if the application is built using distributed systems principles. Distributed systems can handle increased load by distributing the workload across multiple nodes. However, a singleton application is designed to have only one instance running at any given time. So scaling might not be appropriate for all workloads.

Evaluate scale out versus scale up

Evaluating scale out versus scale up involves determining the most cost-effective approach between increasing resources in an existing system (scale up) or adding more instances of that system (scale out) based on various factors like pricing, workload requirements, and acceptable downtime. Choosing the right scaling approach can lead to significant savings, ensuring you pay for only what you need while still meeting performance and reliability standards.

The goal is to determine the most cost-efficient choice based on service-tier pricing, workload traits, acceptable downtime, and the cost model. For some, it might be more economical to opt for more expensive instances in fewer numbers. Conversely, for others, a cheaper tier with more instances might be better. To make an informed decision, you need to analyze real or representative data from your setup and evaluate

the relative cost merits of each strategy. To evaluate the most cost efficient approach, consider these recommendations:

- *Gather usage data*: Collect actual production data or proxy data that represents the workload usage patterns and resource utilization. This data should include metrics such as CPU usage, memory usage, network traffic, and any other relevant metrics that affect the cost of scaling.

- *Define cost metrics*: Identify the cost metrics that are relevant to your workload, such as the cost per hour, cost per transaction, or cost per unit of resource usage. These metrics help you compare the cost effectiveness of different scaling options.

- *Gather usage data*: Collect actual production data or proxy data that represents the workload usage patterns and resource utilization. This data should include metrics such as CPU usage, memory usage, network traffic, and any other relevant metrics that affect the cost of scaling

- *Define cost metrics*: Identify the cost metrics that are relevant to your workload, such as the cost per hour, cost per transaction, or cost per unit of resource usage. These metrics help you compare the cost-effectiveness of different scaling options.

- *Refer to requirements*: When deciding between scale-out and scale-up strategies, consider the reliability, performance, and scaling requirements of your workload. Scaling out can improve reliability through redundancy. Scaling up increases the capacity of a resource, but there might be limits to how much you can scale up.

- *Consider resource limits*: When evaluating scaling options, it's important to consider the inherent limits of every instance, resource, and scale unit boundary. Be aware of the upper scaling limits for each resource and plan accordingly. Additionally, keep in mind the limits of your subscription and other resources.

- *Test scaling*: Create tests for different scaling scenarios, including scale out and scale up options. Applying the usage data, simulate the workload behavior under different scaling configurations. Conduct real-world testing using the modeled scaling scenarios.

- *Calculate costs*: Use the gathered data and cost metrics to calculate the costs associated with each scaling configuration. Consider factors such as instance pricing, resource utilization, and any extra costs related to scaling.

Optimize autoscaling

Optimizing the autoscaling policy involves refining autoscaling to react to load changes based on the workload's nonfunctional requirements. You can limit excessive scaling activities by adjusting thresholds and using the right cooldown period. To optimize autoscaling, consider the following recommendations:

- *Analyze the current autoscaling policy*: Understand the existing policy and its behavior in response to varying load levels.
- *Refer to nonfunctional requirements*: Identify the specific nonfunctional requirements that you need to consider, such as response time, resource utilization, or cost.
- *Adjust scaling thresholds*: Adjust the scaling thresholds based on the workload characteristics and nonfunctional requirements. Set thresholds for scaling up or down based on factors like CPU utilization over time, network traffic, or queue length.
- *Adjust a cooldown period*: Adjust the cooldown period to prevent excessive scaling activities triggered by temporary load spikes. A cooldown period introduces a delay between scaling events, allowing the system to stabilize before further scaling actions.
- *Monitor and fine-tune*: Continuously monitor the system's behavior and performance. Analyze the scaling activities and adjust the policy as needed to optimize cost and meet the desired nonfunctional requirements.

>  **Tradeoff**: Reducing the number of scaling events raises the chances of encountering issues related to scaling. It means you're eliminating the extra cushion or buffer that could help manage potential problems or delays from scaling.

Consider event-based scaling

Event-driven autoscaling allows the application to dynamically adjust resources based on specific events or triggers rather than traditional metrics like CPU or memory utilization. For example, Kubernetes event-driven autoscaling (KEDA) can scale applications based on scalers such as the length of a Kafka topic. Precision helps prevent unnecessary scaling fluctuations and resource waste. A high level of precision ultimately optimizes costs. To use event-based scaling, follow these steps:

- *Choose an event source*: Determine the event source that triggers the scaling of your scale unit. A source can be a message queue, a streaming platform, or any other event-driven system.

- *Set up event ingestion*: Configure your application to consume events from the chosen event source. It typically involves establishing a connection, subscribing to the relevant topics or queues, and processing the incoming events.
- *Implement scaling logic*: Write the logic that determines when and how your scale unit should scale based on the incoming events. This logic should consider factors such as the number of events, the rate of incoming events, or any other relevant metrics.
- *Integrate with scaling mechanisms*: Depending on your application's runtime environment, you can use different scaling mechanisms to adjust the resources allocated to the application.
- *Configure scaling rules*: Define the scaling rules that specify how your scale unit should scale in response to events. These rules can be based on thresholds, patterns, or any other criteria that align with your application's requirements. Scaling thresholds should relate to business metrics. For example, if you add two more instances, you can support 50 more users in shopping cart processing.
- *Test and monitor*: Validate the behavior of your event-based scaling implementation by testing it with different event scenarios. Monitor the scaling actions and ensure that the actions align with your expectations.

>  **Tradeoff** Configuring and fine-tuning event-based autoscaling can be complex, and improper configuration might lead to over-provisioning or under-provisioning of resources.

Optimize demand and supply

Control demand against your supply. On workloads where usage determines scaling, cost correlates with the scaling. To optimize the costs of scaling, you can minimize scaling spend. You can offload demand by distributing demand to other resources, or you can reduce demand by implementing priority queues, gateway offloading, buffering, and rate limiting. Both strategies can prevent undesired costs due to scaling and resource consumption. You can also control supply by capping the scaling limits. To optimize workload demand and supply, consider the following recommendations.

Offload demand

Offloading demand refers to the practice of distributing or transferring resource demand to other resources or services. You can use various technologies or strategies:

- *Caching*: Use caching to store frequently accessed data or content, reducing the load on your backend infrastructure. For example, use content delivery networks (CDNs) to cache and serve static content, reducing the need for scaling the backend. However, not every workload can cache data. Workloads that require up-to-date and real-time data, like trading or gaming workloads, shouldn't use a cache. The cached data would be old and irrelevant to the user.

  >  **Tradeoff**. Caching might introduce challenges in terms of cache invalidation, consistency, and managing cache expiration. It's important to carefully design and implement caching strategies to avoid potential tradeoffs.

- *Content offloading*: Offload content to external services or platforms to reduce the workload on your infrastructure. For example, rather than store video files on your primary server, you can host these files in a separate storage service that's independent from your primary server. You can load these large files directly from the storage service. This approach frees up resources on your servers, allowing you to use a smaller server. It can be cheaper to store large files in a separate data store. You can use a CDN to improve performance.

- *Load balancing*: Distribute incoming requests across multiple servers using load balancing. Load balancing evenly distributes the workload and prevents any single server from becoming overwhelmed. Load balancers optimize resource utilization and improve the efficiency of your infrastructure.

- *Database offloading*: Reduce the load on your main application server by offloading database operations to a separate database server or a specialized service. For example, use a CDN for static content caching and a Redis cache for dynamic content (data from database) caching. Techniques like database sharding, read replicas, or using managed database services can also reduce the load.

  >  **Tradeoff:** Offloading specific tasks to alternate resources helps reduce or avoid extra scaling and costs associated with scaling. However, it's important to consider the operational and maintenance challenges that might arise from offloading. Conducting a comprehensive cost-benefit analysis is crucial when selecting the most appropriate offloading techniques for your workload. This analysis ensures that the chosen method is both efficient and feasible in relation to the anticipated savings and operational complexities.

Reduce demand

Reducing resource demand means implementing strategies that help minimize resource utilization in a workload. Offloading demand shifts demand to other resources. Reducing demand decreases demand on the workload. Reducing demand allows you to avoid overprovisioning resources and paying for unused or underutilized capacity. You should use code-level design patterns to reduce the demand on workload resources. To reduce demand through design patterns, follow these steps:

- *Understand design patterns*: Familiarize yourself with various design patterns that promote resource optimization.
- *Analyze workload requirements*: Assess the specific requirements of your workload, including its expected demand patterns, peak loads, and resource needs.
- *Select appropriate design patterns*: Choose the design patterns that align with your workload's requirements and objectives. For example, if your workload experiences fluctuating demand, event-driven scaling and throttling patterns can help manage the workload by dynamically allocating resources. Apply the selected design patterns to your workload architecture. You might need to separate workload components, containerize applications, optimize storage utilization, and more.
- *Continuously monitor and optimize*: Regularly evaluate the effectiveness of the implemented design patterns and adjust as needed. Monitor resource usage, performance metrics, and cost optimization opportunities.

By following these steps and using appropriate design patterns, you can reduce resource demand, optimize costs, and ensure the efficient operation of their workloads.

Use these design patterns to reduce demand:

- *Cache aside*: The pattern checks the cache to see if the data is already stored in memory. If the data is found in the cache, the application can quickly retrieve and return the data, reducing the need to query the persistent data store.
- *Claim check*: By separating data from the messaging flow, this pattern reduces the size of messages and supports a more cost-effective messaging solution.
- *Competing consumers*: This pattern efficiently handles items in a queue by applying distributed and concurrent processing. This design pattern optimizes costs by scaling that is based on queue depth and setting limits on maximum concurrent consumer instances.
- *Compute resource consolidation*: This pattern increases density and consolidates compute resources by combining multiple applications or components on shared

infrastructure. It maximizes resource utilization, avoiding unused provisioned capacity and reducing costs.

- *Deployment stamps*: The use of deployment stamps provides several advantages, such as geo-distributing groups of devices, deploying new features to specific stamps, and observing cost per device. Deployment stamps allow for better scalability, fault tolerance, and efficient resource utilization.

- *Gateway offloading*: This pattern offloads request processing in a gateway device, redirecting costs from per-node resources to the gateway implementation. Using this design pattern can result in a lower cost of ownership in a centralized processing model.

- *Publisher/subscriber*: This pattern decouples components in an architecture, replacing direct communication with an intermediate message broker or event bus. It enables an event-driven approach and consumption-based billing, avoiding overprovisioning.

- *Queue-based load leveling*: The pattern buffers incoming requests or tasks in a queue. The buffering smooths out the workload and reduces the need for overprovisioning of resources to handle peak load. Incoming requests are processed asynchronously to reduce costs.

- *Sharding*: This pattern directs specific requests to a logical destination, allowing optimizations with data colocation. Sharding can lead to cost savings by using multiple instances of lower-spec compute or storage resources.

- *Static content hosting*: This pattern delivers static content efficiently by using a hosting platform designed for this purpose. It avoids the use of more expensive dynamic application hosts, optimizing resource utilization.

- *Throttling*: This pattern puts limits on the rate (rate limiting) or throughput of incoming requests to a resource or component. It helps inform cost modeling and can be tied directly to the business model of the application.

- *Valet key*: This pattern grants secure and exclusive access to a resource without involving more components, reducing the need for intermediary resources and improving efficiency.

Control supply

Defining an upper limit on the amount that you're willing to spend on a particular resource or service is one way to control supply. It's an important strategy for

controlling costs and ensuring that expenses don't exceed a certain level. Establish a budget and monitor the spending to ensure it stays within the defined amount. You can use cost management platforms, budget alerts, or tracking usage and spending patterns. Some services allow you to throttle supply and limit rates, and you should use those features where helpful.

Controlling supply refers to defining an upper limit on the amount that you're willing to spend on a particular resource or service. It's an important strategy because it helps control costs and ensures that expenses don't exceed a certain level. Establish a budget and monitor the spending to ensure it stays within the defined threshold. You can use cost management platforms, budget alerts, or tracking usage and spending patterns. Some services allow you to throttle supply and limit rates, and you should use those features where helpful.

>  **Tradeoff**: Stricter limits might result in missed opportunities to scale when demand increases, potentially impacting user experience. It could cause shutdowns or unable to respond to load. It's important to strike a balance between cost optimization and ensuring that you have sufficient resources to meet your business needs.

Azure facilitation

Evaluating scale-out versus scale-up: Azure provides a test environment where you can deploy and test different scaling configurations. By using the actual workload data or proxy data, you can simulate real-world scenarios and measure the effects on costs. Azure offers tools and services for performance testing, load testing, and monitoring, which can help you evaluate the cost effectiveness of scale out versus scale up options.

Azure provides cost management recommendations through various tools and services, such as the Azure Advisor. These recommendations analyze your usage patterns, resource utilization, and scaling configurations to provide insights and suggestions for optimizing costs.

Azure Load Testing is a fully managed load-testing service that generates high-scale load. The service simulates traffic for your applications, regardless of where they're hosted. Developers, testers, and quality assurance (QA) engineers can use load testing to optimize application performance, scalability, or capacity.

Optimizing autoscaling: Many Azure compute services support deploying multiple identical instances, and rapidly tuning the scaling thresholds and policies. Azure provides autoscaling capabilities that allow you to automatically adjust the number of

instances or resources based on workload demand. You can define scaling rules and thresholds to trigger scale-out or scale-in actions. By using autoscaling, you can optimize resource allocation and cost efficiency by dynamically scaling resources based on actual demand.

Azure maintains a list of subscription and service limits. There's a general limit to the number of instances of a resource you can deploy in each resource group with some exceptions. For more information, see Resource instance limits per resource group.

Optimizing demand and supply: Azure Monitor provides insights into the performance and health of your applications and infrastructure. You can use Azure Monitor to monitor the load on your resources and analyze trends over time. By using metrics and logs collected by Azure Monitor, you can identify areas where scaling adjustments might be needed. This information can guide the refinement of your autoscaling policy to ensure it aligns with the nonfunctional requirements and cost optimization goals.

- *Offloading supply*: Azure has a modern cloud Content Delivery Network (CDN) called Azure Front Door and caching services (Azure Cache for Redis and Azure HPC Cache). The CDN caches content closer to the end-users, reducing network latency and improving response times. Caching stores a copy of the data in front of the main data store, reducing the need for repeated requests to the backend. By using CDN and caching services, you can optimize performance and reduce the load on servers for potential cost savings.

- *Controlling supply*: Azure also allows you to set resource limits for your cloud workload. By defining resource limits, you can ensure that your workload stays within the allocated resources and avoid unnecessary costs. Azure provides various mechanisms for setting resource limits such as quotas, policies, and budget alerts. These mechanisms help you monitor and control resource usage.

  API Management can rate limit and throttle requests. Being able to throttle incoming requests is a key role of Azure API Management. Either by controlling the rate of requests or the total requests/data transferred, API Management allows API providers to protect their APIs from abuse and create value for different API product tiers.

Related links

- Scale the workload
- Azure Advisor Cost recommendations
- What is Azure Load Testing?
- Azure subscription and service limits, quotas, and constraints

- Resources not limited to 800 instances per resource group
- What is Azure Front Door?
- What is Azure Cache for Redis?
- What is Azure HPC Cache?
- Advanced request throttling with Azure API Management

Cost Optimization checklist

Refer to the complete set of recommendations.

Cost Optimization checklist

Recommendations for optimizing personnel time

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Cost Optimization checklist recommendation:

CO:13	**Optimize personnel time. Align the time personnel spends on tasks with the priority of the task. The goal is to reduce the time spent on tasks without degrading the outcome. Optimization efforts should include minimizing noise, reducing build times, high fidelity debugging, and production mocking.**

This guide describes the recommendations for optimizing personnel time. This optimization is a strategic process of maximizing the productivity and efficiency of employees that design, implement, and operate the workload during their working hours. It involves aligning their skills, strengths, and tasks in a manner that ensures that every hour they spend at work is used most effectively. The goal is to eliminate wasted personnel potential and capabilities. Failure to optimize personnel time can lead to employee burnout, reduced competitive edge, and reduced productivity.

Definitions

Term	Definition
Noise	Irrelevant or misleading information that can distract from actual issues or trends.
Signal	Meaningful and relevant information that provides insights into the behavior and performance of a system or application.
Technical debt	The accumulated inefficiencies, suboptimal design choices, or shortcuts intentionally taken during the development process to deliver code faster.

Key design strategies

Personnel typically create the most significant expense in a workload. Personnel cost and value underscore the importance of efficient time management. This guide is about maximizing the potential of every hour worked. Given that employees can't work all day and night, the emphasis is on ensuring that each person is more effective within their designated hours or equally effective in a reduced timeframe. The goal is to achieve better utilization of their time for the benefit of the individual and the workload.

Set optimization targets

Setting personnel time optimization targets is a process of establishing clear, measurable goals. These targets serve as guidelines for desired improvements in tasks and functions. You can use these benchmarks to evaluate outcomes against the targets. First, define the metrics for measuring the success of personnel time optimization efforts. Determine the specific objectives that you want to achieve through optimization. Example objectives might be to reduce time spent on administrative tasks or to reduce the time it takes to respond to customer inquiries. To set targets for personnel time optimization, consider the following strategies:

- *Select quantitative metrics*: Choose metrics that align with your objectives and can be measured accurately. Consider metrics like time saved, productivity increases, efficiency improvements, and task completion time.
- *Gather qualitative metrics*: In addition to quantitative metrics, gather feedback from personnel to measure their satisfaction with their roles. This feedback can provide valuable insights into the effects of personnel time optimization efforts on employee morale and engagement.
- *Set targets*: Set realistic and achievable targets for each selected metric. These targets should be based on the current performance levels and the desired level of improvement.

Optimize development time

Optimizing development involves refining the software development processes to achieve greater efficiency. As a result, developers can invest more time in refining features, innovating within the constraints of a particular workload, and addressing any unique challenges that the workload presents.

Keep features lean

When you design and customize features, keep them lean and simple. Avoid unnecessary complexity and configuration options that can increase the time required to develop, test, and maintain the workload. Keeping the workload simple and focused leads to easier adaptability and optimization over time.

Reduce build times

Reducing build times is the process of minimizing the time it takes to compile and generate a deployment. Shorter build times enable developers to spend less time waiting for builds to finish and allows them to focus on writing code and delivering features. Reducing build times also helps ensure that developers receive feedback on their code changes more quickly. Quicker feedback allows them to iterate and fix issues faster, which supports the Agile development model. Faster build times facilitate more frequent builds, enabling teams to adopt Agile development practices like continuous integration and continuous delivery (CI/CD). Here are some strategies for reducing build times:

- *Optimize build configurations*: Review the build configuration settings and eliminate unnecessary steps or processes that add overhead to the build process. Checkpointing builds and combining partial builds with prebuilt builds can help reduce build times and improve efficiency. This approach enables you to reuse previously built components and build only the necessary parts, which leads to faster build times and reduced time investment.

- *Parallelize build tasks*: Identify tasks that can be run simultaneously and configure the build system to run them in parallel. Take advantage of available computing resources.

- *Use caching*: Cache dependencies, intermediate build artifacts, and other reusable components to avoid redundant work during subsequent builds.

- *Use incremental builds*: To avoid unnecessary recompilation, implement techniques that allow the build system to rebuild only the parts of the deployment that changed since the previous build.

- *Distribute the build process*: If applicable, distribute the build process across multiple machines or build agents to use parallelism and reduce overall build time.

- *Optimize infrastructure*: Ensure that the build environment has sufficient resources, like CPU, memory, and disk I/O, to handle the build.

Use production mocking

By mocking components or services, developers can isolate their code for focused testing by simulating dependencies. Mocking enables developers to create specific scenarios and edge cases that are difficult or impractical to reproduce in a real production environment. It can speed up testing cycles, facilitate parallel work, and eliminate troubleshooting dependencies. Here are some approaches to implementing production mocking:

- *Mocking frameworks*: Use specialized mocking frameworks or libraries that enable you to create mock objects, stubs, or fakes to replace dependencies.
- *Dependency injection*: Design your application to use dependency injection, which enables easy substitution of real dependencies with mock objects during testing or debugging.
- *Service virtualization*: Use service virtualization tools or techniques to simulate the behavior of external services or APIs. Doing so enables developers to test integrations without accessing the real services.
- *Configuration-driven mocking*: Implement a configuration-driven approach in which the application's behavior can be modified via configuration settings or flags to enable mocking as needed.
- *Dynamic and conditional mocking*: Design the application to support dynamic and conditional mocking, which enable developers to switch between real and mock components depending on specific conditions or scenarios.

Optimize the development environment

The goal is for developers to get fast feedback on changes. Make necessary technology changes to improve the development experience.

Containerization: Consider containerizing the workload to run locally. Containers help developers replicate the production environment locally and test their changes quickly. They enable faster iteration and debugging, which leads to a more efficient development process. Containers also provide a consistent and isolated environment for running the application. Finally, they enable easy scaling and deployment of the application.

Developer workstations: An optimal developer workstation should have a suitable integrated development environment (IDE). A good developer workstation boosts developer efficiency, reducing the time and resources needed for various tasks. A good IDE provides code completion and syntax highlighting tailored to the programming language. It should also support version control like Git. A well-equipped IDE enables developers to pinpoint and fix issues quickly during development, which reduces debugging time.

Developer environments: Developers' environments shouldn't be too constrained. Developers should have the permissions necessary to complete tasks without undue restrictions so they can work efficiently and effectively.

Optimize preproduction environments

In general, the closer preproduction environments are to production environments, the more time you save. This increased consistency also helps to minimize risk. The closer the two environments are, the better you can test and validate the functionality and performance of your releases before deploying them to the production environment. This similarity in environments helps you identify and address any issues or bottlenecks early on, which reduces the risk of problems occurring in the production environment.

>  **Tradeoff**: You need to balance personnel time against resource costs. The closer an environment is to the production environment, the more it costs.

Reuse components and libraries

Reusable components and libraries can save developers substantial amounts of time. Instead of writing, testing, and debugging code, developers can reuse validated components and libraries and develop or fix application features faster. Be sure to provide documentation for each component or library. Store the code and documentation in a central repository that has version control like GitHub.

Additionally, use open-source software or libraries from trusted publishers that are available in package managers, like NuGet or Maven. These package managers provide a centralized and reliable source for accessing and managing libraries. Using trusted libraries from package managers can further enhance productivity and reduce the time spent on developing and maintaining code.

Remove technical debt

Removing technical debt is essential for maintaining a healthy and efficient codebase. By following specific standards and implementing mechanisms like quality gates, you can effectively address technical debt and improve the overall quality of your code. Here's how you can incorporate this guidance into your approach:

- *Allocate time to resolve technical debt*: Dedicate a portion of your development team's time to resolving technical debt. A good starting point is to allocate about 20% of the team's time specifically to addressing technical debt. The dedicated time enables developers to focus on refactoring, code cleanup, and improving the overall quality of the codebase.

- *Empower the development team*: Allow the development team to own the prioritization of technical debt resolution. The development team is in the best position to identify areas of the codebase that require attention and understand

the effects of technical debt on workload functionality. Encourage open communication and collaboration within the team to ensure that technical debt is addressed effectively.

- *Prioritize*: Prioritize technical debt items based on their effects on workload functionality. Focus on addressing the issues that have the most significant effect on the performance, maintainability, and scalability of the workload. By prioritizing effectively, you can maximize the effects of your efforts to remove technical debt.

Removing technical debt is an ongoing process. It requires a proactive approach and continuous effort from the development team. By setting and adhering to specific standards in the codebase and implementing mechanisms like quality gates, you can effectively address technical debt and create a cleaner, more maintainable codebase:

- *Set coding standards*: Establish clear and specific coding standards that define the desired structure, style, and best practices for your codebase. These standards should cover areas like naming conventions, code formatting, documentation, and error handling. By adhering to these standards, you ensure consistency and readability throughout the codebase.

- *Implement quality gates*: Quality gates are mechanisms that enforce the defined coding standards and catch potential issues early in the development process. They can include automated code reviews, static code analysis tools, and continuous integration pipelines. By integrating quality gates into your development workflow, you can identify and address code quality issues before they become technical debt.

Optimize personnel collaboration

Optimizing personnel collaboration is a process of enhancing team dynamics, communication, and knowledge-sharing. The goal is to prevent misunderstandings, duplicated efforts, and wasted time. It involves breaking down silos, revising unnecessary standards, creating shared knowledge repositories, and investing in relevant training. Effective collaboration reduces repeated errors and maximizes the collective expertise of a team. To optimize personnel collaboration, consider the following strategies:

- *Eliminate silos*: Silos can lead to a lack of shared knowledge and unnecessary replication of tasks. Cross-functional collaboration can save time and improve results. Break down barriers between departments or teams to promote inter-departmental cooperation. Foster cross-departmental meetings, workshops, and joint projects. Encourage open communication channels across teams.

- *Optimize standards*: Unnecessary standards can lead to wasted time and resources without contributing to better outcomes. Assess, improve, or eliminate standards or protocols that don't add value but increase the workload. Periodically review standards and protocols. Get feedback from ground-level employees. If a standard doesn't add value, consider eliminating or revising it.

- *Create a shared knowledge repository*: A shared knowledge base prevents repeated mistakes, aids training, and reduces the time spent searching for information. Develop a centralized place where all members can access and contribute to collective knowledge. Employ knowledge management tools, regularly update the repository, and incentivize contributions from team members.

- *Invest in training*: Make a substantial investment in training for the processes, tools, and project. Doing so ensures that a baseline requirement is met before people start contributing to the project. Ensure that teams are trained on the established standards and processes to enable them to work efficiently and effectively within the defined guidelines. Team members should be trained on those standards and processes so that they don't waste effort identifying them on their own.

Optimize processes

Optimizing processes involves refining workflows to eliminate unnecessary steps, reduce manual effort, and streamline roles and change management. This enhancement ensures that tasks are more efficient. Streamlined processes reduce the time and resources needed for tasks. The time reduction leads to improved productivity and saves money. To optimize processes, consider these recommendations:

- *Refine the software development lifecycle (SDLC) approach*: Adopting an optimal SDLC can help you achieve high quality with less overhead. Assess your current SDLC method and consider more efficient alternatives. Explore and adopt methodologies like Scrum, Kanban, or Waterfall. Periodically reassess chosen frameworks for better efficiency, recognizing that SDLC is inherently collaborative.

- *Optimize per role*: Defined roles ensure clear responsibilities and expectations and increased efficiency. Understand and define the requirements of each role, including, for example, developers and solution architects. When you want to expand the team, you should know what each role needs in terms of hardware, licenses, and access.

- *Streamline change management*: Positive receptiveness to change ensures smoother transitions and better outcomes. Make the process of implementing change smooth and accepted. Cultivate a culture of active participation rather than

resistance. Promote change adoption via coaching and continuous learning. Adapt to change constructively.

Optimize operational tasks

Optimizing workload operational tasks is a process of making job tasks faster and more straightforward. The goal is to streamline activities to enhance efficiency and ensure the most effective use of resources. This streamlining ensures that tasks are completed with fewer errors, distractions, and delays. It conserves personnel time, which leads to faster decision-making, reduced troubleshooting durations, and overall improved efficiency and cost savings. To optimize operational tasks, consider the following strategies.

Reduce the noise-to-signal ratio

Distinguishing signal from noise is crucial to observability because it enables teams to focus on the most critical aspects of their systems and applications. Filtering out noise can help teams make informed decisions, troubleshoot problems, and optimize the workload faster. Identifying and addressing issues more efficiently and quickly leads to a reduction in personnel costs.

To differentiate signal from noise, you need to define clear objectives and metrics. Identify the key performance indicators (KPIs) and metrics that are relevant to your workload. Establish thresholds or ranges for each metric to specify normal behavior and what should be flagged as an anomaly. Use monitoring tools to collect data and track the defined metrics in real time and identify patterns that indicate potential issues or areas of improvement.

Prioritize actionable insights. Focus on insights that point to degradations in the workload and prioritize them for further investigation or action. Regularly review and update your monitoring strategy based on feedback.

Use high fidelity debugging

High fidelity debugging refers to the ability to accurately diagnose and fix issues in software applications. You gain detailed insights into the application's behavior and state during runtime. High fidelity debugging is crucial for effective software development and troubleshooting. With high fidelity debugging, developers can reproduce and analyze issues with greater precision, which reduces the time and effort required to fix bugs. An understanding of the application's behavior enables developers to make informed decisions faster to improve code quality.

- *Use a debugging tool*: Use a feature-rich debugger that provides comprehensive insights into the application's execution flow, variables, and memory state.
- *Enable detailed logging and tracing*: Instrument code with logging and tracing statements to capture relevant information during runtime. Doing so helps you diagnose issues.
- *Analyze error messages and stack traces*: Carefully examine error messages and stack traces to understand the context and sequence of events leading to an issue.

Enhance technical support

Improve the efficiency and efficacy of technical support operations. Reducing recurring issues saves time and improves user satisfaction. Identify recurring support issues, integrate engineering and support teams via support shadowing, and adopt IT classic deployment model processes to reduce overall support load.

Learn from incidents

Analyzing incidents can prevent recurrence and improve reaction times. Use past incidents as learning opportunities for future improvement. Conduct retrospectives to analyze incidents, identify improved actions and contact protocols, and enhance system observability through comprehensive logs and metrics.

Implement robust governance

Standardization reduces errors and rework to ensure consistent quality and cost optimization. Strengthen compliance and standardization within your organization. Automate compliance checks, and advocate for standardized solutions, architectures, and blueprints. To streamline decision-making, minimize choices that don't align with organizational constraints or SLAs.

Optimize personnel skills

Better skills lead to increased efficiency and fewer mistakes. Invest in the development and improvement of your team's skills. To optimize personnel skills, here are some recommendations to consider:

- *Upskilling*: Ensure that team members have essential cost optimization and monitoring skills. Provide sandbox environments for hands-on learning and skill

development. Encourage team members to obtain certifications, and promote shadowing with experienced colleagues.

- *Tools*: Proficiency with tools is a key skill for optimizing tasks and gaining valuable insights for cost management. Ensure that personnel are proficient with essential tools and can adapt to new ones. Prioritize familiarity with key tools, especially tools that are related to monitoring. Train personnel to extract meaningful insights from data across various layers of the system, emphasizing the link between effective monitoring and cost management.

- *Aligned expertise*: Match employees to tasks based on their skills and expertise. Utilize their strengths and allocate tasks accordingly to maximize efficiency.

Azure facilitation

Setting optimization targets: Azure DevOps provides a suite of tools for defining objectives, selecting metrics, and setting targets. It offers features like work item tracking, dashboards and reporting capabilities. It also provides source code management, continuous integration, continuous delivery, and project management features. By using Azure DevOps, teams can automate processes, collaborate effectively, and reduce manual effort.

Optimizing development time: Azure provides various tools and features to optimize developer time, including:

- *Development environments*: Azure offers development environments in multiple forms, like Microsoft Dev Box, which provides Windows and Linux VMs on which developer tools are installed. Microsoft also provides Docker VMs for containerized development and Azure Container Registry, which enables Docker builds.

- *Integration with Azure DevOps*: Azure integrates with Azure DevOps to enhance productivity and streamline development processes.

- *IDE integration*: Azure provides IDE integration with popular development tools like Visual Studio and Visual Studio Code. This integration enables developers to seamlessly work with Azure services.

- *Standard SDKs and libraries*: Azure provides standard SDKs and libraries for all Azure services. These SDKs and libraries enable developers to reuse code and reduce the time it takes to integrate and implement Azure services.

- *Quickstart templates and samples*: Azure provides quickstart templates and samples that can accelerate the development process.

- *Package managers and standard libraries*: Azure supports package managers and provides standard libraries, like the NuGet package manager. They can simplify development and help developers reduce the time they spend on implementing common functionalities.

- *Open-source support*: Azure has a strong ecosystem that supports open-source technologies, so developers can use existing open-source tools and frameworks to optimize their time.

These features and tools provided by Azure help developers save time and increase productivity in their development workflows.

Optimizing operational tasks: Azure supports infrastructure as code (IaC) capabilities, which enable you to define and manage your infrastructure by using code. Doing so helps reduce complexity and improves the adaptability of your systems.

Azure Monitor is a comprehensive monitoring service that provides visibility into the performance and health of applications and infrastructure on Azure. You can use it to collect telemetry, set up alerts, and gain real-time insights. By using Azure Monitor, you can proactively identify and resolve issues. It enables you to reduce the time you spend on troubleshooting.

Azure Automation provides a way to automate manual, repetitive tasks on Azure. You can use it to create and manage runbooks, which are sets of instructions for performing specific tasks. By automating routine tasks, you can save time and free up personnel to focus on more critical activities.

Optimizing personnel skills: Microsoft provides a comprehensive suite of training materials and activities. Training is available for developers, architects, and business stakeholders.

Related links

- Azure DevOps
- Azure Automation
- Microsoft training

Community link

- McKinsey & Company: Measure software developer productivity

Cost Optimization checklist

Refer to the complete set of recommendations.

Cost Optimization checklist

Recommendations for consolidation

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Cost Optimization checklist recommendation:

> **CO:14** **Consolidate resources and responsibility. In a workload, determine ways to consolidate resources and increase density. Outside a workload, use existing centralized resources and services, so you can consolidate workload responsibilities.**

This guide describes the recommendations for consolidating resources and responsibilities to optimize workload costs. Consolidating resources is a nuanced task that differs from simply eliminating waste. Consolidation involves combining components of a workload, such as servers, databases, applications, and responsibilities.

Consolidation can reduce redundant resources and licenses, and increase density. Look for opportunities to consolidate your workload responsibilities. Use centralized resources or teams to optimize costs. If you don't consolidate resources and responsibilities by using shared resources and optimizing economies of scale, you might miss opportunities for cost savings.

Definitions

Term	Definition
Centralized resource	A shared resource that multiple components use, rather than each component having its own dedicated resource.
Change control	A structured methodology for managing and implementing changes.
Consolidate	The act of combining components to optimally meet workload requirements.
Resource density	A measure of logical separation within a resource. Increased density typically equates to higher utilization due to the collocation of disparate components, consumers, or environments.

Key design strategies

The primary objective of consolidation is optimization, not reduction. Consolidation involves restructuring workloads, resources, and team roles to achieve maximum cost

efficiency. Unlike optimizing component costs, consolidation is a process that requires careful consideration.

Almost every consolidation effort has tradeoffs and potential risks but can significantly reduce costs. It's important to analyze the potential benefits and associated tradeoffs. All consolidation strategies follow these steps:

1. *Assessment*: Perform a thorough evaluation to identify areas where consolidation might be advantageous.
2. *Identification and evaluation*: Pinpoint and assess potential consolidation targets to determine whether potential cost benefits and tradeoffs justify the effort of consolidating.
3. *Communication and implementation*: If you determine that consolidation is beneficial, announce the impending changes and apply them.

Consolidate resources

Consolidating resources involves combining resources within a workload. You can collocate functionalities or consumers. For example, you might consolidate three web servers into a single server or three databases into a single database server. You might consolidate multiple firewalls into a single firewall that serves multiple environments.

The aim is to increase resource density, so you can maximize the cost efficiency of each resource. Expand the use of a resource and minimize resource redundancy.

Common types of services that you can consolidate include application platforms, databases, network appliances, gateways, and distributed denial-of-service (DDoS) protection. To consolidate resources within a workload, consider the following recommendations:

Assess the workload resources. Assess the existing workload and its resource utilization. Analyze factors such as CPU usage, memory usage, storage capacity, and network bandwidth. Identify areas in which consolidation might be beneficial. Consolidation might involve optimizing resource allocation, eliminating redundant or underutilized resources, or reconfiguring the workload to run more efficiently. Consider factors such as workload dependencies, performance requirements, and scalability.

Identify a consolidation target. Choose a resource to consolidate. It can be an existing resource or a new resource created within the workload. Identify existing resources that you might use for consolidation. For example, you might have servers that can accommodate some of the workload components. If no existing resources meet the

consolidation requirements or if it's more beneficial to consolidate a new resource, consider creating a new resource.

Evaluate the consolidation viability. Ensure functional and technical requirements, such as CPU, memory, and growth, support consolidation. Avoid compromising requirements like performance, reliability, and security. For example, don't create an undesired cross-regional dependency or consolidate resources across preproduction and production environments.

Estimate the cost. Determine the effort and potential complications of consolidation. You should calculate costs, including resource, licensing, and operational expenses. Consider the implications, such as potential challenges in resource monitoring due to consolidation.

Communicate and coordinate with your team. Ensure that you inform all stakeholders about upcoming changes and necessary actions that they need to take. Coordinate with teams to avoid conflicts and ensure a smooth implementation.

>  **Risk**: Consider the effects of resource density, such as noisy neighbors, scale-unit effects, and reduced redundancy. Resource consolidation is often too risky for mission-critical and business-critical workload flows.

>  **Tradeoffs**:
>
> - Resource consolidation reduces isolation and can create a noisy neighbor scenario in a workload. Find other ways to implement logical isolation and increased capacity for the hosting environment. For example, increase firewall capacity if it supports multiple workloads.
> - Consolidation eliminates segmentation and can increase security risk, which makes it easier for attackers to move horizontally. It also makes some compliance standards hard to achieve. Prioritize compliance over consolidation.
> - Resource consolidation results in less redundancy. Carefully plan to ensure that you have the proper amount of reliability in the workload.

Consolidate responsibilities

The goal of consolidating workload responsibilities is to reduce the workload team's responsibilities. It’s a strategic cost optimization effort that requires organizational awareness and collaboration outside the workload team.

There are two principal ways to consolidate your workload team's responsibilities. You can use external shared or centralized resources and not run that resource in the workload environment. You can also offload workload responsibilities to other teams in your organization, so your team isn't directly responsible for those tasks or personnel.

Use external centralized resources

External centralized resources refer to shared resources outside the workload environment. For example, an organization might have a centralized gateway that serves multiple workloads. The goal of external centralized resources is to minimize duplication and overhead. Instead of having a dedicated resource for your workload, you can use a shared resource to optimize costs. Consider the following recommendations:

- *Assess the workload resources*. Evaluate the current state of the workload, and identify areas in which consolidation might be beneficial.
- *Find external opportunities*. Survey your organization for pre-existing centralized resources. These resources might be potential solutions for your workload. For example, you can use a shared security information and event management (SIEM) instead of setting up an independent SIEM tool.
- *Consider change control*. Understand the process of managing changes to the centralized resource. Consider the approval workflow, testing protocols, and deployment methods. Analyze potential challenges if you reduced control of resource modifications.
- *Estimate the cost*. Before you implement centralized resources, clearly quantify the expected savings against the costs that are associated with a transition. Weigh the cost-saving benefits against risks to make an informed decision.
- *Communicate and coordinate with your team*. Establish a mechanism for continuous feedback among teams to address concerns, improve collaboration, and refine processes.
- *Document and track changes*. Maintain detailed documentation of all approved changes, including their scope, implementation steps, and associated risks or issues. Use a centralized system or change-management tool to track and monitor the status of changes throughout their lifecycle.

>  **Tradeoff**: Over-consolidation can result in resource contention, which can lead to performance issues. Consolidation might limit the flexibility and agility of

> individual teams and workloads because they must adhere to centralized standards that can inhibit customization.

Offload responsibilities to external teams

Offloading workload responsibilities to external teams refers to using expert centralized teams that perform specialized services such as a security operations team. You can offload responsibilities to existing teams to help optimize costs and delegate expertise for specific functions.

- *Evaluate team skills*. Assess the current skill set of your team. Identify skill gaps or areas in which a centralized team optimizes costs.
- *Find available opportunities*. Explore your organization for available services, such as the services of a security operations team. Ensure that the centralized team can accommodate the added responsibilities without compromising quality.
- *Consider change control*. Familiarize yourself with how the centralized team handles changes, like approval workflows, testing protocols, and deployment strategies. Determine potential challenges that might arise if you have less direct control of these functions.
- *Communicate and coordinate with your team*. Ensure that teams are familiar with each other's processes, tools, and expectations. Consider a phased transition or pilot period to ease the shift and identify potential challenges early.
- *Document and track changes*. Maintain detailed documentation of all approved changes, including their scope, implementation steps, and associated risks or issues. Use a centralized system or change-management tool to track and monitor the status of changes throughout their lifecycle.

Azure facilitation

Density support: Many Azure services support increased resource density. The following table shows a sampling of these services.

Azure service	Segmentation control
Azure Front Door	Customer domains and URL paths
Azure Firewall	Network and application rules
Azure Application Gateway	Listeners, URL path-based routing

Azure service	Segmentation control
API Management	API policies
Azure Kubernetes Service (AKS)	Namespaces, node pools
Azure App Service	Multiple web apps and APIs on an App Service plan
Azure SQL Database	Multiple databases on a server

Resource observability: Azure Monitor provides a centralized platform for monitoring and managing the performance and health of your Azure resources. You can collect and analyze telemetry data, set up alerts, and gain insight into resource utilization and opportunities for consolidation.

Log Analytics provides centralized log management and analysis. You can collect, analyze, and visualize log data from various Azure resources, which helps to identify issues, troubleshoot problems, and gain operational insight.

Related links

- Azure Monitor
- Recommendations for optimizing component costs

Cost Optimization checklist

Refer to the complete set of recommendations.

Cost Optimization checklist

Operational excellence quick links

Apply operational excellence guidance to your workload to ensure workload quality through standardized processes and team cohesion.

Learn key points

QUICKSTART

Design principles

Checklist

Tradeoffs

Operational excellence patterns

Azure Well-Architected Review assessment

TRAINING

Operational excellence

VIDEO

Achieve Operational Excellence

Review design principles

CONCEPT

Embrace DevOps culture

Establish development standards

Evolve operations with observability

Automate for efficiency

Adopt safe deployment practices

Start with the fundamentals

HOW-TO GUIDE

Start with a DevOps culture

Formalize operational tasks

Formalize software development and management

Standardize tools and processes

Use safe deployment practices (SPD)

Use automation

HOW-TO GUIDE

Automate pragmatically

Design for automation

Ship safely and support what you ship

HOW-TO GUIDE

Design for observability

Use infrastructure as code

Design a reliable workload supply chain

Handle deployment failures

Design an emergency response strategy

Explore related resources

REFERENCE

Azure Advisor: Operational excellence recommendations

DevOps resource center

Azure DevOps

GitHub

Azure Resource Manager

Terraform on Azure

Operational Excellence design principles

Article • 05/06/2025

At the core of the Operational Excellence pillar are DevOps practices that **ensure workload quality through standardized workflows and team cohesion**. This pillar defines operating procedures for **development practices, observability, and release management**. The goal is to minimize process variance, chances of human error, and disruption to customers. To assess your operational health, start with these questions:

- Do you execute operations with discipline?
- Are customers using the workload with maximum predictability?
- How do you learn from experience and collected data to drive continuous improvement?

Workload operations can devolve into chaotic practices when there's no clear ownership or leadership. In this type of environment, teams often resort to methods that are executed with high effort and produce low outcomes, which leads to poor user experience. These approaches meet only short-term goals. Long-term benefits are realized through **continuous evaluation and strategic investments**.

The design principles provide guidelines for operational strategies that must be considered to **address the underlying causes and not just treat symptoms**. Start with the recommended approaches, and then observe what works and what doesn't to identify areas of improvement. After you set your strategy, continue to drive action by using the Operational Excellence checklist.

The operational requirements of a workload are as important as its business requirements. Efficient processes ensure the workload achieves business outcomes within the constraints of compliance, whether that compliance is organizational or external. The key is to find repeatability with consistency.

The goals of the Operational Excellence pillar are **to do the right thing, to do it the right way, and to solve the right problems as a team**.

If you meet these goals, workloads will run reliably and predictably even during times of change. **Inability to fulfill operational requirements can lead to failed deployments**, inconsistent user experience, and added costs that could have been avoided through proper planning and streamlined execution.

Embrace DevOps culture

Expand table

Empower development and operations teams to continuously improve their system design and processes by working together with a mindset of collaboration, shared responsibility, and ownership.

DevOps is a community of practice where diversity of perspective and skills drives toward one mission. Teams must **foster a collaborative environment of shared knowledge** instead of siloed learning. Use shared functions to strive to overcome resource constraints.

A good DevOps culture thrives on shared responsibility. Development and operations teams should align their goals and priorities with the expectations of their customers and keep business focus in mind. The development team should involve the operations team in the feedback loop so the improvements are driven upstream and other teams benefit equally. Conversely, operations teams are responsible for making the development team successful in their business outcomes by sharing resources and feedback that are relevant to the workload.

At the same time, DevOps practices **apply clear lines of ownership and accountability to each team**. Regardless of where the application runs, the workload team is responsible for that application.

DevOps optimizes operational tasks so that they're effective but not burdensome. To reap the full benefit of DevOps, the culture should optimize processes through technology and have processes for people in the organization to promote transparent communication.

Expand table

Approach	Benefits
Use common systems and tools that promote a collaborative environment for communication and tracking progress.	Common tools and processes enable **transparent communication**. Both development and operations teams benefit from situational awareness across various environments, common support issues, and overall challenges and wins. Teams will already be familiar with existing escalation paths if there's an incident. A shared backlog makes priorities, such as working on new features or fixing bugs, clear.
Build a **continuous learning and experimentation mindset** throughout the development cycle. **Support knowledge sharing** across teams and maintain documentation for reuse.	Through experimentation mechanisms, such as A/B testing and developing proofs of concept, you can encourage innovation while keeping costs low. Share knowledge through collaboration that makes the team proficient in design approaches, tooling,

Approach	Benefits
Conduct blameless analysis and debrief post-release and/or post-incident reviews.	and processes. Doing retrospectives after a project helps **identify areas for improvement** and celebrate success.
Adopt proven industry agile practices that focus on action optimization. Look for **opportunities to "shift left"** in operations for manual and automated processes, deployment and quality assurance practices, and observability.	Agile development practices lead to shorter release lifecycles, which are an indicator of business value. Detecting, resolving, and thereby preventing issues earlier is often less intrusive to the process.
Set standards for all development and operational procedures and review and validate them at a regular cadence. These procedures include routine tasks, out-of-band processes, emergency drills and situations, choice of tooling, monitoring procedures, skilling plans, and even communication with stakeholders and customer disclosures. **Be intentional and explicit about your decisions**.	Standards add predictability to operations and make processes and practices scalable. Validating standards is a great way to draw points of improvement. Be prepared for emergency and recovery situations by conducting regular drills. Execute with precision and enable governance to **prevent anomalies** that lead to risks.
Take advantage of centralized operations teams with **specialized skills** and breadth of experience.	There's a cost benefit to using shared resources both for operations and resources. Although you own your workload, the centralized team helps you with cross-functional skills, such as incident management, a proactive perspective on monitoring, and outsourcing expertise with trust.

Establish development standards

Expand table

Optimize productivity by standardizing development practices, enforcing quality gates, and tracking progress and success through systematic change management.

The development team is responsible for addressing workload issues prior to release with minimal friction. Be mindful of developer efficiency and **optimize for fast turnaround cycles**, from coding to testing results. Implement effective and right-sized processes that plan and standardize technical activities and also drive consensus within the team and the stakeholders.

Expand table

Approach	Benefits
Document workload features and capture customer benefits. Derive **scope and detailed functional and nonfunctional requirements** of the architecture. **Create sizing estimation models** to report on scope and cost of the tasks involved.	Good specifications **cut operational costs and chances of failure** by supporting more productive and streamlined development cycles. Developers understand the **technical design, goals, and completion criteria** before they start the coding cycle. Good documentation facilitates repeatable **communication** and **onboarding** of new team members.
Use an **industry standard software development methodology that's appropriately tuned** for the needs of your workload and team size. **Maintain a backlog that's shared among all roles**.	Adoption of a well-known methodology **sets the rhythm of the project**. It removes process ambiguities by giving team members clear expectations and accountability. By tracking against a common list, **tasks can be refined and prioritized** with standard practices. The project will have better chances of being delivered on time. Standard methodologies help with **risk management**. With granular milestone reviews, developers can address potential issues before they become showstoppers.
Use unified source control for all code, scripts, deployment templates, pipeline definitions, and related documentation. The branching strategy must support friction-free release of independent and interdependent features, bug fixes, and hotfixes. Use shared knowledge across the organization to build your branching strategy and deployment processes.	Proper use of source control is crucial in supporting **concurrent changes** and versioning. Maintain a repeatable workflow for releasing changes of various sizes and risks, conduct peer reviews as part of the process, and keep an audit trail.
Have **quality assurance** processes that emphasize testing early in the development lifecycle. **Include all artifacts for planned test procedures**, including application components, infrastructure, and data plane operations that are part of a feature release or update.	Quality assurance ensures that functional and nonfunctional requirements were met with confidence, which leads to positive customer impact. Having test plans ensures quality and completeness and takes possible failure cases into

Approach	Benefits
Treat artifacts as immutable when they're promoted through environments, gaining confidence each time they pass through a quality gate. Where practical, automate routine checks.	consideration. With quality gates, you can enforce best practices to reduce risks. Immutability brings confidence because it ensures the system that you test is exactly what you release. Testing cycles efficiently block progress unless quality criteria are met.
Drive consistency by using **style guides and tools**, which enforce **conventions**, and adopt a **common tool chain** for development, testing, and communication with stakeholders. **Technology standards for developers should** necessitate implementation **of patterns**, API design, **logging, exception handling, and other processes**.	Consistency in code drives readability and easier maintenance. It also reduces complexity and enables code reuse. Common tooling and conventions also help teams optimize processes without the need to address one-off choices.
Consistently and deliberately insist on developer documentation of code as its written.	Clear code documentation ensures that logic and functionality are easily understood when old code needs to be revisited or when development teams rotate.
Report progress and trends to measure efficiency.	Trends in bugs, failed updates, time to deploy, feedback loops, and other metrics are published, and that drives improvements.

Evolve operations with observability

Expand table

Gain visibility into the system, derive insight, and make data-driven decisions.

Build a culture that continuously **improves quality by monitoring the workload** and taking all the pillars of the Azure Well-Architected Framework into consideration. Enable the team and stakeholders to make both short-term and long-term decisions across many facets by providing the necessary data, statistics, and trends. Learn from your data and drive improvements.

Operations built for the purposes of observability are key in proactive maintenance of the application, quality and security assurance, capacity planning, and product management.

A crucial aspect of monitoring is application **using health modeling to help you anticipate issues before they become incidents** and affect customer experience. Efficient monitoring reduces reactive cycles spent on incident management.

Expand table

Approach	Benefits
Build a monitoring system with its own stack and flows. Treat the monitoring system as a dimension of the workload that's decoupled from its utility. The stack must cover all layers, including infrastructure, application health, and build and release processes. **Capturing or sampling business data** is out of scope for observability implementations.	Decouple monitoring and workload stacks to **separate functional requirements and observability requirements** and make independent evolution possible. Changes in code shouldn't affect monitoring, and vice versa. Because observability requirements are separate from functional requirements, **business data won't be disrupted** by monitoring configuration changes or outages.
Drive consistency in the collection process for each **type of data source**. Standardize instrumentation in code by using industry standards for telemetry, collection of infrastructure metrics, and tooling.	Consistency prevents variance in sensing and measurement because familiarity across similar resources **reduces time spent correlating and analyzing data**. You have a holistic perspective to anticipate issues.
Emit telemetry from application code that correlates the key points of the execution flow and gives an end-to-end view at different levels of granularity.	Prioritize actions based on the severity level, and understand the context given its verbosity. This information is crucial for troubleshooting purposes.
Own the responsibility of emitting and collecting data, even when data sinks are shared by multiple teams and managed by central teams.	By localizing monitoring data to the workload environment, the team can access logs and metrics to address workload concerns.
Collect **just enough data** and retain it for **just enough time**. Consider the cost tradeoffs associated with logging and storing data.	Intentional data collection helps you **optimize financial and operational costs** associated with collecting more data than you need. **Minimize the noise and avoid intensive computation** during analysis, and reduce the cost of storing data that you no longer need.

Approach	Benefits
Make a distinction between the different monitoring signals: profiles, logs, metrics, and traces. Use each signal for the right purpose. Prioritize the **use of metrics to trigger actions** that rely on numeric measurements. Use **profiles to get lower-level visibility**, such as memory allocation, into the system. Reserve the **use of logs and traces to provide context** for flows and dependencies.	By using the signals for the right purposes, you can prevent inefficient implementation of the monitoring system. For example, using logs for actions requires parsing. You might be able to achieve the same goals faster with metrics.
Aggregate and visualize data in dashboards to present monitoring data that's catered to audiences and keeps the business context in mind. Use **situational dashboards** for surfacing data to drive awareness among the stakeholders. **Use operational dashboards and workbooks** with drill-down capabilities for operator activities like incident response. Frequently refresh the dashboards and provide granular data.	With visualizations, you can analyze trends, track against business targets, and manage incidents. Dashboards that are tailored to the interest of the customer make interpretation relevant and accelerate time to detection and action.
Make alerts actionable by notifying the accountable roles with standardized descriptions and severity levels. Provide information that's collated from various sources and track deviations from business targets. **Trigger alerts only for incidents that require action**. **Strive for proactive** and thought-provoking alerts that initiate actions before a degraded state becomes a failure.	Alerts bring attention to significant events as defined by the organization. A good alert system identifies actions and severity and provides just enough data to drive clarity and purpose. **Operators can start on remediation without delay**.

Automate for efficiency

Expand table

 Replace repetitive manual tasks with software automation **that completes them quicker, with greater consistency and accuracy, and reduces risks.**

The workload might have workflows with processes that involve team members doing mundane, repetitive, and time-consuming tasks that don't actually need human intellect. Depending on the frequency, you might spend considerable time on these efforts, investing more time as the workload grows. Also, these processes are often error-prone due to human input.

Through automation, you save time, effort, and money, and you avoid mistakes.

Expand table

Approach	Benefits
Evaluate all workflows against criteria that's at the right level of complexity, effort, frequency, accuracy, timeliness, and lifespan. **Automate workflows** based on that evaluation and prioritize the workflows with the highest expected returns. **Remove redundant workflows** or add value to justify human effort.	You can reinvest team capacity in higher value work and increase productivity and consistency. Building an inventory of workflows ensures that you automate the right tasks. Removing redundant tasks reduces complexity and errors.
Be explicit about your decision when you evaluate whether to **build custom tooling or buy software**. Reserve building automation for highly specialized and high-value work.	By buying off-the-shelf software and taking advantage of the support contract, you save on maintenance costs. By building software, you have more control and can cater to use cases that are unique to your team and workload. However, there's a cost impact. Choice of tooling brings a level of standardization to your operations. With training, you can achieve a uniform level of readiness for adoption.
Design your workload components to **support automation capabilities**.	Avoid the situation where lack of automation in your system design promotes the anti-pattern of repetitive tasks, slows down growth, and starts accumulating technical debt.
Treat all **automation as a critical dependency** of your workload. Adapt to the workload's expected growth. Your automation tooling is an integral part of your workload, and it should adhere to the five Well-Architected Framework pillars.	Design your automation component to withstand risks, such as security threats. With applied best practices, you can avoid implementation sprawl. The workload will continue to operate with a high-level guarantee if this dependency is kept functional and safe.

Approach	Benefits
Automate at-scale by exploring options beyond your workload. **Favor a "design once, run everywhere"** model by providing templates and frameworks to onboard new projects and promote reuse of existing designs and implementations.	Employ tried and tested methods and reduce chances of failure.

Adopt safe deployment practices

Expand table

 **Achieve consistency in deployments by using guardrails that reduce the effect of errors and unexpected conditions.**

Build an automated and modular workload supply chain to ensure consistent, predictable, and repeatable deployments across all environments. Applying safe practices early ensures confidence in production and enables quick recovery if issues reach customers.

All changes, whether code, configuration, or artifacts, must be deployed with the same level of rigor. Testing, monitoring, and versioning are common practices to achieve consistency.

Expand table

Approach	Benefits
Use Infrastructure as Code (IaC) to define the desired state of all infrastructure. Use a modular and layered approach but avoid unnecessary abstractions. Align layers with lifecycle needs, keeping foundational layers stable.	IaC enables deployment automation and consistency and serves as self-documentation that can be used for tracing. IaC artifacts become part of your software development lifecycle, which enables testing and quality review processes. IaC also helps detect and mitigate configuration drift.
Prefer small, incremental updates that are deployed frequently.	Smaller updates simplify validation by reducing the number of concurrent errors. When multiple defective changes are released simultaneously, they can significantly increase the blast radius.
Deploy every code and infrastructure change by using **automated pipelines** across all environments.	Consistent deployment methods reduce errors and variance making deployments reliable and repeatable. The deployment

Approach	Benefits
	process documents itself, and each run creates a record of activities.
Test updates rigorously throughout the development lifecycle, in preproduction and production environments.	Early testing catches issues sooner, allows for iterative fixes and reduces problems by the time the update is ready for production. Having multiple pre-production environments enables various types of testing, boosting confidence in a successful production release.
Roll out new features using deployment patterns that allow for progressive exposure and gradual adoption by users. Test for backward and forward compatibility.	Controlled rollout of updates reduces the risk of widespread issues from defects. Gradually increasing exposure helps ensure compatibility and stability, building confidence in the release.
Be ready with compensating actions to **recover from faulty deployments or critical defects** in production. Use **automation backed by testing** to roll out fixes. For emergency updates, have an expedited process that's pre-approved by stakeholders.	Having a mitigation plan reduces the duration of potential impact. You can quickly deploy urgent fixes, like security patches, to get users a safe version faster.

Next steps

We recommend that you review the Operational Excellence checklist to explore other concepts.

Operational Excellence checklist

Design review checklist for Operational Excellence

Article • 03/07/2025

This checklist presents a set of recommendations to help you build a culture of operational excellence. Start with a DevOps approach to integrate specializations from multiple disciplines. This approach creates a rigorous design and development practice. This approach leads to repeatable, reliable, and safe deployments of infrastructure and code.

Prioritize human intervention in areas that benefit from it, and incorporate automation in other areas. Observability serves operational excellence by monitoring health events and also for validating the current workload design and implementation to inform future product development.

If you don't consider tradeoffs and recommendations for operational excellence, your workload might be at risk. Carefully consider the points covered in the following checklist to instill confidence in your design's success.

Checklist

Expand table

	Code	Recommendation
☐	OE:01	**Define your standard practices to develop and operate your workload.** Foster a blameless culture that emphasizes continuous learning and prioritizes continuous improvement and optimization.
☐	OE:02	**Formalize the way you run routine, as needed, and emergency operational tasks**. Increase consistency and predictability by adopting industry-proven practices and approaches.
☐	OE:03	**Formalize software ideation and planning processes.** Draw from established industry and organizational standards for team communication, requirements and design documentation, and software development processes.
☐	OE:04 OE:04 OE:04	**Enhance software development and quality assurance** by implementing industry-standard practices. Ensure clear role definitions and consistent processes by standardizing tools, source control, design patterns, documentation, and style guides.
☐	OE:05	**Use a standardized infrastructure as code (IaC) approach to prepare resources and configurations.** Use IaC to ensure consistent styles, modularization, and quality

	Code	Recommendation
		assurance. Prefer declarative over imperative approaches when practical.
☐	OE:06	**Build a workload supply chain that drives changes** through predictable, automated pipelines. Ensure these pipelines test and promote changes across all environments and quality gates. Incorporate comprehensive testing.
☐	OE:07 OE:07	**Design and implement a monitoring system** to capture and expose telemetry, metrics, and logs from your infrastructure and code. Use this data to validate design choices and guide future design and business decisions.
☐	OE:08	**Establish a robust emergency operations practice.** Create an incident response plan that clearly documents roles, responsibilities, and all emergency response processes and procedures. Capture learnings through postmortems and incident reports to continuously improve the plan and the workload.
☐	OE:09	**Automate tasks that are repetitive, procedural, and provide a clear return on investment.** Prefer off-the-shelf automation tools over custom solutions. Apply the Well-Architected Framework pillars to the design and implementation of all automation efforts.
☐	OE:10	**Design and implement automation upfront** for tasks like lifecycle management, bootstrapping, and governance. Avoid retrofitting automation later. Simplify your design by adopting platform-native automation functionality.
☐	OE:11	**Clearly define your workload's safe deployment practices.** Focus on small, incremental releases with quality gates. Use modern deployment patterns and progressive exposure to manage risk. Plan for both routine and emergency deployments.
☐	OE:12	**Implement a deployment failure mitigation strategy** to handle unexpected issues during rollout. Use approaches like rollback, feature disablement, or the native capabilities of your deployment pattern for quick recovery.

Next steps

We recommend that you review the Operational Excellence tradeoffs to explore other concepts.

Operational Excellence tradeoffs

Feedback

Was this page helpful? Yes No

Operational Excellence tradeoffs

Article • 10/10/2024

Operational Excellence provides workload quality through the implementation of clear team standards, understood responsibility and accountability, attention to customer outcomes, and team cohesion. The implementation of these goals is rooted in DevOps, which recommends minimizing process variance, reducing human error, and ultimately increasing the return of value for the workload. That value isn't just measured against the functional requirements served by the components of the workload. It's also measured by the value that the team delivers in striving for improvement.

During the design phase of a workload and over its lifecycle, as continuous improvement steps are taken, it's important to consider how decisions based on the Operational Excellence design principles and the recommendations in the Design review checklist for Operational Excellence might influence the goals and optimizations of other pillars. Certain decisions might benefit some pillars but constitute tradeoffs for others. This article describes example tradeoffs that a workload team might encounter when designing workload architecture and operations.

Operational Excellence tradeoffs with Reliability

>  **Tradeoff: Increased complexity.** Reliability prioritizes simplicity, because simple design minimizes misconfiguration and reduces unexpected interactions.

- Safe deployment strategies often require some amount of forward and backward compatibility between application logic and data in the workload. This added complexity increases the testing burden and can lead to complexities or integrity issues with the workload's data.
- Highly layered, modularized, or parameterized infrastructure as code can increase the chance of accidental misconfiguration because of the complexity of the interaction between the code components.
- Cloud design patterns that benefit operations sometimes necessitate the introduction of additional components, for example, the use of an external configuration store or the coordination of sidecar deployments in a containerized application platform. The additional components and added layers of indirection increase the points of interaction in the system, increasing the surface area for malfunction or misconfiguration.

- Workload components that are designed to independently evolve to support agile development and hosting introduce dependencies on service discovery as a layer of indirection. Service discovery might lack responsiveness to change, and malfunction can be hard to diagnose.

>  **Tradeoff: Increased potentially destabilizing activities.** The Reliability pillar encourages the avoidance of activities or design choices that can destabilize a system and lead to disruptions, outages, or malfunctions.

- Deploying small, incremental changes is a technique for mitigating risk, but those small changes are also expected to be delivered to production more frequently. Deployments can destabilize a system, so as the rate of deployment increases, so does this risk.
- A culture that measures itself with velocity metrics like deployments per week and uses automation that can facilitate introducing changes at a faster pace is also likely to perform more deployments in a shorter period.
- Increasing density to simplify operations by reducing the number of control and observability surfaces can also lead to an increased availability risk because malfunction or misconfiguration increases the impact radius of a destabilizing event.

Operational Excellence tradeoffs with Security

>  **Tradeoff: Increased surface area.** The Security pillar recommends a reduced workload surface area in terms of components and exposure to operations. This reduction minimizes attack vectors and produces a smaller scope for security control and testing.

- Components that surround the workload and support its operations, like automation or a custom control plane, must also be in scope for regular security hardening and testing.
- Routine, ad hoc, and emergency operations increase the points of contact with the workload. A zero trust approach requires that these processes are considered attack vectors and must be included in the security controls and validation for the workload.

- The observability platform of the system collects logs and metrics about the workload, which can be a valuable source of information disclosure. Therefore, the workload's security needs to extend to protect data sinks from internal and external threats.
- Build agents, externalized configuration and feature toggle stores, and side-by-side deployment approaches all increase the application surface area that requires security.
- A higher deployment frequency caused by small, incremental changes or by "get current, stay current" efforts results in more security testing in the software development lifecycle.

>  **Tradeoff: Increased desire for transparency.** A secure workload is based on designs that protect the confidentiality of data that flows through the components of the system.

Observability platforms ingest data of all types to gain insights into a workload's health and behavior. As teams try to attain higher fidelity in observability data, there's an increased risk that data classification controls, like data masking, of the source systems don't extend to the logs and log sinks of the observability platform.

>  **Tradeoff: Reduced segmentation.** A key security approach for isolating access and function is to design a strong segmentation strategy. This design is implemented through resource isolation and identity controls.

- Co-locating disparate application components in shared compute, network, and data resources to make management easier reverses segmentation or makes role-based segmentation harder to achieve. Co-located components might also need to share a workload identity, which can lead to over-assignment of permissions or a lack of traceability.
- Collecting all logs from across the system in a unified log sink can make querying and building alerts easier. However, doing so can also make it harder or impossible to provide row-based security in order to treat sensitive data with the required audit controls.
- Simplifying the management of attribute-based or role-based security by reducing the granularity of roles and their assignments can lead to inappropriately broad permissions.

Operational Excellence tradeoffs with Cost Optimization

The Operational Excellence pillar never recommends activities that reduce productivity or jeopardize a workload's return on investment. Recommendations that seem to shift focus from delivery activities take into account long-term best interests for the workload and team. If your workload is nearing its sunset date, it probably doesn't make sense to invest highly in recommendations that trigger these tradeoffs.

>  **Tradeoff: Increased resource spending.** A major cost driver for a workload is the cost of its resources. Deploying fewer resources, right-sizing resources, and reducing consumption generally helps keep costs low.

- Implementing safe deployment practices, even if the changes are relatively small, can lead to an increase in the number of resources that are concurrently deployed. These patterns require the deployment of multiple concurrent instances of the application or infrastructure component so that traffic can be shifted in a controlled way. This increase is more pronounced in a workload that uses an immutable infrastructure approach.

- The team might need to introduce additional workload components in order to implement operationally aligned cloud design patterns or workload automation. For example, to support deployment agility, they might add a gateway routing component. To support better configuration management, they might add an external configuration store. To support tenant lifecycle events, they might build a control plane. These resources also influence the costs of preproduction environments.

- Increasing the number of preproduction environments to improve the development and testing experience through isolation also increases the number of resources. These resources, which aren't used to deliver supply against production demand, increase the cost of the solution.

- Increasing the parity of preproduction environments with the production environment, in terms of resource count, SKUs, and data volumes, improves the quality assurance process. The cost increases as parity increases.

- Although telemetry data isn't directly a resource, to enable the effectiveness of observability platforms, this data needs to be persisted. Most operational data stores have pricing that's based on a combination of ingestion rates and volume. Generally, as the amount of low-latency, high-diversity telemetry increases, costs

also increase. For multi-region deployments, these operational data sinks are expected to be deployed per region, so any per-resource costs become a factor.

>  **Tradeoff: Decreased focus on delivery activities.** Workload team members deliver increased workload value by efficiently performing tasks that are aligned to their capabilities.

- Workload teams that spend time creating and refining a healthy and responsible support structure and incident response are providing a valuable service to the workload's users. As the support effort increases (for example, formal on-call rotations), usually because of a change in business criticality, the costs of these activities increase. This cost increase can be the result of an increase in staff or can be incurred indirectly in the form of attention that's shifted from delivery activities to supporting functions.

- Training is a critical part of a workload team's personal continuous improvement process. This training can be formal or self-directed during personal enrichment time. As the amount of training time increases, the amount of time available for direct development of the workload decreases. Investment in training is diminished when the training isn't role-based or specifically relevant to the workload or its future.

- Standardized routine operational tasks for protecting the reliability, security, and performance efficiency of a workload take time to define, refine, and perform. This time isn't directly spent on delivery. Some examples of these tasks are comprehensive change impact analysis, change control processes, thorough testing, and increased patch management. As the frequency, comprehensiveness, or operational burden of these tasks increases, the time invested also increases.

>  **Tradeoff: Increased tooling demands and diversity.** The Cost Optimization pillar recommends the reduction of tooling sprawl, consolidation of vendors, and a right-sized approach to all tooling purchases.

A workload team purchases tools and hardware to support activities that are performed during the entire software development lifecycle (SDLC), including planning and design, development and testing, and monitoring. The marketplace for tooling in this space is growing. Tools are offered at various price points that usually correspond to the tools' features and capabilities. Except for free offerings, these tools incur initial licensing costs, which might be per-user, per-device, or site-wide. They often also require ongoing maintenance contracts. New vendor relationships might need to be established. Here

are some examples of expected tooling or hardware spending that's associated with the principles of operational excellence:

- Requirements and backlog management
- Architecture design tools
- UI/UX design tools
- Code and asset hosting
- Code and low-code development environments
- Automation tools
- Development and quality assurance workstations
- Development and deployment pipelines
- Test execution and tracking
- Observability tools

Operational Excellence tradeoffs with Performance Efficiency

>  **Tradeoff: Increased resource utilization.** The Performance Efficiency pillar recommends the allocation of as much of the available compute and network as possible to the requirements of the workload.

- A workload's observability framework requires that the components in the architecture allocate time and resources to create, collect, and stream logs and metrics. These data points help ensure that effective alerting and monitoring is possible for reliability, security, and performance. As the level of instrumentation increases, the strain on system resources might also increase.
- Some deployment models, like blue/green deployment, which a workload might use for safe deployment, might introduce side-by-side deployments on the production application platform. These deployments require preemptive scaling to provide enough supply to meet future demand, or leave a mostly dormant deployment in place for a period of time to support rollback.

>  **Tradeoff: Increased latency.** To create performant workloads, teams look for ways to reduce the time and resources that workloads consume to perform their tasks.

- Many deployment models require the use of gateway routing access patterns, which can introduce latency. This latency draws against the performance target

budget for the related flows.

- Some cloud design patterns that support "independent change over time" approaches to support the ideals of incremental improvement can introduce latency due to the traversal of additional components. This latency can be introduced by gateways, messaging brokers, or anti-corruption layers.

Related links

Explore the tradeoffs for the other pillars:

- Reliability tradeoffs
- Security tradeoffs
- Cost Optimization tradeoffs
- Performance Efficiency tradeoffs

Feedback

Was this page helpful? Yes No

Cloud design patterns that support operational excellence

Article • 10/10/2024

When you design workload architectures, you should use industry patterns that address common challenges. Patterns can help you make intentional tradeoffs within workloads and optimize for your desired outcome. They can also help mitigate risks that originate from specific problems, which can affect reliability, security, performance, and cost. Because operations cut across all those areas, risks will eventually affect workload operations. These patterns are backed by real-world experience, are designed for cloud scale and operating models, and are inherently vendor agnostic. Using well-known patterns as a way to standardize your workload design is itself a component of operational excellence.

Many design patterns directly support one or more architecture pillars. Design patterns that support the Operational Excellence pillar use topologies that provide a solid foundation for safe deployment practices and facilitate architecture evolution over time, migration scenarios, and observability.

Design patterns for operational excellence

The following table summarizes cloud design patterns that support the goals of operational excellence.

Expand table

Pattern	Summary
Anti-Corruption Layer	Protects new system components from the behavior or implementation choices of legacy systems by adding a mediator layer to proxy interactions between legacy and new components. This pattern helps ensure that new component design remains uninfluenced by legacy implementations that might have different data models or business rules when you integrate with these legacy systems. The pattern is especially useful in gradual system migrations. It reduces technical debt in new components while still supporting existing components.
Choreography	Coordinates the behavior of autonomous distributed components in a workload by using decentralized, event-driven communication. This pattern can be useful when you expect to update or replace services frequently during the lifecycle of a workload. Because the distributed

Pattern	Summary
	components are autonomous, you can modify the workload with less overall change to the system.
Compute Resource Consolidation	Optimizes and consolidates compute resources by increasing density. This pattern combines multiple applications or components of a workload on a shared infrastructure. The consolidation leads to a more homogenous compute platform, which can simplify management and observability, reduce disparate approaches to operational tasks, and reduce the amount of tooling that's required.
Deployment Stamps	Provides an approach for releasing a specific version of an application and its infrastructure as a controlled unit of deployment, based on the assumption that the same or different versions will be deployed concurrently. This pattern aligns with immutable infrastructure goals, supports advanced deployment models, and can facilitate safe deployment practices.
External Configuration Store	Extracts configuration to a service that's externalized to the application to support dynamic updates to configuration values without requiring code changes or application redeployment. This separation of application configuration from application code supports environment-specific configuration and applies versioning to configuration values. External configuration stores are also a common place to manage feature flags to enable safe deployment practices.
Gateway Aggregation	Simplifies client interactions with your workload by aggregating calls to multiple backend services in a single request. This topology enables backend logic to evolve independently from clients, allowing you to change the chained service implementations, or even data sources, without needing to change client touchpoints.
Gateway Offloading	Offloads request processing to a gateway device before and after forwarding the request to a backend node. Adding an offloading gateway to the request process enables you to manage the configuration and upkeep of the offloaded functionality from single point instead of managing it from multiple nodes.
Gateway Routing	Routes incoming network requests to various backend systems based on request intents, business logic, and backend availability. Gateway routing enables you to decouple requests from backends, which in turn enables your backends to support advanced deployment models, platform transitions, and a single point of management for domain name resolution and encryption in transit.
Health Endpoint Monitoring	Provides a way to monitor the health or status of a system by exposing an endpoint that's specifically designed for that purpose. Standardizing which health endpoints to expose, and the level of analysis in the results, across your workload can help you triage issues.

Pattern	Summary
Messaging Bridge	Provides an intermediary to enable communication between messaging systems that are otherwise incompatible because of protocol or format. This decoupling provides flexibility when you transition messaging and eventing technology within your workload or when you have heterogeneous requirements from external dependencies.
Publisher/Subscriber	Decouples components of an architecture by replacing direct client-to-service or client-to-services communication with communication via an intermediate message broker or event bus. This layer of indirection can enable you to safely change the implementation on either the publisher or subscriber side without needing to coordinate changes to both components.
Quarantine	Ensures external assets meet a team-agreed quality level before being authorized to consume them in the workload. Automation and consistency on these checks are a part of the workload's software development lifecycle and safe deployment practices (SDP).
Sidecar	Extends the functionality of an application by encapsulating nonprimary or cross-cutting tasks in a companion process that exists alongside the main application. This pattern provides an approach to implementing flexibility in tool integration that might enhance the application's observability without requiring the application to take direct implementation dependencies. It enables the sidecar functionality to evolve independently and be maintained independently of the application's lifecycle.
Strangler Fig	Provides an approach for systematically replacing the components of a running system with new components, often during a migration or modernization of the system. This pattern provides a continuous improvement approach, in which incremental replacement with small changes over time is preferred rather than large systemic changes that are riskier to implement.

Next steps

Review the cloud design patterns that support the other Azure Well-Architected Framework pillars:

- Cloud design patterns that support reliability
- Cloud design patterns that support security
- Cloud design patterns that support cost optimization
- Cloud design patterns that support performance efficiency

Feedback

Was this page helpful? Yes No

Recommendations for fostering DevOps culture

Article • 04/30/2024

Applies to this Azure Well-Architected Framework Operational Excellence checklist recommendation:

Expand table

OE:01	**Determine workload team members' specializations, and integrate them into a robust set of practices to design, develop, deploy, and operate your workload to specification. Team members must have clarity in decision-making and responsibilities, value continuous improvement and optimization, and adopt a blameless culture that incorporates continuous learning.**

This guide describes the recommendations for implementing DevOps principles and practices in your workload. Fostering DevOps culture can help build a foundation of shared ownership, mutual respect, and appreciation of high quality work in your workload team. Devops culture provides a template for high-performing teams to thrive in the system that they're in.

Key design strategies

A workload that operates according to the Well-Architected Framework recommended practices starts with the adoption of the DevOps culture of cohesiveness, responsibility, continuous learning, and improvement. Team members bring their own expertise and might focus on specific areas of workload operation. However, your team as a whole should be able to independently manage day-to-day, as-needed, and emergency tasks, with support from outside teams when necessary. Your team must work within the overall organizational requirements and collaborate with other teams by using a mindset that values shared knowledge.

The following recommendations can help you adopt and implement DevOps practices in your team to optimize the operation of your workload and add value to your organization.

Foster mutual respect

A team should operate by using a code of ethics based on mutual respect. Everyone on the team has expertise that brings value to the team. Recognizing individual ability as

a core tenet of the team culture allows conversation to start from a safe place. Individuals should feel that they can offer honest opinions about workload operations and be treated respectfully.

Mutual respect fosters a blameless culture. When issues occur, the workload team should take collaborative ownership and find ways to improve instead of assigning blame and affecting the team's cohesiveness.

Team roles and responsibilities

Teams take ownership and responsibility for the workload when they value their work. The workload team ultimately has end-to-end responsibility for the operation of their workload. Although there might be outside services required for certain aspects of the workload operation, your team is responsible for collaborating with other teams and ensuring that all functions are successfully completed. Regardless of how involved they are in supporting services, workload team members must consider every function that supports the workload as their responsibility. This mindset helps reinforce a common sense of ownership.

Clearly define team roles and decision-making responsibilities. Team decision-making should be as democratic as possible, but structured so that decisions are made efficiently. When there are differing opinions about a situation, someone must be responsible for making the final decision based on the evidence that's presented. Team decisions can affect the entire workload, so it's important that individuals feel heard and valued throughout the decision-making process even if they don't agree with the final decision.

Continuous learning and improvement

Use enablement teams to the workload teams' advantage. Some organizations have enablement teams, such as platform teams, architecture review boards, or cloud centers of excellence. These teams provide standards that all workload teams must follow to ensure that there's consistency in design and process. Empower your workload team to have open lines of communication with enablement teams and to work collaboratively to improve processes and share knowledge. Support a mindset of continuous learning and improvement in your team via open communication.

Learn from each other to develop a cross-functional team. Establish a team structure in which everyone is a specialist in their function and a generalist in all other functions so that team members can support each other when needed. Cross-functionality helps

team members to develop appreciation for each other's expertise and can help them understand the complexity of the entire workload.

Commitment to optimization

Understand business, regulatory and other requirements and integrate them into your practices. Workload teams don't operate in a vacuum. Your team is subject to requirements enforced by the business, industry, and geographic regions you operate in. Ensure that your workload team members understand the requirements that they must follow and the consequences of a failure to meet those requirements.

Proactively adapt your practices to ensure that you're compliant with requirements by integrating testing mechanisms that specifically target required functions. Your organization might impose some degree of governance over your workload. Use the requirements your business standardizes as guardrails to ensure that you're operating appropriately.

Regularly review your standard operating procedures with the team to foster discussions about areas of improvement. Avoid complacency and encourage innovative thinking by fostering a philosophy that all standard operating procedures should be continuously reviewed and improved throughout the workload lifecycle. Team members should feel empowered to offer opinions on improvements at any time. However, ensure that you dedicate time to review procedures together so that everyone has space to think about areas for improvement and conduct focused discussions about their ideas.

Embrace safe experimentation. Give team members access to sandbox environments and ensure that time is built into sprints to allow for experimentation. Document standards that define how new functionality is integrated into the workload when a team member discovers a function or component that would offer tangible benefits. Be careful to ensure that new functionality is aligned with your safe deployment practices.

Considerations

Strictly defined roles and responsibilities could result in a level of discomfort for some team members when they're performing functions outside their responsibility. Conduct open and honest discussions with the team about team structure, and be open to making adjustments when needed.

Azure facilitation

Microsoft publishes extensive documentation about DevOps culture in a dedicated DevOps resource center.

Related links

- DevOps resource center
- Safe deployment practices

Operational Excellence checklist

Refer to the complete set of recommendations.

Operational Excellence checklist

Feedback

Was this page helpful? Yes No

Recommendations for formalizing routine and nonroutine tasks

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Operational Excellence checklist recommendation:

OE:02	**Formalize the way you run routine, as needed, and emergency operational tasks by using documentation, checklists, or automation. Strive for consistency and predictability for team processes and deliverables by adopting industry-leading practices and approaches, such as a shift left approach.**

This guide describes the recommendations for formalizing routine and nonroutine tasks. Efficient and successful workload teams have consistent and predictable workload management practices. Optimize efficiency and consistency by adopting industry-proven tools and practices and by automating processes when it's practical. By being consistent about how routine, improvised, and emergency tasks are handled, you minimize the risk of being unprepared when issues arise. By taking a continuous improvement approach to workload management, you increase your team's consistency and efficiency throughout the workload's lifecycle.

Key design strategies

Consistent processes make your work quality predictable, and predictable work quality makes workload support smoother. To achieve consistency in your processes, you need to be intentional and explicit about how you run processes in standard patterns. Use strategies like automation and shift-left approaches to minimize potential areas of unpredictability.

Process standardization takes many shapes. Describing every way that you might approach standardization is out of scope for this guide, but some general recommendations include:

- The processes that you standardize should cover all aspects of workload management: reliability, security, cost optimization, performance, and operational processes. The workload team should have as much ownership of the processes as they need to maintain and continuously improve the workload under the overall governance of the organization.

- The documentation that you produce captures your standard operating procedures and dictates how things are done, but the documentation is never final. Procedures should evolve as your workload and your team evolve. Regularly review and challenge standards to ensure that they're the right standards for your team right now. Your documentation should be templatized and version controlled to ensure consistency in the documentation formats and that there's a record of reviews and updates. Versioning also helps to reinforce requirements that the team conducts reviews of the documentation at regularly scheduled intervals.

- Break down routine, improvised, and emergency tasks into checklist items that are easy to understand. An example of a routine task is the process of applying an update to an open-source dependency. The workload might need to use an open-source library like an SDK to use a third-party message service. This SDK should be updated regularly for security patches, bug fixes, and functionality improvements.

  When you determine that you need an update, the workload team might have a checklist that includes items like testing the update in lower environments, creating a change management request to deploy the update in production, and updating documentation, such as a wiki or knowledge base, to ensure that they reflect the correct version. Focus each checklist item on a discrete task that's clearly defined.

- Improvised and emergency tasks are scenario-specific, but operators should still clearly understand their roles and responsibilities. They need to know how to interact with the workload team and with other teams in the organization to efficiently work through those types of tasks.

  For example, an improvised task might deploy a new type of resource, such as a machine learning service, that has been approved for enhancing the workload functionality. There might not be a fully realized checklist for the deployment and testing of this type of resource. However, there should be general checklists for adding new resources to your infrastructure as code templates and standards that cover infrastructure testing for performance, security, and reliability at each stage of your promotion chain.

  Likewise, your emergency response plan should clearly define roles and responsibilities and general processes and procedures. You must adhere to this plan in emergency situations to ensure that you handle them efficiently.

  Improvised and emergency operations are also good opportunities to learn how to improve your standard operating procedures. Ask the workload team to reflect on ways that the operations could have gone smoother, and determine whether an update to existing processes could be beneficial going forward.

- Adopt industry-proven practices to minimize the time your team spends on inventing processes and standards. Following Agile practices with Scrum, organizing your work through Kanban boards, and adopting a shift-left ethos are all examples of practices that have been developed over many years and have been proven effective for organizations of any size. Many mature organizations use version-controlled standard operating procedures, wikis, new employee manuals, and operations manuals to enforce consistency.

  Rely on your team's experience to decide which practices fit your workload lifecycle management. Learn from other teams about the standards that they've successfully implemented to understand how particular practices fit into your organizational structure.

  In this context, a shift-left ethos means that workload teams should be empowered to look for measures that can improve the security, reliability, and cost efficiency of the workload. Then, they add those improvement measures to their own backlog, rather than shifting the responsibility to outside teams.

  For example, an exploratory test might uncover an area for improvement in security that might not have been discovered until it was exposed by security scans, which happen monthly or even less frequently. Encourage the workload team to take ownership of the workload in all aspects of its lifecycle and to contribute to its continuous improvement proactively, rather than relying on other teams.

- Incorporate organizational requirements and cross-cutting functions into your standard operating procedures. Your organization might have standards for some processes that you should adopt. However, you might also be empowered to develop your own standards for other processes, so look for ways to incorporate required standards into your processes. The processes that you own are likely to intersect with other teams' processes, so strive to align standards to a practicable extent.

  Document where the workload team's and other team's processes diverge to ensure that the workload team can work better with other teams when there are intersection points. Central security teams might use different tools and procedures than the workload team, and the teams can collaborate more easily if they're aware of those differences.

- Incorporate compliance requirements into your standard operating procedures. Depending on your industry and the regions in which you operate, there might be strict requirements about how to perform and document tasks. Make sure that you

understand and incorporate those requirements while you build your standards. Regularly train the workload team on those requirements.

- Use automation to help achieve consistency. Automate tasks that are repetitive and prone to human error to relieve management burden from your team. Look for opportunities to automate processes, like generating ITSM tickets, for example. For more information, see the Recommendations for implementing automation.

- Be intentional about your approach to open-source adoption. Standardize the rules about when the use of open-source tools is allowed, and ensure that you align with organizational and compliance requirements. You might want to create standards about workload team members' contributions to open-source projects and decide whether to open your in-house code to other development teams in the organization.

Azure facilitation

Although there are no Azure products that directly facilitate the formalization of processes and procedures, Microsoft publishes lots of guidance on this topic. Use this guidance to understand industry-proven and recommended practices and to consider how to apply them to your workload.

The Well-Architected Framework also provides detailed guidance on the processes and procedures that should be codified to ensure that your workload and workload team run according to industry standards.

Organizational alignment

Cloud Adoption Framework provides guidance for central teams on many areas of standardizing processes and procedures across an organization.

For more information, see the Azure landing zone design areas and DevOps considerations.

Tradeoffs

Codifying standard operating procedures can carry the risk of leading to stagnation or complacency. Standards should be followed, but they shouldn't be rigid or static. Strive to find a balance between strict adherence and allowance for innovation so that your processes can evolve safely over time.

Related links

- Recommendations for implementing automation
- Recommendations for enabling automation in your workload
- Recommendations for formalizing software development management practices

Operational Excellence checklist

Refer to the complete set of recommendations.

Operational Excellence checklist

Recommendations for formalizing software development and management

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Operational Excellence checklist recommendation:

OE:03	**Formalize software ideation and planning processes. Draw from established industry and organizational standards. Use a common, prioritized backlog and sufficiently detailed specifications. Based on outcomes, drive continuous improvements in your planning process.**

This guide describes the recommendations for managing software development practices in accordance with established standards. Your team's ability to produce high quality software relies on a structured, collaborative approach to development planning. Product owners and managers must be able to clearly understand and articulate to stakeholders the work that developers are doing at any time in a development cycle. Conversely, developers must understand the goals of the development cycle via well-written features, user stories, and acceptance criteria. Established standards define how development practices should be performed and allow the workload team to collaborate effectively, reducing the risk of confusion on goals and expectations.

Key design strategies

Formalize your software development practices to help ensure that product owners, project managers, and developers understand the goals of each sprint and deliver consistent quality to stakeholders. To review guidance on development practices, see the continuous integration guide.

Standards for development planning

- **Collaboration**: The process of defining proposed changes to the workload should be a collaborative effort. Most changes to the workload will impact multiple functions and/or components, so involving as many workload team members as possible will help ensure that important considerations are not missed and that everyone is aware of the impact on their particular domain. Collaboration also helps clearly define the scope of a change and how to divide the necessary tasks

needed to accomplish the change into well-defined work items, as a larger group with expertise across domains will be able to provide experience-backed estimates for the required effort.

- **Tools**: Use established, industry-proven tools and processes, like Agile, Scrum, and Kanban boards. Developing your own tools and processes is a significant undertaking, taking time and development cycles that could otherwise be spent on your workload. Most experienced DevOps engineers and product owners are familiar with these types of tools and processes, so the learning curve in adopting them should be minimal. Likewise, the onboarding process for new hires will also benefit from using standard tools and processes as they are likely to have a degree of exposure to the same tools and processes already.

- **Deployment**: Plan to use frequent small, iterative deployments instead of large infrequent deployments. Using this approach will help keep user stories and work items manageable from a project management standpoint and reduce the risk of large-scale issues when deployments fail.

- **Terms**: Standardize your definition of *finished* development cycles to help ensure that supporting functions, including testing, documentation, and accessibility features, are successfully completed.

- **Communication**: Define the standard protocols for product owners and project managers to promote upcoming releases internally and externally. For example, you might establish a standard for communications to external parties about upcoming releases. The standard might dictate that communication should be sent two weeks before the release and a reminder should be sent 24 hours before the release.

- **User stories**: Standardize a template for user stories. Ensure that each user story is a discrete unit of work, written from the perspective of the end user. Well-written user stories should have the following characteristics:

    - Each user story should be wholly independent from each other. Keeping user stories independent of each other avoids any confusion with overlapping work and helps the team understand whether work on a given user story relies on the work on any others, which helps with scheduling and prioritization.

    - Each user story is negotiable. The end user and workload team members' perspectives are both essential to capture realistic user stories that can be accomplished over a short amount of time.

    - User stories are valuable to the end user. When you write user stories from the perspective of the end user, you capture the changes that they are interested in seeing and that will add value to their experience. Keeping this focus as the user story is broken down into work items helps ensure that each deployment provides an improved experience.

    - The effort required for a user story is estimable with a high degree of confidence. Without being able to have a close approximation of the hours required for a given user story, planning will be difficult and the potential for missing deadlines increases, potentially causing cascading effects on other planned work.

    - Well-written user stories are small, so that they can be completed within a few weeks. Smaller scoped stories help keep them estimable and manageable and help keep work items accomplishable.

    - User stories should be testable. Without being able to test that a feature has been delivered, the end user can't have confidence that the goal has been accomplished. Even if a test hasn't been written already for a given user story, there should be a clear understanding of how a test can be developed to prove the delivery of the feature.

- **Acceptance criteria**: Standardize a template for acceptance criteria. Ensure that acceptance criteria relates specifically to the user story and can be unambiguously proven using one or more acceptance tests.

- **Tracing**: Ensure that the development process is traceable. You should clearly trace the state of your production workload and the associated code back to quality assurance testing, acceptance criteria, user stories, and features. Detailed tracing might also be a regulatory requirement in some cases, like healthcare, for example.

- **Review**: Regularly perform internal audits of your development practices via development cycle retrospectives and postmortems. Process reflection should be blameless and should focus on learning that can be applied as improvements. Ensure that the team reflects on how effective the user story and tasks were in defining the necessary tasks and on the accuracy of time estimates.

- **Reports**: Standardize reports for stakeholders that provide useful metrics focusing on change. Focusing on change allows you to track product acceleration and deceleration. Helpful metrics can include changes in:

    - The monthly growth rate of adoption.

    - Performance.

- Training time.
- The frequency of incidents.

  Reporting shouldn't be used as a tool to evaluate the work of individuals, so avoid metrics like story points or lines of code for each engineer.

Azure facilitation

Azure Boards is a web-based service that enables teams to plan, track, and discuss work across the entire development process. It's well suited for Agile-based development practices.

GitHub Projects is a customizable project management tool that can organize projects and integrates by using issues and pull requests in GitHub.

Tradeoffs

Agile methodology can become too strict if it's overly prescriptive. Strive for a balance between well-defined standards and innovation.

Related links

- Best practices for Agile project management
- Azure Boards

Community links

- GitHub Projects

Operational Excellence checklist

Refer to the complete set of recommendations.

Operational Excellence checklist

Recommendations for standardizing tools and processes

06/10/2025

Applies to this Azure Well-Architected Framework Operational Excellence checklist recommendation:

Expand table

OE:04	**Optimize software development and quality assurance processes by following industry-proven practices for development and testing. For unambiguous role designation, standardize practices across components such as tooling, source control, application design patterns, documentation, and style guides.**

This guide describes the recommendations for defining standards for software development tools and processes. Defining consistent practices leads to an efficient workload team and high-quality work. High-performing teams use industry-proven tools and processes to minimize wasted effort and potential code errors.

Key design strategies

The first step of optimizing development practices is standardizing tools and processes. When possible, use industry-proven solutions rather than developing in-house solutions. To further optimize your practices, adopt low-code and no-code tools. These tools enable you to focus efforts on your application and help you save time. For all tools and processes that you standardize, implement training so your teams understand and use them efficiently. To define standards that help optimize your development practices, consider the following recommendations.

Use well-known and mature off-the-shelf tools

Use well-known and mature off-the-shelf tools and standardize their use. Highly effective engineering teams adopt the best-in-class tools. This approach minimizes the need to develop solutions for planning, development, testing, collaboration, and continuous integration and continuous delivery (CI/CD). Many enterprises give developers a choice between a few tools, but all options are standard tools for the organization and are validated internally. Most importantly, choose tools that meet the requirements for your workload. Off-the-shelf tools should provide the following functions:

- Work planning and backlog management

- Version control and repositories
- CI/CD pipelines
- Testing, such as integration, smoke, synthetic user, simulation, chaos, and other quality tests
- Code development

In some cases, one tool or a suite of tools might provide several functions. Ensure that you understand the capabilities of your tools and their limitations so they meet your requirements across functions.

Determine if you should invest in expensive tools or premium versions of tools. Consider the time and effort of developing your own solutions compared to features that the premium tools provide. Consider one-time costs versus recurring costs. In most cases, off-the-shelf tools provide higher value to your team.

Use low-code, no-code, and AI tools when practical. Low-code and no-code tools save experienced developers time by allowing them to easily plug in functionality rather than performing the entire code development process. These tools also allow workload team members that might not be trained developers to contribute to the operation of the workload. AI tools can help with code development, reviews, and optimization.

Standardize your branching strategy

Choose a trunk-based model when possible. Trunk-based branching keeps the workload development team in sync and encourages continuous delivery. Define branch policies to protect important branches, like the main branch. For more information, see Adopt a Git branching strategy and Branch policies and settings.

Evaluate metrics to quantify development effectiveness

Software development and quality assurance teams can improve only if they can quantify their effectiveness. To quantify effectiveness, they must identify the metrics that measure developer velocity and define KPIs. Examples of these metrics include:

- *Deployment frequency*: The number of deployments that each developer deploys each day.
- *Lead time*: The time that it takes for a task or user story to go from the backlog to a production deployment.
- *Mean time to resolution*: The average time that's spent fixing bugs or defects in code.

- *Change failure rate*: The percentage of changes that result in a failure.

To help stakeholders and the workload team easily track velocity, visualize KPIs by using dashboards or other reporting tools.

Standardize how your workload team writes, reviews, and documents code

Standardize how your workload team writes, reviews, and documents code by using a style guide. A standard style makes collaboration easy and helps with on-boarding new developers. To work effectively, new developers need to know how the workload team operates. A style guide with clearly defined standards can ease their training process. In the style guide, define standards for development languages, libraries, frameworks, and other conventions.

When it's practical, use tooling to enforce code-formatting standards. For example, Visual Studio offers several tools that scan code for style, quality, maintainability, design, and other issues. For infrastructure as code (IaC), you can use Checkov or Terrascan for Terraform.

To ensure consistency and avoid potential confusion, the style guide should include standard naming conventions for artifacts, environments, branches, builds, and runs.

You should also set guidelines and standards for the allowable degree of variance in your environments. If there are new languages, frameworks, or other technologies that workload team members want to add to the standard list, implement a process for using those tools in a sandbox or lower environment. Test their viability, and replace existing technologies when appropriate.

Use architecture decision records (ADRs) to keep a historical record of your workload team's design decisions. ADRs help your teams maintain a fresh understanding of the workload. They also help new team members learn about the design decisions that are made during the workload's lifecycle. Ensure that ADRs are version controlled.

In your ADR, include:

- Specific tools and technologies, for example using SQL or NoSQL, that your team chooses.
- The reasons for your team's decisions.
- Other options that were considered, which helps contextualize the final decision.
- Functional and nonfunctional requirements that are factored into decisions.
- The context of the decision-making process, like the problem that was addressed.

Implement standards for addressing technical debt

Adopt a mindset that technical debt is intentional and necessary for your workload team's deliverables. This mindset motivates your team to consider and address technical debt regularly to avoid accumulation. Address technical debt as a regularly recurring task in the backlog.

For example, suppose your team standardized on a library. Over time, you need to switch to a different library for new functionality in the workload. That transition might result in technical debt. Frequently, transitions like this can leave the workload team supporting two technologies because they can't fully transition smoothly. The workload team must prioritize completing the transition because when the workload achieves the new functionality, stakeholders are satisfied and are less likely to consider the technical debt.

Standardize how you apply versioning to your artifacts

Standardize how you apply versioning to your artifacts and how versioning is exposed internally and externally. For example, client-facing systems should expose their running version in the user interface. This technique is helpful when the workload team troubleshoots issues because the customer can easily communicate which version they use. REST interfaces can expose versions for certain components or databases. You might use a specific table in the metadata for a schema to expose the schema version.

Use industry-proven application design patterns to ensure that your application is reliable, performant, and secure. Use these patterns to save time and effort compared to developing your own solutions for your application. Choose the patterns that benefit your workload. Regularly review design patterns to ensure that you use the right patterns as your workload evolves.

Implement a shift-left approach to testing

Implement a shift-left approach to testing by performing unit testing early and often throughout the development process. Frequent testing in each development environment helps developers gain confidence in their applications. To help create your testing strategy with a shift-left approach, consider the following principles:

- *Write tests at the lowest level possible*. Favor tests with the fewest external dependencies, and run tests as part of the build.
- *Write tests once, and run tests everywhere, including production*. Write tests that you can run in every development environment without accounting for factors that are specific to one environment, like encrypted secrets or configurations.

- *Design your workload for testing*. When you develop your application, make testability a requirement.
- *Treat test code as application code*. Apply the same quality and development standards to application code and test code. Store test code alongside application code. Develop and maintain test code with application code. To ensure the quality of tests, discard tests that aren't reliable.
- *Consider test ownership, which is based on workload ownership*. Your workload team owns their testing and shouldn't rely on other teams to test their code.
- *Automate tests as much as possible*. Automated code relieves the burden on your workload team and enforces consistent quality.

Implement a variety of testing types, such as unit, smoke, integration, and acceptance testing. For a detailed review of these types of tests, see the testing section of the workload supply chain recommendation guide.

Require DevSecOps practices as part of your standard operating procedures. Your workload team should understand the security practices related to software development and quality assurance. They must follow these practices without exception. For more information, see Security development lifecycle guide.

Implement standards for naming and tagging resources

Implementing tagging and naming conventions is a best practice for managing and organizing Azure resources. Tagging and naming conventions help to identify, classify, and group resources based on common attributes, such as environment, application, owner, or cost center. They also enable security, automation, reporting, and governance of resources across subscriptions and resource groups.

Some of the benefits of using standardized tagging and naming conventions are:

- They provide consistency and clarity for resource identification and management, facilitating discovery and search across the Azure portal, PowerShell, CLI, and APIs.
- They enable filtering and grouping of resources for billing, monitoring, security, and compliance purposes.
- They support resource lifecycle management, such as provisioning, decommissioning, backup, and recovery.
- They're essential for security purposes. If you come upon a security incident, it's critical to quickly identify affected systems, the functions that those systems support, and the potential business impact.

For more information on using naming conventions for your cloud resources, see Define your naming convention. For more information on how to apply metadata tags to your cloud resources, see Define your tagging strategy.

Azure facilitation

- Azure DevOps is a collection of services that you can use to build a collaborative, efficient, and consistent development practice. Azure DevOps bundles the following solutions:
  - Azure Pipelines provides build and release services to support the CI/CD of your applications.
  - Azure Boards is a web-based work-management tool that supports Agile practices like Scrum and Kanban.
  - Azure Repos is a version control tool that supports the Git distributed version control system and the Team Foundation Version Control system.
  - Azure Test Plans is a browser-based test management solution that provides capabilities that are required for planned manual testing, user acceptance testing, exploratory testing, and gathering feedback from stakeholders.
  - Azure Artifacts is used to enable developers to efficiently share their code and manage their packages.
- GitHub Actions for Azure is a tool that you can use to automate CI/CD processes. It integrates directly with Azure to simplify deployments. You can create workflows that build and test every pull request to your repository, or deploy merged pull requests to production.
- GitHub Projects is a work-management tool that you can use to create Kanban boards, reports, dashboards, and other functions.
- Low-code and no-code tools include:
  - Azure Logic Apps
  - Azure Functions
  - Microsoft Power Platform
- Azure Resource Manager templates and Bicep are Azure-native tools that you can use to deploy IaC. Terraform is another Azure-supported IaC tool that you can use to deploy and manage infrastructure.

- Visual Studio is a robust development tool that integrates with Azure and supports many languages.
- GitHub Copilot is an AI service that acts as a pair programmer and provides autocomplete style suggestions as you code. Copilot is available as an extension in Visual Studio and several other development tools.
- Azure Load Testing is a fully managed load testing service that you can use to generate high-scale load by simulating traffic for your applications, regardless of where they're hosted.

Organizational alignment

The Cloud Adoption Framework for Azure provides general guidelines and recommendations for tagging and naming Azure resources, as well as specific rules and examples for different resource types.

Related links

- Adopt a Git branching strategy
- Branch policies and settings
- Cloud design patterns
- Developer velocity
- Develop your naming and tagging strategy for Azure resources
- DevOps resource center
- Enable DevSecOps with Azure and GitHub
- Overview of source code analysis
- Security development lifecycle guide
- Security in DevOps (DevSecOps)
- Shift testing left with unit tests
- Video series: Introduction to GitHub CoPilot

Operational Excellence checklist

Refer to the complete set of recommendations.

Operational Excellence checklist

Recommendations for using infrastructure as code

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Operational Excellence checklist recommendation:

OE:05	**Prepare resources and their configurations by using a standardized infrastructure as code (IaC) approach. Like other code, design IaC with consistent styles, appropriate modularization, and quality assurance. Prefer a declarative approach when possible.**

This guide describes the recommendations for using IaC as the standard for your infrastructure deployments. Using IaC enables you to integrate your infrastructure deployments and management into your existing software development practices. It provides a consistent, standard methodology for development and deployment for all components of your workload. Relying on manual deployments puts your workload at risk of inconsistent configurations and potentially insecure design.

Definitions

Term	Definition
Declarative tools	A category of tools that define the end-state of a deployment and rely on the system to determine how to deploy the resources to match the defined end-state.
Immutable infrastructure	An infrastructure that's intended to be replaced with new infrastructure that runs the new configuration with each deployment. It must not be changed in place.
Imperative tools	A category of tools that list the execution steps that result in the desired end-state.
Module	A unit of abstraction for dividing groups of resources to simplify complex deployments.
Mutable infrastructure	An infrastructure that's intended to be changed in place. Deployments change the configuration of the infrastructure rather than replacing it with new infrastructure.

Key design strategies

As discussed in the supply chain and standardizing tools and processes guides, you should have a strict policy of deploying infrastructure changes (including configuration changes) only through code. You should deploy IaC through your continuous integration and continuous delivery (CI/CD) pipelines. Adopting these policies enforces consistency in processes for all IaC deployments, minimizes the risk of configuration drift across your environments, and ensures infrastructure consistency across your environments. Additionally, you should standardize your IaC development and deployment tools and processes in a style guide. Recommendations for your style guide include:

Prefer declarative over imperative tools. Declarative tools and their associated files are a better overall choice for deploying and managing IaC than imperative tools. Declarative tools use a simpler syntax for their definition files, defining only the desired state of the environment after the deployment finishes. Imperative tools depend on you defining the steps required to get to the desired end-state, so the files can be much more complex than declarative files. Declarative definition files also help reduce the technical debt of maintaining imperative code, like deployment scripts, that can accrue over time.

Use your cloud platform's native tools and other industry-proven tools that natively integrate into the platform. Your cloud platform provides tools to make deploying IaC easy and straightforward. Take advantage of these tools and other third-party tools that have native integration, like Terraform, rather than developing your own solutions. Native tools are supported by the platform and include built-in functionality for most of your needs. They're continuously updated by the platform provider, making them more useful as the platform evolves.

> **Note**
>
> Be mindful that as cloud providers and third-party developers update their tools and APIs, you can run the risk of unanticipated issues when using the latest version in your workload. Ensure that you thoroughly test new versions of tools and APIs before adopting them. Likewise, avoid using the 'latest' flag when calling on a tool or API in your deployment code. Be intentional about calling the latest known good version for your workload.

Use the right tools for specific tasks and infrastructure types. Multiple tasks, beyond deployments, are involved in an infrastructure lifecycle. Configuration needs to be applied and maintained, for example, and the tool you use to script deployments, like Bicep, might not be the best tool for every management operation.

Likewise, applying desired state configuration (DSC) for different infrastructure types might require different tools. For example, there are specific tools like Ansible for managing DSC for VMs, whereas Flux is a good tool for managing DSC on Kubernetes clusters. Platform as a service (PaaS) services might provide different tools for configuration management (like Azure App Configuration) that can be handled through IaC. Software as a service (SaaS) services might be more limited because they're more tightly controlled by the platform.

Think about all the tasks and types of infrastructure that are in scope for your IaC practices and standardize on tools that do the jobs that you need them to do and can be integrated into your development and management practices.

Your scripts and templates should be flexible enough to easily deploy a variety of environments. Use parameters, variables, and configuration files to deploy a standard set of resources that can be modified to deploy any environment in your code promotion stack. Abstract settings like resource size, count, name, locations to deploy into, and some configuration settings. Be careful not to abstract too much, however. There are settings that can be abstracted with a parameter or variable that might not actually change over the course of the workload lifecycle, or that might change rarely. They shouldn't be abstracted.

> **Note**
>
> Avoid using different IaC assets for different environments. You shouldn't have different Terraform files for production and test environments, for example. All environments should use one file. You can manipulate that file to deploy into different environments as needed.

Strategize and standardize on the use of modules. Like parameters and variables, modules can make your infrastructure deployments repeatable. Be thoughtful, however, about how you use them. A standardized abstraction strategy helps ensure that modules are built to meet specific, agreed-upon goals. Use modules to encapsulate complex configurations or combinations of resources. Avoid modules if you're using only the default configuration of the resource. Additionally, be judicious in developing new modules. Use maintained open-source modules when appropriate, in, for example, nonsensitive scenarios.

Document standards for manual steps. There might be steps related to deploying and maintaining infrastructure that are particular to your environment and that require manual intervention. Ensure that these steps are minimized as much as possible and

clearly documented. In your style guide and standard operating procedures, standardize manual steps to ensure that tasks are performed safely and consistently.

Document standards to handle orphaned resources. Depending on the tools you use for configuration management and their limitations, there might be times when a particular resource is no longer needed by your workload and your IaC tools can't automatically remove the resource. For example, say you're moving from VMs to a PaaS service for some function, and the IaC tooling doesn't have logic to remove the retired resources. Those resources can become orphaned if the workload team doesn't remember to manually delete them. To handle these scenarios, standardize a strategy to scan for orphaned resources and delete them. You also need to consider how to ensure that your templates are up to date. Research the limitations of your IaC tooling to understand what you might need to plan for in these situations.

Other IaC strategies

Consider the following recommendations that apply to using IaC for your workload:

Use a layered approach to align your IaC pipelines within the workload stack. Separating your IaC pipelines into layers helps you manage complex environments. Deploying dozens or hundreds of resources as a monolithic package is inefficient and can introduce multiple issues, like broken dependencies. The use of multiple pipelines that are aligned with layers composed of resources whose deployment lifecycles or factors like functionality closely match makes managing IaC deployments easier.

Core infrastructure like networking resources rarely need changes more complex than configuration updates, so those resources should make up a *low-touch* IaC pipeline. You might have one or more *medium-touch* and *high-touch* IaC pipelines for resources, depending on the complexity of your workload. Using a Kubernetes-based application stack as an example, one medium-touch layer might be composed of the clusters, storage resources, and database services. High-touch layers would be composed of the application containers that are updated very frequently in a continuous delivery mode.

Treat your IaC and application code the same. Treating your IaC artifacts the same as your application code artifacts helps you apply the same rigor for managing code across all pipelines. Moreover, IaC development and deployment practices should mirror application practices. Standards for version control, branching, code promotion, and quality should all be identical. Also consider collocating your IaC assets together with your application code assets. Doing so helps ensure that the same processes are followed with every deployment and helps you avoid issues like inadvertently deploying infrastructure before the necessary application code, or vice versa.

Collaborate with other teams in your organization for standardization and reusability. Large organizations can sometimes have multiple teams developing and supporting workloads. Collaboration across teams to agree on standards helps you reuse libraries, templates, and modules to gain efficiency and consistency across workload environments. Likewise, IaC tools should be standardized across the organization to the extent that doing so is practical.

Apply the principle of "security as code" to ensure that security is part of the deployment pipeline. Include vulnerability scanning and configuration hardening as part of the IaC development process. Scan your IaC repos for keys and secrets that are exposed. One advantage of using IaC is that security-focused team members can review code before deployment to ensure that the configuration that's approved for release by security is actually what's deployed to production. For detailed guidance, see Recommendations for securing a development lifecycle.

Test routine and non-routine activities. Test deployments, configuration updates, and recovery processes, including deployment-rollback processes.

Mutable vs. immutable infrastructure

The choice between deploying mutable versus immutable infrastructure depends on a few factors. If your workload is business critical, it's best to use immutable infrastructure. Likewise, if you have a mature infrastructure design that's based on deployment stamps, using immutable infrastructure can make sense, because you can deploy application code and new infrastructure reliably. Conversely, using mutable infrastructure can be a better choice if your safe deployment practices dictate that rolling forward with deployments when mitigable deployment issues arise is the preferred option. In this case, you would probably update the infrastructure in place.

Azure facilitation

Azure Resource Manager templates (ARM templates) and Bicep are Azure-native tools for deploying infrastructure by using declarative syntax. ARM templates are written in JSON, whereas Bicep is a domain-specific language. Both can easily be integrated into Azure DevOps pipelines or GitHub Actions CI/CD pipelines.

Terraform is another declarative IaC tool that's fully supported in Azure. It can be used to deploy and manage infrastructure, and can be integrated into your CI/CD pipeline.

You can use Microsoft Defender for Cloud to discover misconfigurations in IaC.

Organizational alignment

Cloud Adoption Framework provides guidance for central teams on how to standardized infrastructure as code across the organization's workload teams.

For more information, see Infrastructure as Code in the Cloud Adoption Framework.

Tradeoffs

Increased specialization: In some cases, introducing new languages in your workload team comes with a learning curve, and vendor lock-in can make it a poor choice. Training your team members and analyzing the right tool based on your cloud providers' tooling support is required.

Increased maintenance effort: Code base and tooling maintenance are required to keep your IaC implementation current and secure. Properly track your technical debt and foster a culture where reducing debt is rewarded.

Increased time for configuration changes: Deploying infrastructure by using command-line instructions or directly from a portal requires no coding time and/or testing artifacts. Minimize deployment time by following recommended practices like code reviews and quality assurance practices.

Increased complexity of modularization: Using more modules and parameterization increases the time it takes to debug and document the system and adds a layer of abstraction. Balance the use of modularization to reduce complexity and avoid over-engineering.

Example

See the Azure Virtual Desktop landing zone accelerator architecture and the associated reference implementation for an example of a Virtual Desktop implementation that can be deployed via provided Resource Manager, Bicep, or Terraform files.

Related links

- What is infrastructure as code (IaC)?
- Enterprise infrastructure as code using Bicep and Azure Container Registry
- Discover misconfigurations in IaC
- Recommendations for designing a workload development supply chain
- Recommendations for standardizing tools and processes

- Recommendations for securing a development lifecycle
- Recommendations for using safe deployment practices
- Deployment Stamps pattern
- Azure Resource Manager templates (ARM templates)
- Bicep
- Azure DevOps pipelines
- GitHub Actions
- Terraform

Operational Excellence checklist

Refer to the complete set of recommendations.

Operational Excellence checklist

Recommendations for designing a workload development supply chain

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Operational Excellence checklist recommendation:

OE:06	**Build a workload supply chain that drives proposed changes through predictable, automated pipelines. The pipelines test and promote those changes across environments. Optimize a supply chain to make your workload reliable, secure, cost effective, and performant.**

This guide describes the recommendations for designing a workload development supply chain that's based on continuous integration and continuous delivery (CI/CD) pipelines. Develop a supply chain to ensure that you have a predictable, standardized method of maintaining your workload. CI/CD pipelines are the manifestation of the supply chain, but you should have a single supply chain. And you might have several pipelines that follow the same processes and use the same tools.

Incorporate a supply chain to protect your workload from damage that can occur when you don't properly manage workload changes. Always be aware of the state of your workload, so you're not at risk of experiencing unpredictable behavior. This risk compounds if you need to spend critical time retracing unaccounted for changes when issues arise. To minimize these risks, standardize the processes and tools that define your supply chain, and ensure that your workload team fully commits to their use.

Definition

Term	Definition
Supply chain	In cloud workloads, a supply chain is a standardized suite of tools and processes that you use to affect infrastructure and application change across environments.

Key design strategies

> ⓘ **Note**
>
> The recommendations in this guide refer to workload environments in a code promotion chain. Sandbox or other exploratory and proof-of-concept environments require less rigor and structure.

Core tenets

The following recommendations can help you define the core tenets of your supply chain.

Make proposed workload changes through supply chain processes and tools. Enforce a strict policy of automated template-based deployments. This method helps ensure that your workload's configuration across all environments is standardized, well-defined, and tightly controlled. For environments in a code promotion chain, don't perform updates by using manual processes or human interaction with the cloud platform's control plane, for example the portal or an API. Incorporate all changes to the environment through a pipeline by following deployment practices that you define. To help enforce this policy, consider limiting access to read-only as a default and using an access authorization gate to allow write access.

An important aspect of this tenet is that all changes are *proposed changes* until they're deployed into production. Through automated testing, like integration and smoke testing, you enable your supply chain to automatically reject changes.

Deploy repeatable and immutable infrastructure as code (IaC). IaC is the management of infrastructure in a descriptive model that uses a versioning system that mirrors source code. When you create an application, the same source code should generate the same binary every time it's compiled. In a similar manner, an IaC model generates the same environment every time it's applied.

Incorporate IaC to ensure that your team follows a standard, well-established process. Some organizations designate a single individual or small group of individuals to deploy and configure infrastructure. When you implement a fully automated process, infrastructure deployments require less management from individuals. The responsibility is transferred to the automation process and tooling. Team members can initiate infrastructure deployments to maintain consistency and quality.

Design your workload as a logical group of components that you can bundle into one template to make deployments easy and repeatable. You can think of these bundles as *stamps* or *units of scale*. For more information, see Deployment Stamps pattern. When you need to deploy your workload to scale out into another region or zone within the same region, deploy a stamp by using a pipeline. Depending on how you design your stamps, you might deploy a subset of your workload instead of the entire workload. Include networking components in your IaC pipelines to ensure that your deployment stamps automatically connect to existing resources.

To optimize your IaC pipeline for consistency and efficiency, design an immutable infrastructure rather than a mutable infrastructure. Implement an immutable

infrastructure to ensure that all systems in scope are replaced with the updated configuration simultaneously and identically with each deployment.

Use one set of code assets and artifacts across all environments and pipelines. A common pain point for organizations is when nonproduction environments are different from production environments. Building production and nonproduction environments manually can result in mismatched configurations between the environments. This mismatch slows down testing and makes it more likely that changes might harm a production system. An IaC approach minimizes these problems. When you use IaC automation, you can use the same infrastructure configuration files for all environments to produce almost identical environments. You can add parameters to the infrastructure configuration files and adjust them to meet the requirements for each environment.

To control costs, there's typically a variance between production and nonproduction environments. You often don't need the same degree of redundancy and performance in nonproduction environments as you do in production. The number and SKU of your resources might differ between environments. Ensure that you control and understand the variance by using standardized parameters to help you maintain predictability as you make changes.

Reflect your organizational structure in your supply chain and pipeline designs. Your organization might be siloed among teams. Each team might manage a part of the supply chain. For example, many organizations have teams that manage infrastructure domains, like networking, data, and compute resources. These teams are separate from development teams that manage application development, testing, and deployments. In some organizations, development and infrastructure teams are incorporated into DevOps teams that collectively manage all infrastructure and application deployments. There are many ways to organize the teams that are involved in a supply chain. Your supply chain relies on all the teams seamlessly working together. Ensure that all teams follow standard processes and use standard tools to make the supply chain as efficient as possible.

Your supply chain might rely on third-party vendors if you outsource parts of the workload lifecycle. These vendors are just as critical to the success of your supply chain as internal resources. Ensure that there's a mutual agreement across all teams about the processes and tools that you use.

Standardize your deployment method. Talk to the product owner about the acceptable amount of production downtime for your workload. Depending on how much, if any, downtime is acceptable, you can choose the deployment method that's right for your requirements. Ideally, you should perform deployments during a maintenance window

to reduce complexity and risk. If no downtime is acceptable, employ a blue-green deployment method.

Use a progressive-exposure approach to reduce the risk of introducing undetected bugs to your customers at large. Also known as canary deployments, this method deploys to controlled groups of users in a gradual sequence. It catches issues before they affect more users. The initial rollout group might be a subsection of your customers that are aware of the rollout strategy. This subsection of customers can tolerate some amount of unexpected behavior and provide feedback. Or it might be a group of internal users, which helps contain the blast radius of bugs during the rollout.

When you define your deployment method, adopt a standard policy of only deploying the smallest viable change in each deployment. Depending on factors like the criticality of the workload and complexity of the components, determine the smallest viable change. If you use an immutable infrastructure, the smallest viable change is typically the deployment of resources with the latest configuration to replace resources that run the previous version. If you use a mutable infrastructure, you might decide that the smallest viable change is only a single update on the group of resources in scope.

Follow a layered approach to reflect different lifecycles. Foundational layers should remain static throughout most of the workload lifecycle, and application layers change frequently. To account for these differences, you should have different pipelines to effect changes at each layer.

A landing zone is at the lowest layer. A landing zone is a logical grouping of foundational elements, like subscriptions, management groups, resource groups, governance functions, and networking topology. A landing zone enables you to easily deploy and manage your workload, and provides central operations teams, or platform teams, with a repeatable approach to an environmental configuration. To deliver consistent environments, all Azure landing zones provide a set of common design areas, a reference architecture, a reference implementation, and a process to modify a deployment to fit your design requirements. The Azure landing zone design principles provide recommended practices based on policy-driven governance alongside subscription democratization. A landing zone should require minimal changes over the course of your workload lifecycle.

Your core infrastructure and functions, like ingress and egress network controllers, load balancing, network routing solutions, DNS management, and core servers, should also require infrequent major changes. But they might require frequent configuration updates.

Your application and data layer requires frequent configuration updates and frequent infrastructure changes. These components should have the most dynamic pipelines.

Plan for a holistic testing strategy. A core tenet of system reliability is the *shift left* principle. Developing and deploying an application is a process that's depicted as a series of steps going from left to right. You shouldn't limit testing to the end of the process. As much as possible, shift testing to the beginning, or to the left. Errors are cheaper to repair when you catch them early. They can be expensive or impossible to fix later in the application lifecycle.

Test all code, including application code, infrastructure templates, and configuration scripts. The environment that runs applications should be version-controlled and deployed through the same mechanisms as application code. You can test and validate the environment by using the same testing paradigms that your teams already use for application code.

When possible, use automated testing to ensure consistency. Include the following types of testing in your supply chain design.

- *Unit testing*: Unit tests are typically run as part of a continuous integration routine. Unit tests should be extensive and quick. They should ideally cover 100 percent of the code and run in under 30 seconds.

  Implement unit testing to verify that the syntax and functionality of individual modules of code work the way that they should, for example producing a defined output for a known input. You can also use unit tests to verify that IaC assets are valid.

  Apply unit tests to all code assets, including templates and scripts.

- *Smoke testing*: Smoke tests verify that a workload can be stood up in a test environment and performs as expected. Smoke tests don't verify the interoperability of components.

  Smoke tests verify that the deployment methodology for the infrastructure and the application works, and that the system responds as intended after the process finishes.

- *Integration testing*: Integration tests ensure that the application components operate individually, and then determine whether components can interact with each other as they should.

  It can take a considerable amount of time to run a large integration test suite. That's why it's best to incorporate the shift left principle and perform testing early in the software development lifecycle. Reserve integration tests for scenarios that you can't test with a smoke test or unit test.

You can run long-running test processes on a regular interval if needed. A regular interval offers a good compromise and detects interoperability issues between application components no later than one day after they're introduced.

Some testing scenarios benefit from manual runs. Use manual testing when you need to introduce human interactivity elements into tests.

- *Acceptance testing*: Depending on the context, you can manually perform acceptance testing. It can be partially or fully automated. Acceptance testing determines whether the software system meets the requirement specifications.

  The main purpose of this test is to evaluate the system's compliance with the business requirements and determine whether the system meets the required criteria for delivery to end users.

Implement quality gates throughout your code promotion process via testing. Deploy your code into lower environments, like development and testing, and up through higher environments, like staging and production. As your deployment passes through quality gates, ensure that it meets your quality targets before changes go to production. Your business requirements determine what the focus of your quality gates are. Also consider the fundamental Well-Architected Framework principles: Security, Reliability, and Performance Efficiency.

Also integrate approval workflows into your quality gates. Clearly define and automate approval workflows when appropriate. Define quality acceptance criteria into your automation, so you can move through your gates efficiently and safely.

Azure facilitation

Azure DevOps is a collection of services that help you build a collaborative, efficient, and consistent development practice.

Azure Pipelines provides build and release services to support CI/CD in your applications.

GitHub Actions for Azure integrates with Azure to simplify deployments. Use GitHub Actions to automate CI/CD processes. You can create workflows that build and test every pull request to your repository, or deploy merged pull requests to production.

You can use Terraform, Bicep, and Azure Resource Manager for IaC deployments. Depending on your requirements and your team's familiarity with the tools, you might use one or more of these tools for your deployments and management of the resources.

Organizational alignment

Cloud Adoption Framework provides guidance for central teams to provide workload landing zones. The workload landing zones are where the workload's custom supply chain will deploy applications into.

For more information, see Landing zones and the Azure landing zone design principles.

Tradeoffs

In your lower environments, there's a tradeoff between cost and infrastructure parity with production. Each successively higher environment should be closer in parity to production to ensure that your quality gates are as effective as possible.

Example

For an example that shows how to use Azure Pipelines to build a CI/CD pipeline, see Azure Pipelines baseline architecture.

Related links

- Azure DevOps
- Azure Pipelines
- Deployment Stamps pattern
- GitHub Actions for Azure
- Landing zones
- Performance Efficiency pillar
- Reliability pillar
- Security pillar

Operation Excellence checklist

Refer to the complete set of recommendations.

Operational Excellence checklist

Recommendations for designing and creating a monitoring system

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Operational Excellence checklist recommendation:

OE:07	**Design and implement a monitoring system to validate design choices and inform future design and business decisions. This system captures and exposes operational telemetry, metrics, and logs that emit from the workload's infrastructure and code.**

Related guide: Recommendations for instrumenting an application

This guide describes the recommendations for designing and creating a monitoring system. To effectively monitor your workload for security, performance, and reliability, you need a comprehensive system with its own stack that provides the foundation for all monitoring, detection, and alerting functions.

Definitions

Term	Definition
Logs	Recorded system events. Logs can contain different types of data in a structured or free-form text format. They contain a timestamp.
Metrics	Numerical values that are collected at regular intervals. Metrics describe some aspects of a system at a particular time.

Key design strategies

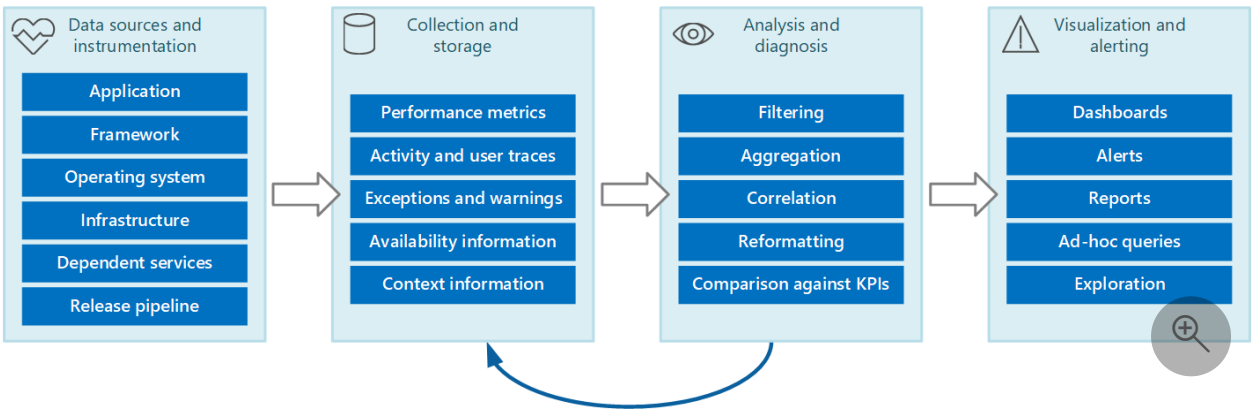
To implement a comprehensive monitoring system design for your workload, follow these core tenets:

- Whenever practical, take advantage of platform-provided monitoring tools, which typically require very little configuration and can provide deep insights into your workload that might otherwise be difficult to accomplish.
- Collect logs and metrics from the entire workload stack. All infrastructure resources and application functions should be configured to produce standardized, meaningful data, and that data needs to be collected.
- Store the collected data in a standardized, reliable, and secure storage solution.

- Process stored data so that it can be handled by analysis and visualization solutions.
- Analyze processed data to accurately determine the state of the workload.
- Visualize the state of the workload in meaningful dashboards or reports for workload teams and other stakeholders.
- Configure actionable alerts and other automatic responses to intelligently defined thresholds to notify workload teams when issues arise.
- Include monitoring and alerting systems in your overall workload testing practices.
- Ensure that monitoring and alerting systems are in scope for continuous improvement. Application and infrastructure behavior in production provides continuous learning opportunities. Incorporate those lessons into monitoring and alerting designs.
- Tie the monitoring data that you gather and analyze back to your system and user flows to correlate the health of the flows with the data in addition to the overall health of the workload. Analyzing that data in terms of the flows will help align your observability strategy with your health model.

You should automate all functions of the monitoring system as much as possible, and they should all run continuously, all day, every day.

This workflow pipeline illustrates the monitoring system:



Collection

> **Note**
>
> You need to instrument your application to enable logging. For more information, see the **instrumentation guide**.

You should configure all workload components, whether they're infrastructure resources or application functions, to capture telemetry and/or events like logs and metrics.

Logs are primarily useful for detecting and investigating anomalies. Typically, logs are produced by the workload component and then sent to the monitoring platform or pulled by the monitoring platform via automation.

Metrics are primarily useful for building a health model and identifying trends in workload performance and reliability. Metrics are also useful for identifying trends in the usage behavior of your customers. These trends can help guide decisions about improvements from the customer perspective. Typically, metrics are defined in the monitoring platform, and the monitoring platform and other tools poll the workload to capture metrics.

Application data

For applications, the collecting service can be an application performance management (APM) tool that can run autonomously from the application that generates the instrumentation data. After APM is enabled, you have clear visibility into important metrics, in real time and historically. Use an appropriate level of logging. Verbose logging can incur significant costs. Set log levels according to the environment. Lower environments don't need the same level of verbosity as production, for example.

Application logs support the end-to-end application lifecycle. Logging is essential to understanding how the application operates in various environments, which events occur, and the conditions under which they occur.

We recommend that you collect application logs and events across all major environments. Separate the data between environments as much as possible by using different data stores for each environment, if doing so is practical. Use filters to ensure that noncritical environments don't complicate the interpretation of production logs. Finally, corresponding log entries across the application should capture a correlation ID for their respective transactions.

You should capture application events in structured data types with machine-readable data points rather than unstructured string types. A structured format that uses a well-known schema can make parsing and analyzing logs easier. Also, structured data can easily be indexed and searched, and reporting can be greatly simplified.

Data should be in an agnostic format that's independent of the machine, operating system, or network protocol. For example, emit information in a self-describing format like JSON, MessagePack, or Protobuf rather than ETL/ETW. A standard format enables

the system to construct processing pipelines. Components that read, transform, and send data in the standard format can be easily integrated.

Infrastructure data

For infrastructure resources in your workload, ensure that you collect both logs and metrics. For infrastructure as a service (IaaS) systems, capture OS, application-layer, and diagnostic logs in addition to metrics related to workload health. For platform as a service (PaaS) resources, you might be limited in your ability to capture logs that are related to underlying infrastructure, but be sure that you can capture diagnostic logs in addition to metrics related to workload health.

As much as possible, collect logs from your cloud platform. You might be able to collect activity logs for your subscription and diagnostic logs for the management plane.

Collection strategies

Avoid retrieving telemetry data manually from every component. Move data to a central location and consolidate it there. For a multi-region solution, we recommend that you first collect, consolidate, and store data on a region-by-region basis, and then aggregate the regional data into a single central system.

>  **Tradeoff**: Be aware that there are cost implications to having regional and centralized data stores.

To optimize the use of bandwidth, prioritize based on the importance of data. You can transfer less urgent data in batches. However, this data must not be delayed indefinitely, especially if it contains time-sensitive information.

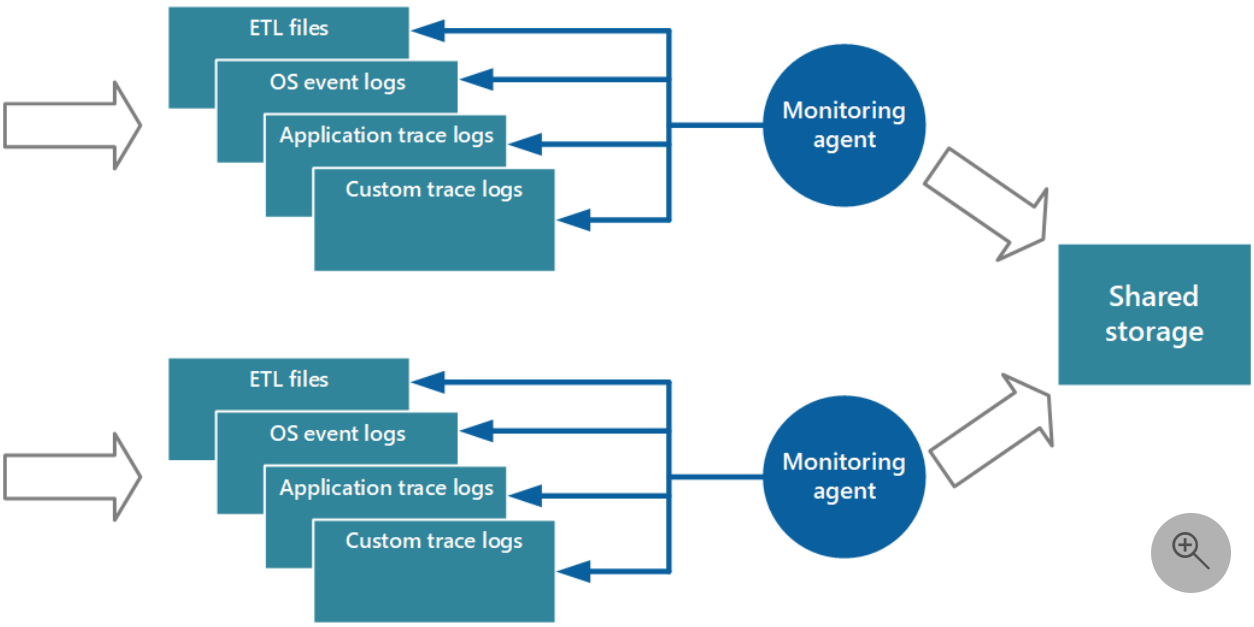
There are two primary models that the collection service can use to collect instrumentation data:

- **Pull model**: Actively retrieves data from the various logs and other sources for each instance of the application.
- **Push model**: Passively waits for the data to be sent from the components that constitute each instance of the application.

Monitoring agents

You can use monitoring agents in the pull model. Agents run locally in a separate process with each instance of the application, periodically pulling data and writing the

information directly to common storage that's shared by all instances of the application.

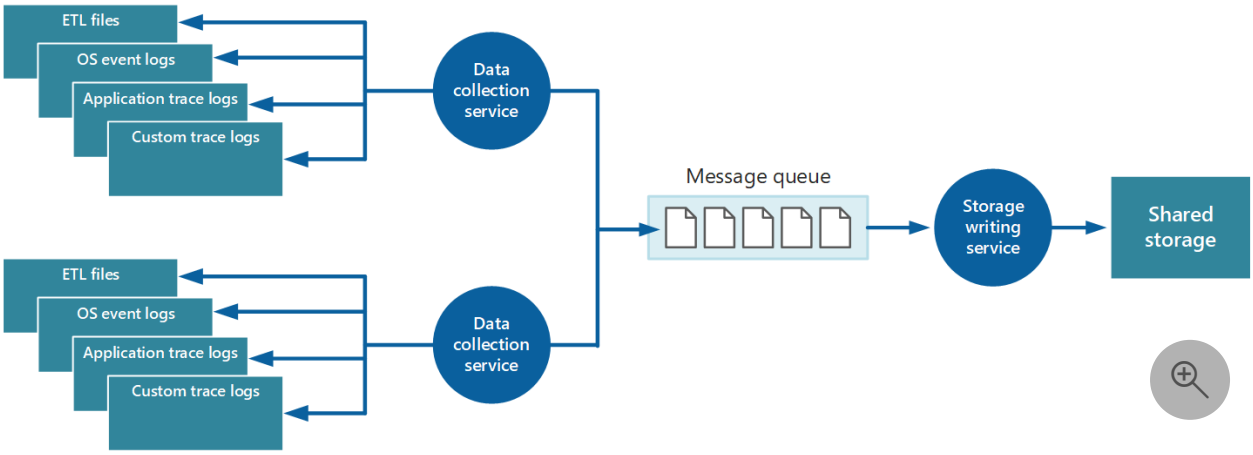


> **Note**
>
> Using a monitoring agent is ideally suited to capturing instrumentation data that's naturally pulled from a data source. It's appropriate for a small-scale application that runs on a limited number of nodes in a single location. Examples include information from SQL Server dynamic management views or the length of an Azure Service Bus queue.

Performance considerations

A complex and highly scalable application might generate huge volumes of data. The amount of data can easily overwhelm the I/O bandwidth available for a single, central location. The telemetry solution must not act as bottleneck and must be scalable as the system expands. Ideally, the solution should incorporate a degree of redundancy to reduce the risks of losing important monitoring information (like auditing or billing data) if part of the system fails.

One way to buffer instrumentation data is to use queuing:

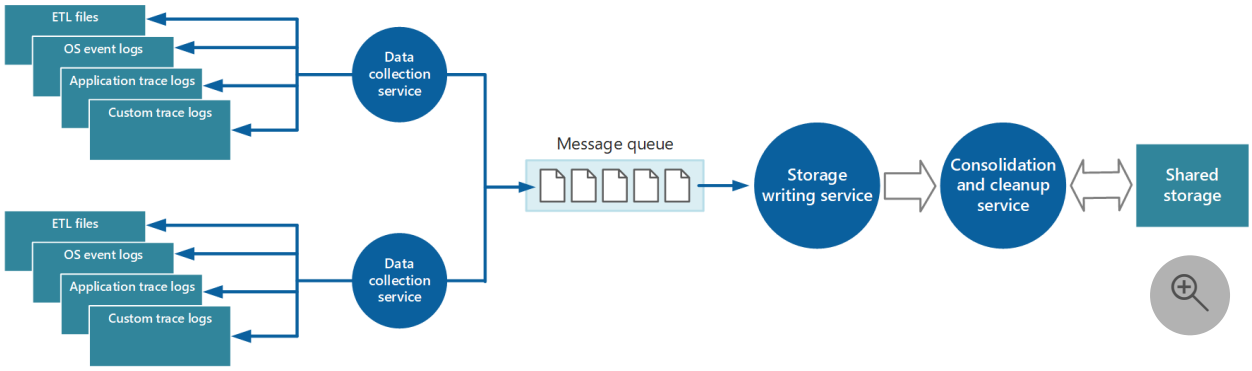


In this architecture, the data-collection service posts data to a queue. A message queue is suitable because it provides "at least once" semantics that helps ensure that queued data won't be lost after it's posted. You can implement the storage-writing service by using a separate worker role. You can use the Priority Queue pattern to implement this architecture.

For scalability, you can run multiple instances of the storage-writing service. If a high volume of events or a high number of data points is being monitored, you can use Azure Event Hubs to dispatch the data to a different compute instance for processing and storage.

Consolidation strategies

The data collected from a single instance of an application provides a localized view of the health and performance of that instance. To assess the overall health of the system, you need to consolidate some aspects of the data from the local views. You can do that after the data is stored, but, in some cases, you can do it as the data is collected.



The instrumentation data can pass through a separate data consolidation service that combines data and acts as a filter and cleanup process. For example, you can amalgamate instrumentation data that includes the same correlation information, like an activity ID. (A user might start a business operation on one node and then get transferred to another node if the first node fails, or because of how load balancing is configured.) This process can also detect and remove any duplicated data. (Duplication

can occur if the telemetry service uses message queues to push instrumentation data out to storage.)

Storage

When you choose a storage solution, consider the type of data, how it's used, and how urgently it's required.

> **Note**
>
> Use separate storage solutions for non-production and production environments to ensure that data from each environment is easy to identify and manage.

Storage technologies

Consider a polyglot persistence approach, where different types of information are stored in technologies that are most appropriate to the way each type is likely to be used.

For example, Azure Blob Storage and Azure Table Storage are accessed in similar ways. But the operations that you can perform on them differ, as does the granularity of the data that they hold. If you need to perform more analytical operations or require full-text search capabilities on the data, it might be more appropriate to use data storage that provides capabilities that are optimized for specific types of queries and data access. For example:

- Performance counter data can be stored in a SQL database to enable ad hoc analysis.
- It might be better to store trace logs in Azure Monitor Logs or Azure Data Explorer.
- You might store security information in an HDFS solution.

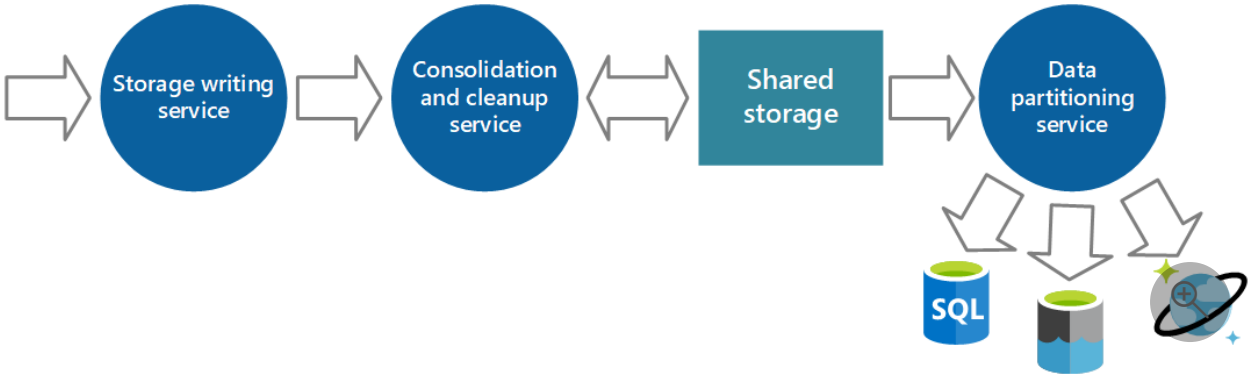
The same instrumentation data might be required for more than one purpose. For example, you can use performance counters to provide a historical view of system performance over time. This information might be combined with other usage data to generate customer billing information. In these situations, the same data might be sent to more than one destination, like to a document database that can be a long-term store for holding billing information, and to a multidimensional store for handling complex performance analytics.

Be sure to enable functionality to protect the data from accidental deletion, like resource locks and soft delete.

Also, be sure that you secure access to storage by using role-based access control to help ensure that only individuals who need to access the data can do so.

Consolidation service

You can implement another service that periodically retrieves the data from shared storage, partitions and filters it according to its purpose, and then writes it to an appropriate set of data stores.



An alternative approach is to include this functionality in the consolidation and cleanup process and write the data directly to these stores as it's retrieved rather than saving it in an intermediate shared storage area.

Each approach has its advantages and disadvantages. Implementing a separate partitioning service reduces the load on the consolidation and cleanup service, and it enables at least some of the partitioned data to be regenerated if necessary (depending on how much data is retained in shared storage). However, this approach consumes additional resources. Also, there might be a delay between the receipt of instrumentation data from each application instance and the conversion of this data into actionable information.

Querying considerations

Consider how urgently the data is required. Data that generates alerts must be accessed quickly, so it should be held in fast data storage and indexed or structured to optimize the queries that the alerting system performs. In some cases, it might be necessary for the collection service to format and save data locally so that a local instance of the alerting system can send notifications quickly. The same data can be dispatched to the storage writing service shown in the previous diagrams and stored centrally if it's also required for other purposes.

Data retention considerations

In some cases, after data is processed and transferred, you can remove the original raw source data that was stored locally. In other cases, it might be necessary or useful to save the raw information. For example, you might want to keep data that's generated for debugging available in its raw form but then discard it quickly after any bugs are resolved.

Performance data often has a longer life so that you can use it for spotting performance trends and for capacity planning. The consolidated view of this data is usually kept online for a finite period to enable fast access. After that, it can be archived or discarded.

It's useful to store historical data so you can spot long-term trends. Rather than saving old data in its entirety, you might be able to down-sample the data to reduce its resolution and save storage costs. For example, rather than saving minute-by-minute performance indicators, you can consolidate data that's more than a month old to form an hour-by-hour view.

Data gathered for metering and billing customers might need to be saved indefinitely. Additionally, regulatory requirements might dictate that information collected for auditing and security needs to be archived and saved. This data is also sensitive and might need to be encrypted or otherwise protected to prevent tampering. You should never record user passwords or other information that might be used to commit identity fraud. You should scrub these details from the data before it's stored.

To ensure that you comply with laws and regulations, minimize the storage of any identifiable information. If you do need to store identifiable information, be sure, when you design your solution, to take into account requirements that allow individuals to request that their information be deleted.

Analysis

After you collect data from various data sources, analyze it to assess the overall well-being of the system. For this analysis, have a clear understanding of:

- How to structure data based on KPIs and performance metrics that you've defined.
- How to correlate the data captured in different metrics and log files. This correlation is important when you're tracking a sequence of events and can help you diagnose problems.

In most cases, data for each component of the architecture is captured locally and then accurately combined with data that's generated by other components.

For example, a three-tier application might have:

- A presentation tier that allows a user to connect to a website.
- A middle tier that hosts a set of microservices that processes business logic.
- A database tier that stores data associated with the operation.

The usage data for a single business operation might span all three tiers. This information needs to be correlated to provide an overall view of the resource and processing usage for the operation. The correlation might involve some preprocessing and filtering of data on the database tier. On the middle tier, aggregation and formatting are common tasks.

Recommendations

- **Correlate application-level and resource-level logs.** Evaluate data at both levels to optimize the detection of issues and the troubleshooting of those issues. You can aggregate the data in a single data sink or take advantage of methods that query events across both levels. We recommend a unified solution, like Azure Log Analytics, to aggregate and query application-level and resource-level logs.
- **Define clear retention times on storage for cold analysis.** We recommend this practice to enable historic analysis over a specific period. It can also help you control storage costs. Implement processes that ensure data is archived to cheaper storage and aggregate data for long-term trend analysis.
- **Analyze long-term trends to predict operational issues.** Evaluate long-term data to form operational strategies and also to predict what operational issues are likely to occur, and when. For example, you might note that average response times are slowly increasing over time and approaching the maximum target.

For detailed guidance about these recommendations, see Analyze monitoring data for cloud applications.

Visualization

Dashboards

The most common way to visualize data is to use dashboards that can display information as a series of chart or graphs, or in some other visual form. These items can be parameterized, and an analyst can select the important parameters, like the time period, for any specific situation.

Align your dashboards with your health model so that they indicate when the workload or components of the workload are healthy, degraded, or unhealthy.

For a dashboard system to work effectively, it must be meaningful to the workload team. Visualize information that relates to workload health and that's also actionable. When the workload or a component is degraded or unhealthy, members of the workload team should be able to easily identify where in the workload the issue originates and begin their corrective actions or investigations. Conversely, including information that isn't actionable or that's not related to workload health can make the dashboard needlessly complex and frustrating to team members who are trying to discern background noise from actionable data.

You might have dashboards for stakeholders or developers that are customized to only show data about the workload that they find relevant. Ensure that the workload team understands the types of data points that other teams are interested in seeing and previews the dashboards before sharing them to check for clarity. Providing dashboards about your workload for stakeholders is a good way to keep them apprised of the workload health, but carries a risk of being counterproductive if the stakeholders don't clearly understand the data they see.

A good dashboard doesn't just display information. It also enables an analyst to pose improvised questions about that information. Some systems provide management tools that an operator can use to complete these tasks and explore the underlying data. Instead, depending on the repository that's used to hold the information, it might be possible to query the data directly or import it into tools like Excel for further analysis and reporting.

> **Note**
>
> Restrict dashboard access to authorized personnel. Information on dashboards might be commercially sensitive. You should also protect the underlying data to prevent users from changing it.

Reporting

Reporting is used to generate an overall view of the system. It might incorporate historical data and current information. Reporting requirements fall into two broad categories: operational reporting and security reporting.

Operational reporting typically includes the following:

- Aggregating statistics that you can use to understand resource utilization of the overall system or specified subsystems during a specified time window.
- Identifying trends in resource usage for the overall system or specified subsystems during a specified period.
- Monitoring exceptions that have occurred throughout the system or in specified subsystems during a specified period.
- Determining the efficiency of the application for the deployed resources, and understanding whether the volume of resources, and their associated costs, can be reduced without affecting performance unnecessarily.

Security reporting tracks customer use of the system. It can include:

- Auditing user operations. This task requires recording the individual requests that each user completes, together with dates and times. The data should be structured to enable an administrator to quickly reconstruct the sequence of operations that a user completes during a specified period.
- Tracking resource use by user. This task requires recording how each request from a user accesses the various resources that compose the system, and for how long. An administrator can use this data to generate a utilization report, by user, for a specified period, possibly for billing.

In many cases, batch processes can generate reports according to a defined schedule. Latency isn't normally an issue. You should also have batch processes that can generate reports on a spontaneous basis, as needed. For example, if you store data in a relational database like Azure SQL Database, you can use a tool like SQL Server Reporting Services to extract and format data and present it as a set of reports.

Alerts

To help ensure that the system remains healthy, responsive, and secure, set alerts so that operators can respond to them in a timely manner. An alert can contain enough contextual information to help them quickly get started on diagnostic activities. Alerting can be used to invoke remediation functions like autoscaling or other self-healing mechanisms. Alerts can also enable cost-awareness by providing visibility into budgets and limits.

Recommendations

- Define a process for alert response that identifies the accountable owners and actions.
- Configure alerts for a well-defined scope (resource types and resource groups) and adjust the verbosity to minimize noise.
- Use an automated alerting solution, like Splunk or Azure Monitor, instead of requiring people to actively look for issues.
- Use alerts to operationalize remediation processes. For example, automatically create tickets to track issues and resolutions.
- Track the health of your cloud platform services in regions, communication about outages, planned maintenance activities, and other health advisories.

Thresholds

Alerts are generated when thresholds are crossed, as detected by your monitoring system. Ensure that the thresholds you set generally give you enough time to implement the necessary changes to your workload to avoid degradation or outages. For example, set your automatic scaling threshold to initiate scaling before any of the running systems become overwhelmed to the point of a degraded user experience. Base the threshold values that you assign on your past experience in managing infrastructure and validate them through the testing that you perform as part of your testing practices.

For detailed guidance on alerting use cases and other considerations, see Designing a reliable monitoring and alerting strategy.

Azure facilitation

- Azure Monitor is a comprehensive monitoring solution for collecting, analyzing, and responding to monitoring data from your cloud and on-premises environments.
- Log Analytics is a tool in the Azure portal that you can use to edit and run log queries against data in the Log Analytics workspace.

  If you're using multiple workspaces, see the Log Analytics workspace architecture guide for best practices.
- Application Insights is an extension of Azure Monitor. It provides APM features.

- Azure Monitor Insights are advanced analytics tools for specific Azure technologies (like VMs, app services, and containers). These tools are part of Azure Monitor and Log Analytics.
- Azure Monitor for SAP solutions is an Azure monitoring tool for SAP landscapes that run on Azure.
- Azure Policy can help you enforce organizational standards and assess compliance at scale.

Tradeoffs

Storing logs and telemetry data, running queries against that data, and other factors like replication all have cost implications that you need to consider when you plan your strategy. Consider options like archive storage and selective replication when they're practical.

Related links

- Instrumentation guide
- Recommendations for designing a reliable monitoring and alerting strategy
- Recommendations for monitoring and threat detection
- Recommendations for collecting performance data

Operational Excellence checklist

Refer to the complete set of recommendations.

Operational Excellence checklist

Recommendations for instrumenting an application

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Operational Excellence checklist recommendation:

OE:07	**Design and implement a monitoring system to validate design choices and inform future design and business decisions. This system captures and exposes operational telemetry, metrics, and logs that emit from the workload's infrastructure and code.**

Related guide: Recommendations for designing and creating a monitoring system

This guide describes the recommendations for enabling observability of your application by using instrumentation. Generate meaningful telemetry that can be ingested and integrated into your monitoring system. By using instrumentation, you can gather information without signing in to a remote production server to manually perform tracing or debugging. Instrumentation data includes metrics and logs that you can use to assess performance, diagnose problems, and make workload decisions.

Key design strategies

To optimize telemetry for your workload, instrument your application to generate the following data:

- Logs are timestamped records of discrete events. There are three forms of logs: plain text, structured, and binary.
- Distributed tracing logs allow you to see the path of a request as it travels through different services and components.
- Metrics are numerical values that describe an aspect of a system at a particular point in time.

> **Note**
>
> You can use tools like Application Insights, Dynatrace, and Elastic Application Performance Monitoring to automatically instrument your application. These tools make instrumentation easier, but they can also be limiting. If you use an automatic

instrumentation tool, you can add more capabilities through manual instrumentation as needed.

Logs and distributed tracing logs

Use structured logging to easily integrate logs into monitoring and analysis platforms. Instrument your application so the levels of verbosity can be changed. Constant verbose logging can waste storage resources, so it should be switched on and off as needed for troubleshooting.

Trace logs contain textual data or binary data that's created from a trace event, if the application uses Event Tracing for Windows (ETW). System logs generate trace log content from events in the infrastructure, such as the web server. Textual log content is designed to be readable by humans, but you should ensure that it's written in a format that an automated system can parse as well.

Categorize logs and use separate logs to record the trace output from each operational aspect of the system. If you categorize your logs, you can quickly filter log messages instead of processing a single lengthy file. Never write information that has different security requirements, such as audit information and debugging data, to the same log.

> **Note**
>
> A log might be implemented as a file in the file system, or it might be held in some other format, such as a blob in blob storage. Log information might also be held in structured storage, such as rows in a table.

Metrics

Metrics, or *samples*, are a count of some aspect or resource in the system at a specific time, with one or more associated tags or dimensions. A single instance of a metric isn't useful in isolation, metrics should be captured over time. Consider which metrics you should record and how frequently. Data that's generated too often can impose a heavy load on the system, but infrequent data capture can cause you to miss the circumstances that lead to a significant event. The appropriate frequency for capturing data might vary from metric to metric. For example, CPU usage on a server might vary significantly from second to second, but high usage only becomes an issue if it's consistent over many minutes.

Information for correlating data

You can easily monitor individual and system-level performance counters, capture metrics for resources, and obtain application trace information from various log files. Some monitoring requires data correlation during the analysis and diagnostics stage in the monitoring pipeline. This data can take several forms and the analysis process must be provided with sufficient instrumentation data to map these different forms. For example, at the application framework level, a thread ID might identify a task. Within an application, the same work might be associated with the user ID for the user who completes that task.

It's unlikely to be a 1:1 map between threads and user requests, because asynchronous operations might reuse the same threads for more than one user. To complicate matters further, a single request can correlate to more than one thread as it flows through the system. If possible, associate each request with a unique activity ID that's propagated through the system as part of the request context. The technique for generating and including activity IDs in trace information depends on the technology that's used to capture the trace data.

All monitoring data should be timestamped in the same way. For consistency, record all dates and times by using Coordinated Universal Time.

> **Note**
>
> Computers that operate in different time zones and networks might not be synchronized. Don't depend on timestamps alone for correlating instrumentation data that spans multiple machines.

Information to include in the instrumentation data

Consider the following points when you decide which instrumentation data you need to collect.

Human-readable data

Ensure that information captured by trace events is both machine and human readable. Adopt well-defined schemas for this information to help implement automated processing of log data across systems, and to provide consistency for operations and engineering staff reading the logs.

Include the following environmental information in your data:

- Deployment environment
- Processing machine
- Details of the process
- Call stack

Invest in traceability and correlation

Provide sufficient context, such as an activity ID that's associated with a specific instance of a request, so that the developer or administrator can determine the source of each request.

Data context might also include information that's used to correlate an activity with the computational work performed and the resources used. *This work might cross process and machine boundaries*.

For metering, the context should directly or indirectly include a reference to the customer who caused the request. This context provides valuable information about the application state at the time that the monitoring data was captured.

Capture all relevant data

Record all requests and the locations or regions where they're made. You can use this information to help identify location-specific hotspots. This information can also be useful to determine whether to repartition an application or the data that it uses.

Record and capture the details of exceptions carefully. Critical debug information is often lost because of poor exception handling. Capture all exception details that the application throws, including any inner exceptions or other contextual information, such as the call stack, if possible.

Strive for data consistency

Consistent data can help you analyze events and correlate them with user requests. Consider using a comprehensive and configurable logging package to gather information. Logging packages can help you avoid dependence on developers to adopt your approach as they implement different parts of the system.

Gather data, such as input/output volume, number of requests, and memory, network, and CPU usage, from key performance counters. Some infrastructure services provide their own performance counters, such as:

- The number of connections to a database.

- The transaction rate.
- The number of transactions that succeed or fail.

Applications might also define their own performance counters.

Consider external dependencies

Log all external service calls. Externals calls might be made to:

- Database systems.
- Web services.
- Other system-level services that are part of the infrastructure.

Record information about the duration of each call and the success or failure of the call. If possible, capture information about all retry attempts and failures for any transient errors that occur.

Ensure telemetry system compatibility

In many cases, the instrumentation information is generated as a series of events and passed to a separate telemetry system for processing and analysis. A telemetry system is typically independent of any specific application or technology.

Telemetry systems use defined schemas to parse information. The schema specifies a contract that defines the data fields and types that the telemetry system can ingest. Generalize the schema to allow for data arriving from various platforms and devices. A common schema should include fields relevant to all instrumentation events, such as:

- Event name.
- Event time.
- IP address of the sender.
- Details required for event correlation, including:
  - User ID
  - Device ID
  - Application ID

Remember that many devices can raise events for the same application, so the schema shouldn't depend on the device type. The application should support roaming or cross-device distribution. The schema can also include relevant domain fields for a particular scenario that's common across applications, such as:

- Information about exceptions.
- Application start and end events.

- Success or failure of web service API calls.

Establish domain fields that produce the same set of events to build a set of common reports and analytics across applications. You might need to configure a schema to contain custom fields for capturing the details of application-specific events.

OpenTelemetry is a vendor-neutral collection of APIs, SDKs, and other tools. You can use OpenTelemetry to instrument applications and generate meaningful telemetry consistently across languages. OpenTelemetry is tool-agnostic, so it's compatible with many observability platforms including open-source and commercial offerings. Microsoft is adopting OpenTelemetry as the standard tool for instrumentation.

Best practices for instrumenting applications

The following list summarizes best practices for instrumenting a distributed application running in the cloud:

- Make logs easy to read and easy to parse. Use structured logging where possible.
- Be concise and descriptive in log messages.
- Identify the source of the log.
- Add timestamp information as each log record is written.
- Use the same time zone and format for all timestamps.
- Categorize logs and write messages in the appropriate place.
- Don't reveal sensitive information about the system or personal information about users. Scrub this information before it's logged, but keep any relevant details.
- Log all critical exceptions but enable the administrator to turn logging on and off as needed for fewer exceptions and warnings.
- Capture and log all retry logic information. This data is useful in monitoring the transient health of the system.
- Trace out process calls, such as requests to external web services or databases.
- Don't mix log messages with different security requirements in the same log file.
- Ensure that all logging calls are *fire-and-forget* operations that don't block the progress of business operations. Exclude auditing events from this rule because they're critical to the business.

- Ensure that logging is extensible and doesn't have any direct dependencies on a concrete target.
- Ensure that all logging is fail-safe and doesn't trigger cascading errors.
- Treat instrumentation as an ongoing iterative process and review logs regularly.

Azure facilitation

Autoinstrumentation is available for many types of Azure and on-premises applications monitored with Application Insights. The autoinstrumentation function automatically configures your application to provide rich telemetry to Application Insights and provides easy access to experiences such as the application dashboard and application map. For supported hosting platforms and development languages, see Supported environments, languages, and resource providers.

Tradeoffs

Implement profiling only when necessary because it can impose a significant overhead on the system. By using instrumentation, profiling records an event, such as a method call, every time it occurs. However, sampling records only selected events.

Profiling selections can be time-based, such as once every *n* seconds, or frequency-based, such as once every *n* requests. If events occur frequently, profiling might cause too much of a burden on the system and affect overall performance. In this case, the sampling approach is preferable. However, if the frequency of events is low, sampling might miss them. In this case, profiling might be the better approach.

Related links

- Application Insights overview
- What is autoinstrumentation for Application Insights?
- Azure Monitor logs overview
- Azure Monitor metrics overview
- Collecting ETW events for analysis Azure Monitor logs
- Recommendations for designing and creating an observability framework
- What is distributed tracing and telemetry correlation?

Community links

- OpenTelemetry

Operational Excellence checklist

Refer to the complete set of recommendations.

Operational Excellence checklist

Recommendations for designing an emergency response strategy

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Operational Excellence checklist recommendation:

OE:08	**Develop an effective emergency operations practice. Ensure that your workload emits meaningful health signals across infrastructure and code. Collect the resulting data and use it to generate actionable alerts that enact emergency responses via dashboards and queries. Clearly define human responsibilities, such as on-call rotations, incident management, emergency resource access, and running postmortems.**

This guide describes the recommendations for designing an emergency response strategy. Some issues that arise over the course of a workload's lifecycle are critical enough to warrant declaring them emergencies. You can implement tightly controlled and focused processes and procedures that your team can follow to ensure that an issue is handled in a calm, orderly manner. Emergencies naturally raise everyone's stress levels and can lead to a chaotic environment if your team isn't well-prepared. To help minimize stress and confusion, design a response strategy, share the response strategy with your organization, and perform regular emergency response training.

Key design strategies

An emergency response strategy should be an orderly, well-defined set of processes and procedures. Each process and procedure should have scripts to ensure that each step progresses your team towards quickly and safely resolving an issue. To develop an emergency response strategy, consider the following overview:

- Prerequisites
  - Develop an observability platform
  - Create an incident response plan
- Incident phases
  - Detection
  - Containment
  - Triage
- Post-incident phases
  - Root cause analysis (RCA)
  - Postmortem

- Ongoing activity
  - Emergency response drills

The following sections provide recommendations for each of these phases.

Observability

To have a robust emergency response strategy, you need to have a robust observability platform in place. Your observability platform should have the following characteristics:

- **Holistic monitoring**: Ensure that you thoroughly monitor your workload from an infrastructure and application perspective.
- **Verbose logging**: Enable verbose logging for your components to assist with investigations when you triage an issue. Structure logs so that they're easy to manage. Automatically send logs to data sinks to be prepared for analysis.
- **Useful dashboards**: Create health model-based dashboards that are tailored to each team across your organization. Different teams are responsible for different aspects of workload health.
- **Actionable alerts**: Create alerts that are useful for your workload teams. Avoid alerts that don't require action from your teams. Too many alerts of this kind can lead to people ignoring or blocking alert notifications.
- **Automatic notifications**: Ensure that appropriate teams automatically receive alerts that require action from them. For example, your tier-1 support team should get notifications for all alerts, whereas your security engineers should only get alerts for security events.

For more information, see Recommendations for designing and creating an observability framework.

Incident response plan

The foundation of an emergency response strategy is an incident response plan. Like a disaster recovery plan, clearly and thoroughly define roles, responsibilities, and procedures for an incident response plan. The plan should be a version-controlled document that's subject to regular reviews that ensure it's up to date.

Clearly define the following components in your plan.

Roles

Identify an incident response manager. This person owns the incident from initiation to remediation to the root cause analysis. An incident response manager ensures that processes are followed and the appropriate parties are informed as the response team performs their work.

Identify a postmortem leader. This individual ensures that postmortems are performed soon after the incident is resolved. They produce a report, which helps you apply the findings that come out of the incident.

Processes and procedures

Your workload team should define and understand emergency criteria. When your team determines that a case is severe, you can declare a disaster and initiate the disaster recovery plan. In less severe cases, the issue might not meet the criteria of a disaster. But you should still consider the issue an emergency, which necessitates initiating the emergency response plan. Emergencies can be issues that are internal to your workload, or they can be a result of an issue with a dependency of your workload. The support team must be able to determine whether an issue that's reported by external users meets the emergency criteria, even if they have no visibility into the underlying issue.

Precisely define communication and escalation plans. Based on the type of alert notification that they receive, ensure that your tier-1 support can easily contact the appropriate teams to escalate issues to. Ensure that they know which type of communication is appropriate for internal and external parties. In communication and escalation plans, include a list of the on-call schedule and staff.

In the overall plan, include containment and triage scripts. Teams follow these step-by-step procedures when they perform their containment and triage functions. Include a description of what defines an incident closure.

Other items to include

Document all standard tools that will be used during incidents for internal communication, like Microsoft Teams, and for tracking the activities over the course of the incident, like ticketing tools or backlog planning tools.

Document your emergency credentials, otherwise known as *break-glass accounts*. Include a step-by-step guide that describes how they should be used.

Create emergency response drill instructions, and keep a record of when drills have been performed.

Document any legal or regulatory measures necessary, for example communicating data breaches.

Incident detection

When you have a well-designed observability platform that monitors for anomalies and automatically alerts on them, you can quickly detect issues and determine their severity. If the issue is deemed an emergency, the plan can be initiated. In some cases, the support team isn't notified via the observability platform. Customers might report issues to support by using support team communication avenues. Or they might reach out to people that they regularly work with, like account executives or VPs. No matter how the support team is notified, they should always follow the same steps to validate the issue and determine the severity. Deviation from the response plan can add stress and confusion.

Containment

The first step in issue remediation is to contain the issue to protect the rest of your workload. A containment strategy depends on the type of issue, but it usually involves removing the affected component from the workload flow paths. For example, you might shut down a resource or remove it from network routing paths. System administrators, engineers, and senior developers should work together to design containment strategies. The containment should limit the blast radius of issues and maintain workload functionality in a degraded state until the issue is resolved. If an affected component needs to be accessible to perform triage, it's vital that its access to the rest of the workload is blocked. As much as possible, you should only access the component via a path that's separated from the workload and the rest of the systems.

Triage

After you successfully contain the issue, you can begin triage work. The steps that you follow during triage depend on the type of issue. The team for a certain area of workload support should create procedures for incidents that are related to their team. For example, security teams should triage security issues, and they should follow scripts that they develop. It's important that teams follow well-defined scripts as they work through their triage efforts. These scripts should be step-by-step processes that include rollback processes to undo changes that are ineffective or can cause other issues. Use off-the-shelf log aggregation and analysis tools to efficiently investigate issues that require deep analysis. After the issue is resolved, follow well-defined processes to safely bring the affected component back into the workload flow paths.

RCA reporting

The service-level agreements (SLAs) to your customers might dictate that you have to issue RCA reports within a certain time period after the incident is resolved. The incident owner should create the RCA reports. If that's not possible, another person who worked closely with the incident owner can create the RCA reports. This strategy ensures an accurate accounting of the incident. Typically, organizations have a defined RCA template with guidelines about how information is presented and what kinds of information can or can't be shared. If you need to create your own template and guidelines, ensure that they are reviewed and approved by stakeholders.

Incident postmortems

An impartial individual should lead blameless postmortems. In postmortem sessions, everyone shares their findings from an incident. Each team that was involved in the incident response should be represented by individuals that worked on the incident. Those individuals should come to the session prepared with examples of the areas that were successful and areas that can be improved. The session isn't a forum for assigning blame for the incident or issues that might have come up during the response. The postmortem leader should leave the session with a clear list of action items that focus on improvement, such as:

- **Improvements to the response plan.** Processes or procedures might need to be reevaluated and rewritten to better capture appropriate actions.
- **Improvements to the observability platform.** Thresholds might need to be reevaluated to catch the specific type of incident earlier, or new monitoring might need to be implemented to catch behavior that wasn't accounted for.
- **Improvements to the workload.** The incident might expose a vulnerability in the workload that must be addressed as a permanent remediation.

Azure facilitation

Azure Monitor is a comprehensive solution for collecting, analyzing, and responding to monitoring data from cloud and on-premises environments. It includes a robust alerting platform that you can configure for automatic notifications and other actions, like automatic scaling and other self-healing mechanisms.

Use Monitor to integrate machine learning. Automate and optimize incident triage and proactive measures. For more information, see AIOps and machine learning in Monitor.

Log Analytics is a robust analytics tool that's built into Monitor. You can use Log Analytics to run queries against aggregated logs and gain insights about your workload.

Microsoft offers Azure-related incident readiness training. For more information, see Introduction to Azure incident readiness and Incident readiness.

Tradeoffs

An overly aggressive response strategy can lead to false alarms or unnecessary escalations.

Similarly, aggressively implementing automatic scaling or other self-healing actions to respond to threshold breaches can lead to unnecessary expenditures and management burden. You might not know the exact thresholds to set for alerting and automatic actions like scaling. Perform testing in lower environments and in production to help you determine the right thresholds for your requirements.

Related links

- Recommendations for designing and creating an observability framework
- Recommendations for designing a reliable monitoring and alerting strategy
- Recommendations for self-healing and self-preservation

Operational Excellence checklist

Refer to the complete set of recommendations.

Operational Excellence checklist

Recommendations for implementing automation

Article • 01/22/2024

Applies to this Well-Architected Framework Operational Excellence checklist recommendation:

Expand table

OE:09	**Automate all tasks that don't benefit from the insight and adaptability of human intervention, are highly procedural, and have a shelf-life that yields a return on automation investment. When possible, choose off-the-shelf software for automation versus custom implementations. Treat all automation the same as workload components, and apply the Well-Architected Framework pillars to their design and implementation.**

This guide describes the recommendations for adopting automation in your workload. You can automate tasks that are repetitive and prone to human error to help your teams gain efficiency and adhere to standards. Automate tasks to make your workload streamlined and consistent. Automation enables your operations and engineering teams to be more efficient because it gives them more time to work on other improvements. Automation is a powerful tool in all aspects of workload management. Thoughtfully implement automation to empower your organization.

Key design strategies

As you develop your workload, look for opportunities to take advantage of automation in order to reduce management burden and minimize human error. Evaluate these opportunities, and consider the value that they bring to your organization. To maximize the value of your investment in automation, prioritize tasks that are straightforward, procedural, and have a long shelf life. Applying automation isn't an all-or-nothing tactic. There are workstreams that might have operations that require human intervention, like decision-making points. These workstreams can still benefit from automation to perform other tasks.

Target tasks to automate

Consider the following recommendations to ensure that you prioritize tasks that benefit the most from automation:

- **Aim for easy wins.** Focus on tasks that are highly procedural and susceptible to human error. These tasks are highly automatable. They're clearly defined, they're free from variables that add complexity, and they're performed as part of normal operations. Conversely, don't prioritize automating tasks that require writing complex scripts to account for variable phenomena, or tasks that rarely occur.

  Examples of highly automatable tasks include rebooting servers, creating accounts, and transferring logs to a data store. These tasks might occur on a schedule, as a response to an event or monitoring alert, or as needed based on external factors.

- **Look for ways to empower operators and free up your SMEs.** You might have experts in your organization that are relied upon for escalations that might be unnecessary. For example, your database administrators might routinely get requests to create new databases when you onboard new customers to your multitenant solution. If you build a self-service portal for your help desk team, you can enable them to safely create an empty database themselves. Or as an intermediary step, you can automate the requests and the steps for the SME to perform by creating scripts to run.

- **Focus on your return on investment.** High-value automation requires minimal management overhead and adds a demonstrable degree of efficiency. If you can save your operations team an hour each day by automating database entries, for example, you give them time to find other areas for improvement.

Areas to implement automation

Adopt automation throughout your entire workload lifecycle, from development to day-to-day management. Use the following list of examples to help you consider the broad areas of your workload lifecycle that can benefit from automation. You can automate:

- **Pipeline definition, execution, and management**: Use continuous integration and continuous delivery (CI/CD) tools, like Azure DevOps and other DevOps tools, to automatically define a pipeline and how it runs. These tools can help you automate CI/CD tasks or other tasks, like creating reports.

- **Deployments**: Use tools like Azure Resource Manager templates, Bicep, Terraform, and Ansible to automate your workload development and release processes. Deploy and update your infrastructure with the same automation platforms by using an infrastructure as code (IaC) approach.

- **Testing**: Many tools are available for automating your testing processes. These tools can relieve a significant burden from your quality assurance team and ensure that tests are standardized and reliable.

- **Scaling**: Use platform-provided functionality and other tools, like orchestration tools, to automatically scale your infrastructure when load increases or decreases.
- **Monitoring and alerting**: Use tooling that's available in your monitoring solution to automatically enroll newly deployed resources and configure alert-triggered actions to help hasten remediation when issues arise.
- **Self-healing**: Use alerts that are generated by your monitoring system to automate actions and recover malfunctioning components or jobs. For more information, see Recommendations for self-healing and self-preservation.
- **Configuration management**: Use orchestration and policy tooling to ensure that all of your resources run the same configuration and that compliance requirements are enforced across your workload.
- **Other administrative tasks**: Use scripts to automate repetitive tasks like updating database records or DNS records.
- **Approvals**: Enable systems to automatically make approval decisions based on predefined rules to improve efficiency for workflows that have approval gates. This method encourages the use of standardized forms and templates, which increases the efficiency of the processes. Automatic approval in high environments can be risky. Tightly focus and test your automated approvals to ensure that specific criteria are defined to grant approval.
- **New user and new employee onboarding**: You can automate many tasks associated with onboarding new application users or new employees, like database updates and credential creation.
- **Monitoring and alerting**: Take advantage of the automation functionality that your observability platform provides. Automatically enroll new devices to monitor and alert on anomalies.

Choose an appropriate automation tool

Developing your own automation in-house is time intensive and can introduce management burden to your development team. They need to maintain an in-house automation tool like they do any other in-house software. It's recommended that you use off-the-shelf tools whenever they can meet your needs. Between commercial, open source, and cloud platform provided tools, there are many options available. It's likely that you'll use a variety of tools to build the automation that you need. Rely on your in-house expertise to help guide your decisions when evaluating tools. Your team might be more familiar with certain development languages and frameworks. You can initially

focus on off-the-shelf tools that they can use without a high learning curve. Reflect on the tasks that you plan to address with automation, and invest in the tools that can specifically address those tasks. Don't procure tools that you generally prefer and then consider the tasks afterward.

Be mindful of factors that can complicate your operations when you build your automation, like version lock-in and plugin overuse. Plugins, like Jenkins or Azure DevOps plugins, are a great way to add functionality. You should adopt plugins when it benefits your automation goals. But when you use multiple plugins to perform a single task, it can make automation updates and troubleshooting difficult. Be judicious in your use of plugins. Also avoid solutions that have framework version dependencies because they're a burden to maintain over time. To help minimize the risk of these types of issues, standardize your selection of automation tools and plugins, and use source control for all automation projects.

Integrate automation into your workload

For any tool that you use to build your automation, make it easily accessible and manageable for your operators. Provide clear and easy-to-use interfaces for your workload team. You can provide access to CI/CD pipelines, APIs, and libraries. Like the workload that the automation supports, you need to manage the automation holistically. Secure automation to the same degree as other workload components. Monitor automation and subject it to the same testing protocols as other workload components.

Considerations

- Sometimes the efficiencies you gain from automation outweigh the management burden of developing your own solution if no off-the-shelf solutions fit your requirements. In these cases, be judicious in your development efforts. Narrowly focus on developing only what you need to cover gaps that you can't solve with off-the-shelf solutions, and minimize complexities like dependencies.

- Complex automation that requires a high degree of maintenance can be difficult for operations teams to manage and troubleshoot. Keep automated tasks tightly focused on only performing discrete jobs. Try to minimize dependencies on other tools or components.

- Be thoughtful about using manual processes. If you decide not to automate an operation, thoroughly document the manual process by creating a step-by-step checklist for operators. This practice reduces the chances of human error, like an

operator mistakenly running the wrong process. This documentation also helps you design automation for that process in the future.

- When you use a hybrid manual and automated approach, you need to be especially careful. If a script runs most of a process but then defers to a human for a specific part or decision, it's important that you give the person the necessary context and information to make an informed decision.

Azure facilitation

Azure offers many tools to help you automate tasks for your workload.

IaC tools: You can use Terraform, Bicep, and Azure Resource Manager for IaC deployments. Depending on your requirements and your team's familiarity with the tools, you might use one or more of these tools for your deployments and management of resources.

Azure Functions: Azure Functions is a serverless tool that you can use to automate tasks by using your preferred development language. Functions provides a comprehensive set of event-driven triggers and bindings that connect your functions to other services. You don't have to write extra code.

GitHub Actions for Azure: You can use GitHub Actions for Azure to automate CI/CD processes. GitHub Actions integrates with Azure to simplify deployments. You can create workflows that build and test every pull request in your repository, or deploy merged pull requests to production.

GitHub Actions goes beyond just DevOps and enables you to run workflows when other events occur in your repository. For example, you can run a workflow to automatically add appropriate labels when someone creates a new issue in your repository.

Azure Automation: PowerShell and Python are popular programming languages for automating operational tasks. Use these languages to perform operations like restarting services, transferring logs between data stores, and scaling infrastructure to meet demand. You can express these operations in code and run them on demand. Alone, these languages don't offer a platform for centralized management, version control, or run history. The languages also lack a native mechanism for responding to events like monitoring-driven alerts. To provide these capabilities, you need an automation platform.

Automation provides an Azure-hosted platform for hosting and running PowerShell and Python code across cloud and on-premises environments, both Azure and non-Azure. PowerShell and Python code is stored in an Automation runbook. Use Automation to:

- Trigger runbooks on demand, on a schedule, or through a webhook.
- Run history and logging.
- Integrate a secrets store.
- Integrate source control.

Azure Update Manager: Update Manager is a unified service to help manage and govern updates for virtual machines. You can monitor Windows and Linux update compliance across your workload. You can also use Update Manager to make real-time updates or schedule them within a defined maintenance window. Use Update Manager to:

- Oversee compliance on your entire fleet of machines.
- Schedule recurring updates
- Deploy critical updates

Azure Deployment Environments: Deployment Environments enables development teams to quickly create consistent app infrastructure by using project-based templates. These templates minimize setup time and maximize security, compliance, and cost efficiency. A deployment environment is a collection of Azure resources that are deployed in predefined subscriptions. Development infrastructure administrators can enforce enterprise security policies and provide a curated set of predefined IaC templates.

Development infrastructure administrators define deployment environments as catalog items. Catalog items are hosted in a GitHub or Azure DevOps repository, called a *catalog*. A catalog item consists of an IaC template and a manifest.yaml file.

You can script the creation of deployment environments and programmatically manage the environments.

Azure Logic Apps and Microsoft Power Automate: When you build custom digital process automation (DPA) to handle workload tasks like approval flows or building ChatOps integrations, consider using Logic Apps or Power Automate. You can construct workflows from built-in connectors and templates. Logic Apps and Power Automate are built on the same underlying technology and are both well-suited for trigger-based or time-based tasks.

Automatic scaling: Many Azure technologies have built-in automatic scaling capabilities. You can also program other services to automatically scale by using APIs. For more information, see Recommendations for designing a reliable scaling strategy.

Azure Monitor action groups: To automatically run self-healing operations when an alert is triggered, use Azure Monitor action groups. You can define these operations by using a runbook, an Azure function, or a webhook.

Example

For an example of using Automation in tandem with other Azure services, see Ops automation by using Azure Event Grid. This example uses Logic Apps and Event Grid to automate operational tasks.

Related links

- Automation
- Azure Update Manager
- Azure Functions
- Azure Monitor action groups
- Deployment Environments
- GitHub Actions for Azure
- Logic Apps
- Ops automation by using Event Grid
- Power Automate
- Recommendations for designing a reliability testing strategy
- Recommendations for designing a reliable scaling strategy
- Recommendations for self-healing and self-preservation

Operational Excellence checklist

Refer to the complete set of recommendations.

Operational Excellence checklist

Recommendations for enabling automation

Article • 03/28/2024

Applies to this Azure Well-Architected Framework Operation Excellence checklist recommendation:

Expand table

OE:10	**Design and implement automation upfront for operations such as lifecycle concerns, bootstrapping, and applying governance and compliance guardrails. Don't try to retrofit automation later. Choose automation features that your platform provides.**

This guide describes the recommendations for designing and implementing your workload to enable automation. Design your workload with automation in mind to ensure that routine tasks such as provisioning resources, scaling, and deployments are performed quickly and reliably. Automation simplifies maintenance tasks and allows you to update, patch, and upgrade your systems more efficiently.

Key design strategies

Workload design

You can design your workload to support automation from the ideation phase to the on-going improvement phase. First, consider how you want to apply automation in your workload to help ensure that you're putting the necessary pieces in place. Think about your workload in terms of the Well-Architected Framework pillars to help plan for the types of automation you'll use. You can automate many functions of security, reliability, performance, operations, and cost control.

Design with automation in mind to minimize refactoring after your workload is running. Consider your workload requirements when deciding which automation tools to use. There might be off-the-shelf automation tools that your team is already familiar with. Adopting those tools can make the path towards automating your workload easier but be mindful of their limitations and compatibility with your cloud platform. For example, some automation tools might integrate well with Azure CLI tooling, while others might require REST interfaces. Always investigate the tools that your cloud platform provides

to ensure they're compatible and provide the functionality you require. Examples of ways that you can proactively plan for automation include:

- *Deployment*: Automate your application and infrastructure deployments to ensure a predictable standard. Plan for automated deployment by developing deployment standards, training your team on the tools that you'll use, and implementing the necessary infrastructure.
- *Validation*: Automatically validate compliance requirements against your workload using orchestration or policy tools. Identify the appropriate validation tool for your workload and plan to implement the required systems, for example, orchestration servers.
- *Automatic scaling*: Use automatic scaling throughout your infrastructure to help you achieve your reliability and performance requirements. You should allocate IP address space and subnets in your workload ahead of time to account for scaling operations, in addition to planning for redundancy and natural growth.

>  **Tradeoff**: When designing your workload to enable automation, consider the degree of control that you want to maintain versus the efficiency you can gain through automation. In some cases, your workload might not be mature enough to automate some functions or you might need a level of flexibility that automation doesn't provide.

> Also consider the skill set of your team when designing your workload. If a high degree of automation requires tools that your team isn’t equipped to support, then you might need to use a less comprehensive design as an intermediate step.

Continuous workload improvements

After your workload is running in the cloud, it's important to prioritize continuous improvement. Observe your workload in action, analyze usage patterns, and review customer behavior related to your workload to identify areas where you can improve automation. Look for ways to enhance existing automation or introduce new automation to improve your customer experience. For example, you might have automated scaling enabled, but the workload increase is short-lived. You can integrate scale-in automation to decrease CPU usage when the load drops below the threshold.

The following sections of this guide offer recommendations on specific areas of automation that can help you in your workload design and implementation.

Bootstrapping

Bootstrapping refers to the configuration updates to a resource that must be made after it's provisioned, but before it's available as part of the workload pool. Bootstrapping is often associated with virtual machines (VMs), but many other resources must be set up as part of the deployment process including platform as a service (PaaS) technologies and container hosting technologies like Azure Kubernetes Service (AKS).

Your cloud platform might provide bootstrapping solutions for you, which you should use where possible. For example, you can use VM extensions in Azure to make predefined configuration changes during the deployment process and customize your configuration changes by injecting PowerShell scripts.

Authentication and authorization

Take automation into account when designing your authentication and authorization strategy. It's important to maintain the highest level of security in production workloads, but this can affect automation. For example, the use of biometric or multifactor authentication adds complexity that must be accounted for in your automation design. Use nonhuman, secure accounts for automated authentication, such as managed identities, workload identities, or certificates. Ensure that you have included secret and key management in your automation for increased authentication security.

Design variability into your workload

Avoid unnecessarily deploying new infrastructure when small changes are made by building flexibility into your artifacts. For example, rather than redeploying your infrastructure when a feature flag changes, you can use parameters that are set to update components like app configurations. Be sure to clearly define and document how variability is used to avoid overuse and configuration drift.

Build a control plane

A control plane is the back-end system or suite of tools that you use to manage the application and its dependencies through a unified interface. Build your control plane like a REST interface, CLI, or webhook to support automation by external tools.

Expose maintenance operations through the control plane that allow you to coordinate workload components, for example orderly backup and restore, bootstrapping, configuration, import/export, and batching operations. Be careful to choose the right level of granularity when deciding the operations to expose through the control plane.

Monitor and log

Develop a monitoring strategy to capture metrics that drive the type of automation you require. Use structured logging and custom metrics to provide the information required by automation in a format that's easy to recognize with automation tools. The metrics that you capture should be paired with thresholds defined in the monitoring system that trigger alerts and automated actions, like notifications or self-healing mechanisms, when appropriate. For more information, see Recommendations for self-healing and self-preservation.

User lifecycle

Design your application and infrastructure to allow for automated user onboarding and offboarding, for individuals or multitenant customers. Plan for automated database updates via scripts, infrastructure provisioning and deprovisioning, and credential and secret management.

Orchestration and policy use

As part of your continuous workload management, you can automate Desired State Configuration (DSC) in your resources to help ensure that they meet compliance and business requirements. DSC automation helps ensure that configuration drift is caught and remediated quickly. You can automate DSC using orchestration tools or policy management tools. Think of orchestration tools, like Azure DevOps services or Jenkins, as push-based mechanisms. Orchestration tools allow configuration updates to be pushed out through a workflow event, like a manual or automated deployment. These updates are run as part of a task sequence defined in your deployment script. Policy management tools use pull-based mechanisms, meaning that a system runs at the foundational level of your workload that periodically polls the workload to check its state against your defined DSC. If the poll identifies a misalignment or configuration drift, the tool takes corrective action. Consider the following factors when deciding between orchestration and policy management tools:

- Orchestration tools don't have built-in capabilities to proactively poll your workload for configuration drift. Orchestration tools should be integrated into your continuous integration and continuous delivery (CI/CD) pipeline to maintain a standard for infrastructure as code (IaC) deployment and management. An advantage of using orchestration tools is that resources are always fully configured when deployed.

- Policy management tools allow you to define policies that affect one or more groups of resources. These policies are enforced when the resource checks in with the policy management system. An advantage of using policy management is that these systems aren't code driven, so they might be easier for operators on your team to adopt.

When deciding between orchestration or policy tools, consider whether the configuration updates you're planning to make on new resources must be made at the time of deployment. Also consider if defining updates in code fits your operational practices and how many resource types you plan to deploy. If there are many different configurations across resource types, policy tools might be an easier way to manage updates.

Azure facilitation

Policy management

Azure Policy: Using Azure Policy, you can enforce standards and assess compliance at scale. Azure Policy provides an aggregated view to evaluate the overall state of the workload environment in the compliance dashboard. Or you can use Azure Policy to evaluate each resource and policy on a granular level. You can also use Azure Policy to remediate new resources automatically or remediate existing resources in bulk.

>  **Tradeoff**: Offloading automation from your CI/CD pipeline to platform tools or services, like Azure Policy, can simplify your pipeline, but has drawbacks like the additional management burden of using multiple systems. For example, execution failures in a platform service will not be caught in your pipeline logs and will have to fed into your observability platform intelligently so the appropriate parties are notified.

Bootstrap automation

Azure Virtual Machines extensions: Virtual Machines extensions are small packages that run post-deployment configuration and automation on VMs. Several extensions are available for different configuration tasks, such as running scripts, configuring anti-malware solutions, and configuring logging solutions. Install and run these extensions on VMs by using an Azure Resource Manager template, Azure CLI, Azure PowerShell module, or the Azure portal. Each VM has a VM agent installed that manages the lifecycle of the extension.

Typically, VM extensions use a custom script extension to install software, run commands, and perform configurations on a VM or Azure Virtual Machine Scale Sets. You can set these extensions up to run as part of IaC deployments so that they run on new VMs using the Azure VM Agent. Extensions can also be run outside of an Azure deployment by using the Azure CLI, PowerShell module, or the Azure portal.

Cloud-init: Cloud-init is an industry tool for configuring Linux VMs on first boot. Much like Azure custom script extensions, cloud-init lets you install packages and run commands on Linux VMs. You can use cloud-init for software installation, system configuration, and content staging. Azure includes many cloud-init-enabled VM images across well-known Linux distributions. For a full list, see cloud-init support for VMs in Azure.

Azure deployment script resource: When you deploy using Azure, you might need to run arbitrary code for bootstrapping the management of user accounts, Kubernetes pods, or querying data from a non-Azure system. Because none of these operations are accessible through the Azure control plane, a separate mechanism is required. For more information, see Microsoft.Resources deploymentScripts. Like any other Azure resource, the deployment script resource:

- Can be used in an Azure Resource Manager template.
- Contains Azure Resource Manager template dependencies in other resources.
- Consumes input and produces output.
- Uses a user-assigned managed identity for authentication.

When deployed, the deployment script runs PowerShell or Azure CLI commands and scripts. Script runs and logging can be observed in the Azure portal or with the Azure CLI and PowerShell module. You can customize the variables for the run environment, timeout options, and resource management after a script failure.

Bootstrap AKS clusters with GitOps: You can bootstrap a newly provisioned AKS cluster using GitOps and the Flux v2 cluster extension by declaring your configuration settings in GitHub repositories. Because AKS cluster files are stored in a GitHub repository, they're versioned, and changes between versions are easily tracked. Kubernetes controllers run in the clusters and continually reconcile the cluster state with the desired state declared in the Git repository by pulling the files from the repository. For more information, see AKS baseline reference architecture.

Configuration management

Azure Automation State Configuration is a DSC management tool managed by the Azure Policy guest configuration feature that you can use to write, manage, and compile PowerShell DSC configurations for nodes in any cloud or on-premises datacenter. You can also use this tool to import DSC resources and assign configurations to target nodes.

Azure App Configuration is a service that you can use to centrally manage your application settings and feature flags. It works with Azure Key Vault so you can securely manage a wide variety of application configurations across your environment.

Change tracking and inventory

Change tracking and inventory using Azure Monitoring Agent tracks OS configuration drift in virtual machines. This automates detection of drift, the inventory running services, and installed packages on the virtual machines in your workload. Items that are tracked by change tracking and inventory include:

- Installed Windows and Linux software
- Key Windows and Linux files
- Windows registry keys
- Windows services and Linux daemons

Related links

- AKS baseline reference architecture
- Azure App Configuration
- Azure Automanage State Configuration
- Azure Policy
- Cloud-init support for VMs in Azure
- GitOps Flux v2 configurations with AKS and Azure Arc-enabled Kubernetes
- Microsoft.Resources deploymentScripts
- Recommendations for self-healing and self-preservation

Operational Excellence checklist

Refer to the complete set of recommendations.

Operational Excellence checklist

Feedback

Was this page helpful? Yes No

Recommendations for safe deployment practices

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Operational Excellence checklist recommendation:

> **OE:11** **Clearly define your workload's safe deployment practices. Emphasize the ideals of small, incremental, quality-gated release methods. Use modern deployment patterns and progressive exposure techniques to control risk. Account for routine deployments and emergency, or hotfix, deployments.**

This guide describes the recommendations for using safe deployment practices (SDP). Safe deployment processes and procedures define how to safely make and deploy changes to your workload. Implementing SDP requires you to think about deployments through the lens of managing risk. You can minimize the risk of human error in your deployments and limit the effects of problematic deployments on your users by implementing SDP.

Key design strategies

There are four important guidelines to keep in mind when implementing safe deployment practices:

- **Safety and consistency**: All changes to the production workload are inherently risky and must be made with a focus on safety and consistency.
- **Progressive exposure**: You can minimize the potential blast radius of deployment-caused issues by adopting a progressive exposure deployment model.
- **Health models**: Deployments must pass health checks before each phase of progressive exposure can begin.
- **Issue detection**: When issues are detected, the deployment should be immediately halted and recovery initiated.

The following sections provide detailed recommendations on each of these points.

Safety and consistency

Whether you're deploying an update to your application code, infrastructure as code (IaC), feature flag, or a configuration update, you're introducing risk to the workload. There are no *low-risk* deployments to production. Every deployment must follow a standard pattern and should be automated to enforce consistency and minimize the risk of human error. It's critical that your workload supply chain and deployment pipelines are reliable, secure, and have clearly defined deployment standards. Treat every deployment as a possible risk and subject every deployment to the same level of risk management. Despite the risks, you should continue to deploy regular changes to your workload. Failing to deploy regular updates introduces other risks, like security vulnerabilities that must be addressed through deployments. For more information, see Recommendations for designing a workload development supply chain.

Frequent small deployments are preferable to infrequent large deployments. Small changes are easier to resolve when issues arise and frequent deployments help your team build confidence in the deployment process. It's also important that you learn from production by reviewing your workload processes when you encounter an anomaly during deployment. You might find weaknesses in the design of your infrastructure or rollout. When issues occur during deployments, ensure that *blameless* postmortems are part of your SDP process to capture lessons about the incident.

Progressive exposure deployment

When deployment issues occur, the goal is to catch them as early as possible to minimize the effect on end users. Implement a gradual rollout deployment model, also known as a *progressive exposure model*, to accomplish this goal. Canary deployments are a common example of progressive exposure. In this deployment model, a small group of internal or external users receive the new feature first. After the first group runs the new version without issue, the feature is deployed to successively larger groups until the entire user population is running the new version. Feature flags are typically used to enable the new version for the target users in canary deployments.

Another common deployment model is a blue-green approach. In this model, two identical sets, or pools, of workload infrastructure are deployed. Both pools are able to handle a full production load. The first (blue) pool runs the current version of the deployment where all users land. The second (green) pool is updated with the new feature and internally tested. After internal testing, a subset of the production traffic is routed from the blue pool to the green pool. Like canary deployments, the rollout is progressive as you shift more of the traffic over to the green pool in successively larger rollout waves. After you finish the rollout, the update pool becomes the blue pool and the green pool is ready for the next deployment. The two pools are logically separated from each other to protect from malfunctions. You can deploy a variation of the blue-

green model in a workload that uses the Deployment Stamps design pattern by deploying on one stamp at a time.

In both of these models, the time between each phase of the rollout should be long enough to enable you to monitor the health metrics of the workload. You should provide ample *bake time*, time between rollout groups, to help ensure that users from different regions or users who perform different tasks have time to use the workload in their normal capacity. Bake times should be measured in hours and days rather than minutes. Bake times should also increase for each rollout group so that you can account for different time zones and usage patterns over the course of the day.

Health models

Develop a robust health model as part of your observability platform and reliability strategies. Your health model should provide in-depth visibility into the components and overall health of the workload. During a rollout, if you receive an alert about a health change relating to an end user, the rollout should immediately halt and an investigation into the cause of the alert should be performed to help determine the next course of action. If there are no issues reported by end users and all health indicators stay green throughout the bake time, the rollout should continue. Be sure to include usage metrics in your health model to help ensure that a lack of user-reported issues and negative health signals aren't hiding an issue. For more information, see Building a health model.

Issue detection

When your deployment causes an issue in one of the rollout groups, the rollout must stop immediately. An investigation into the cause of the issue and the severity of the effects must be performed as soon as the alert is received. Recovery from the issue can include:

- **Rolling back** the deployment by undoing the changes made in the deployment and reverting back to the last known working configuration.
- **Rolling forward** the deployment by addressing the issue in the midst of the rollout. You can address issues mid-rollout by applying a hotfix or otherwise minimizing the issue.
- **Deploying new infrastructure** by using the last known working configuration.

Rolling back changes, especially database, schema, or other stateful component changes, can be complex. Your SDP guidelines should provide clear instructions on how to deal with data changes according to the data estate design for your workload.

Similarly, rolling forward must be handled carefully to ensure that SDP isn't neglected and that the hotfix or other minimizing efforts are performed safely.

General SDP recommendations

- Implement versioning across your build artifacts to help ensure that you can roll back and roll forward when necessary.
- Use a release flow or trunk-based branching structure, which enforces tightly synced collaboration across the development team, instead of a Gitflow or environment-based branching structure.
- Automate as much of your SDP as possible. For detailed guidance on automating IaC and application continuous integration and continuous delivery (CI/CD) processes, see Recommendations for implementing automation.
- Use CI practices to regularly integrate code changes into repositories. CI practices can help you identify integration conflicts and reduce the likelihood of large, risky merges. For more information, see the Continuous integration guide.
- Use feature flags to selectively enable or disable new features or changes in production. Feature flags can help you control the exposure of new code and quickly roll back deployment if issues arise.
- Deploy changes to staging environments that mirror your production environment. Practice environments allow you to test changes in a controlled setting before deploying to the live environment.
- Establish predeployment checks, including code review, security scans, and compliance checks, to help ensure that changes are safe to deploy.
- Implement circuit breakers to automatically halt traffic to a service that's experiencing issues. Doing so can help to prevent further degradation of the system.

Emergency SDP protocols

Establish prescriptive protocols that define how your SDP can be adjusted for a hotfix or for emergency issues like a security breach or vulnerability exposure. For example, your emergency SDP protocols might include:

- Promotion and approval stage acceleration.
- Smoke testing and integration testing acceleration.

- Bake time reduction.

In some cases, the emergency might limit quality and testing gates, but gates should still be run as quickly as possible as an out-of-band exercise. Make sure that you define who can approve SDP acceleration in an emergency and the criteria that must be met for acceleration to be approved. Align your emergency SDP protocols with your emergency response plan to help ensure that all emergencies are handled according to the same protocols.

Azure facilitation

- Azure Pipelines and GitHub Actions support multi-stage deployments by using approval gates, which can help you design your progressive exposure rollout for deployments.
- Use Azure App Service staging slots to easily swap between versions of code. Staging slots are helpful for testing in staging environments and can be used for blue-green deployments.
- Store and manage your web app feature flags in Azure App Configuration. By using this service, you can create, change, and deploy features in a unified management plane.
- Deploy workload applications in your virtual machine by using VM Applications.
- Use Azure load balancers to implement deployment strategies and expose the health of your workload applications by using native resources.
- Use Application Health extension to report on application health from inside a Virtual Machine Scale Set instance. The extension probes on a local application endpoint and updates the health status based on TCP/HTTP(S) responses received from the application.
- Use Azure Logic Apps to create a new version of the application whenever an update is made to it. Azure maintains a history of application versions and can revert or promote to any previous version.
- Many Azure Database services provide point-in-time restore functionality that can help you roll back. Services that support point-in-time restore include:
  - Azure SQL Database
  - Azure SQL Managed Instance
  - Azure Cosmos DB
  - Azure Database for MySQL

    - Azure Database for PostgreSQL

Tradeoffs

Building and maintaining safe deployment practices is complex. Your success in fully implementing robust standards depends on the maturity of your practices across many areas of software development. Use of automation, IaC-only for infrastructure changes, consistency in branching strategies, use of feature flags, and many other practices can help to ensure safe deployment. Use this guide to optimize your workload and inform your plans for improvement as your practices evolve.

There are tradeoffs for each deployment model discussed in this guide. For example, during canary deployments, two versions of an application are supported on the same infrastructure, which increases the management burden on the workload and support teams. Conversely, during blue-green deployment, two sets of production infrastructure are run at the same time, which might result in extra cost and increased management workload.

Example

See the blue-green deployment of Azure Kubernetes Service (AKS) clusters architecture guide for an example of how to use this deployment model.

Related links

- Application Health extension
- Azure App Configuration
- Azure App Service staging slots
- Azure Cosmos DB
- Azure Database for MySQL
- Azure Database for PostgreSQL
- Azure load balancers
- Azure Logic Apps
- Azure Pipelines
- Azure SQL Database
- Azure SQL Managed Instance
- Building a health model
- Continuous integration guide
- Deployment Stamps
- Performance considerations for your deployment infrastructure

- Release engineering: Application development
- Release engineering: Continuous integration
- Release engineering: Deployment
- Release engineering: Rollback
- Testing your application and Azure environment
- VM Applications

Community links

- Advancing safe deployment practices
- GitHub Actions

Operational Excellence checklist

Refer to the complete set of recommendations.

Operational Excellence checklist

Recommendations for designing a deployment failure mitigation strategy

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Operational Excellence checklist recommendation:

> **OE:12** **Implement a deployment failure mitigation strategy that addresses unexpected mid-rollout issues with rapid recovery. Combine multiple approaches, such as rollback, feature disablement, or using your deployment pattern's native capabilities.**

This guide describes the recommendations for designing a standardized strategy to effectively handle deployment failures. Like other workload issues, deployment failures are an inevitable part of a workload lifecycle. With this mindset, you can be proactive by having a well-designed, intentional strategy for dealing with deployment failures. Such a strategy enables your workload team to efficiently mitigate failures with as little impact as possible on your end users.

The absence of such a plan can lead to chaotic and potentially detrimental responses to issues, which can significantly affect team and organizational cohesion, customer trust, and finances.

Key design strategies

Occasionally, despite the maturity of your development practices, deployment issues occur. Using safe deployment practices and operating a robust workload supply chain can help you minimize the frequency of failed deployments. But you also need to design a standardized strategy to handle failed deployments when they happen.

When you use a progressive exposure deployment model as your standard practice:

- You have the right foundation for minimizing the effects on your customers or internal users when deployments fail.
- You can mitigate issues efficiently.

A deployment failure mitigation strategy is composed of five broad phases:

1. Detection: To respond to a failed deployment, you must first detect the failure. Detection can take several forms, like failed smoke tests, user reported issues, or alerts that your monitoring platform generates.

2. Decision: You must decide what the best mitigation strategy is for the specific failure type.

3. Mitigation: You perform the identified mitigation action. The mitigation can take the form of a fallback, rollback, roll forward, or the use of a runtime configuration to bypass the offending function.

4. Communication: Stakeholders and affected end users must be made aware of the status as you detect and work through the issue as required by your emergency response plan.

5. Postmortem: Blameless postmortems provide opportunities for the workload team to identify areas for improvement and create plans to apply learnings.

The following sections provide detailed recommendations for these phases. These sections assume that you detect an issue after you deploy your changes to one or more groups of users or systems but before you update all groups in your rollout plan.

Detection

To quickly identify issues with deployments, you need robust testing and observability practices as they relate to deployments. To help detect anomalies quickly, you can complement your workload monitoring and alerting by taking the following steps:

- Use an application performance management tool.
- Enable logging through instrumentation.

Smoke testing and other quality testing should happen at each phase of your rollout. Successful tests in one deployment group shouldn't influence decisions to test in subsequent groups.

You can implement telemetry that correlates user issues with a deployment phase. Then you can quickly identify which rollout group a particular user is associated with. This approach is especially important for progressive exposure deployments, because you might have multiple configurations running across your user base at any given point in the deployment.

You should be ready to respond to user-reported issues immediately. Whenever practical, deploy your rollout phase during working hours, when you have a full support team available. Ensure support staff is trained on how to escalate deployment issues for proper routing. Escalations should align with your workload emergency response plan.

Decision

Deciding on an appropriate mitigation strategy for a given deployment issue involves considering many factors, including:

- The type of progressive exposure model that you use. For example, you might use a blue-green model or a canary model.

  If you use a blue-green model, falling back is more practical than rolling back. You can easily shift traffic back to the stack that runs the configuration that's not updated. You can then fix the issue in the problematic environment and try your deployment again at an appropriate time.

- The methods that you have at your disposal for bypassing the issue. Examples include the use of feature flags, environmental variables, or any other type of runtime configuration property that you can toggle on and off.

  Sometimes you can easily bypass an issue by turning off a feature flag or toggling a setting. In this case, you might be able to:
  - Proceed with the rollout.
  - Avoid falling back.
  - Roll back while you fix the offending code.

- The level of effort that's required to implement a fix in the code.

  If the level of effort to fix the code is low, rolling forward by implementing a hot fix is the right choice for certain scenarios.

- The level of criticality for the affected workload.

  Business-critical workloads should always be deployed in a side-by-side manner, like in a blue-green model, to achieve zero-downtime deployments. As a result, falling back is the preferable mitigation strategy for these types of workloads.

- The type of infrastructure that the workload uses—mutable or immutable.

  If your workload is designed to use mutable infrastructure, rolling forward can make sense, because it involves updating infrastructure in place. Conversely, rolling back or falling back can make sense when you use immutable infrastructure.

No matter which choices you make, you should include appropriate approvals in your decision-making process and codify them in your decision tree.

Mitigation

- **Rollback**: In a rollback scenario, you revert updated systems to the last-known-good configuration state.

It's important for the entire workload team to agree about the meaning of *last known good*. This expression refers to the last state of the workload that was healthy before the deployment began, which isn't necessarily the prior application version.

Rolling back can be complex, especially as it relates to data changes. Schema changes can make rolling back risky. Implementing them safely can require considerable planning. As a general recommendation, schema updates should be additive. They shouldn't be replacement changes—records shouldn't be replaced with new records. Instead, older records should be deprecated and should coexist with new records until it's safe to remove the deprecated records.

- **Fallback**: In a fallback scenario, the updated systems are removed from the production traffic routing. All traffic is directed to the stack that isn't updated. This low-risk strategy provides a way for you to address the issues in your code without introducing further disruptions.

  With canary deployments, falling back might not be straightforward or even possible, depending on your infrastructure and app design. If you need to perform scaling to handle load on systems that aren't updated, perform that scaling before you fall back.

- **Bypass the offending function**: If you can bypass the issue by using feature flags or another type of runtime configuration property, you might decide that continuing with the rollout is the right strategy for a given issue.

  You should clearly understand the tradeoff of bypassing the function, and you should be able to communicate that tradeoff to stakeholders. Stakeholders should approve the go-forward plan. Stakeholders need to determine the length of time that's tolerable for operating in a degraded state. They also need to weigh that determination against your estimate of the time that's needed to fix the offending code and deploy it.

- **Emergency deployment (hot fix)**: If you can address the issue mid-rollout, a hot fix might be the most practical mitigation strategy.

  Like any other code, hot fixes need to go through your safe deployment practices. But with a hot fix, the timeline is considerably accelerated. You need to use a code promotion strategy throughout your environments. You also need to check hot fix code at all quality gates. But you might need to dramatically shorten bake times, and you might need to modify tests to accelerate them. Ensure that you can run full tests on the updated code as soon as possible after deployment. Automating

quality assurance testing to a high degree helps make testing efficient in these scenarios.

Communication

It's important to have clearly defined communication responsibilities to help minimize chaos during incidents. These responsibilities should establish how the workload team engages with support teams, stakeholders, and emergency response team personnel, like the emergency response manager.

Standardize the cadence that the workload team follows for providing status updates. Ensure that stakeholders are aware of this standard so that they know when to expect updates.

If the workload team needs to communicate directly with end users, clarify the type of information and level of detail that are appropriate for sharing with users. Also communicate to the workload team any other requirements that apply to these cases.

Postmortem

Postmortems should follow all failed deployments, without exception. Every failed deployment is an opportunity for learning. Postmortems can help you identify weaknesses in your deployment and development processes. You also might identify misconfigurations in your infrastructure, among many other possibilities.

Postmortems should always be blameless so that individuals who are involved in the incident feel safe when they share their perspectives on what can be improved. Postmortem leaders should follow up with plans for implementing the improvements that have been identified and adding these plans to the workload backlog.

Considerations and general recommendations

Test deployments thoroughly when you deploy to lower development environments. This practice helps you detect bugs and misconfigurations before they get to production.

Ensure that your deployment pipeline can accept distinct versions as parameters so that you can easily deploy last-known-good configurations.

Maintain consistency with the management and data planes as you roll back or roll forward. Keys, secrets, database state data, and configurations that are specific to resources and policies all need to be in scope and accounted for. For example, pay

attention to the design of your infrastructure scaling in your last-known-good deployment. Determine whether you need to adjust that configuration when you redeploy your code.

Prefer small, frequent changes over infrequent, large changes so that the delta between your new and last-known-good deployments is small.

Perform a failure mode analysis (FMA) on your continuous integration and continuous delivery (CI/CD) pipelines to help identify issues that can complicate mitigations. Like your workload as a whole, your pipelines can be analyzed to identify areas that might be problematic when you attempt a given mitigation type.

Use automated rollback functionality judiciously:

- Some CI/CD tools like Azure DevOps have automatic rollback functionality that's built in. Consider using this functionality if it provides tangible value to you.
- You should adopt automatic rollback functionality only after you test your pipeline thoroughly and regularly. Ensure that your pipeline is mature enough to introduce extra issues if an automatic rollback is triggered.
- You need to trust that the automation deploys only necessary changes and that it's triggered only when necessary. Design your triggers carefully to meet these requirements.

Use platform-provided capabilities during rollbacks. For example, backups and point-in-time restores can help with storage and data rollbacks. Or if you use virtual machines (VMs) to host your application, it can be helpful to restore your environment to a checkpoint that's immediately before an incident.

Test your entire deployment failure mitigation strategy frequently. Like emergency response plans and disaster recovery plans, your deployment failure plan is only successful if your team is trained on it and practices it regularly. Chaos engineering and fault injection testing can be effective techniques for testing your deployment mitigation strategy.

Azure facilitation

- Azure Pipelines provides build and release services to support CI/CD of your applications.

- Azure Test Plans is a browser-based test management solution that's easy to use. This solution offers capabilities that are required for planned manual testing, user acceptance testing, and exploratory testing. Azure Test Plans also provides a way for you to gather feedback from stakeholders.

- Azure Monitor is a comprehensive monitoring solution for collecting, analyzing, and responding to monitoring data from your cloud and on-premises environments. Monitor includes a robust alerting platform. You can configure that system for automatic notifications and other actions, like autoscaling and other self-healing mechanisms.

- Application Insights is an extension of Monitor that provides application performance monitoring (APM) features.

- Azure Logic Apps is a cloud-based platform for running automated workflows that integrate apps, data, services, and systems. You can use Logic Apps to create a new version of your application whenever an update is made to it. Azure maintains a history of the versions and can revert or promote any previous version.

- Many Azure database services provide point-in-time restore functionality that can help you when you need to roll back:

  - Azure SQL Database
  - Azure SQL Managed Instance
  - Azure Cosmos DB
  - Azure Database for MySQL
  - Azure Database for PostgreSQL

- Azure Chaos Studio Preview is a managed service that uses chaos engineering to help you measure, understand, and improve your cloud application and service resilience.

Tradeoffs

Support team members need to be able to perform their normal duties and also support emergencies. You might need to increase head count to help ensure that the support team is properly staffed and able to carry out all required duties.

Also consider these other potential tradeoffs that are associated with mitigation strategies:

- Being able to fall back typically means that you need sufficient infrastructure capacity to handle two versions of your workload configuration at the same time. Your workload teams also need to be able to support two versions in production at the same time.

- Being able to roll back effectively might involve refactoring elements of your workload. For example, you might need to decouple functions or change your data model.

Related links

- Progressive experimentation with feature flags
- Recommendations for designing and creating an observability framework
- Recommendations for designing an emergency response strategy
- Recommendations for designing a reliability testing strategy
- Recommendations for designing a workload development supply chain
- Recommendations for performing failure mode analysis
- Recommendations for safe deployment practices

Operational Excellence checklist

Refer to the complete set of recommendations.

Operational Excellence checklist

Performance efficiency quick links

Apply performance efficiency guidance to your architecture to efficiently meet workload demands.

Learn key points

QUICKSTART

Design principles

Checklist

Tradeoffs

Performance efficiency patterns

Azure Well-Architected Review assessment

TRAINING

Performance efficiency

VIDEO

Performance efficiency: Fast & furious

Design principles

CONCEPT

Negotiate realistic performance targets

Design to meet capacity requirements

Achieve and sustain performance

Improve efficiency through optimization

Achieve performance targets

HOW-TO GUIDE

Set performance targets

Select the right services

Conduct capacity planning

Collect performance data

Improve efficiency

HOW-TO GUIDE

Scaling and partitioning

Code and infrastructure

Data

Critical flows

Operational tasks

Continuous optimization

Implement testing and incident response

HOW-TO GUIDE

Run performance testing

Address performance incidents

Explore related resources

REFERENCE

Azure Advisor: Performance recommendations

Application Insights

Azure Load Testing

Performance Efficiency design principles

Article • 11/14/2023

Performance efficiency is the ability of your workload to adjust to changes in demands. A workload must be able to **handle an increase in load without compromising user experience**. Conversely, **when there's a decrease in load, the workload must conserve its resources**. Capacity, which indicates resource availability (CPU and memory), is a significant factor.

The workload design shouldn't just rely on pre-provisioned capacity, which guarantees performance up to a certain limit. If that limit is exceeded, the workload might have performance issues or even experience outages. When load is under that limit, resources continue to run unnecessarily, incurring costs.

You need a comprehensive strategy to sustain performance targets over time. Performance considerations shouldn't be an afterthought in the design process, only to be addressed when issues arise in production. Instead, **adopt a mindset where performance is a key consideration from the early stages of design**. Initially, build the system without any specific performance targets. But from there, test and measure performance at each stage of development to ensure progress and effectiveness. Continual optimization of these targets throughout the process and incorporating lessons learned from production can significantly mitigate potential issues in advance.

These design principles can help **build your strategy for managing capacity** of resources to sufficiently meet your business requirements for expected usage. Also, reduce wastage during off-peak hours. After you've decided on a strategy, solidify your design by using the Performance Efficiency checklist.

Performance efficiency is about effective use of workload resources. Without a good strategy, you might not be able to anticipate and meet user demands. You might have to resort to an approach of long-term forecasting and pre-provisioned capacity, which doesn't let you take full advantage of your cloud platform.

Negotiate realistic performance targets



The intended user experience is defined, and there's a strategy to develop a benchmark and measure targets against the pre-established business requirements.

From a performance perspective, it's ideal to have well-defined performance targets to start your design process. To set those targets, you need to have a good understanding

of the business requirements and the anticipated quality of service that the workload is expected to deliver. Define the expectations in collaboration with the business stakeholders. Instead of only focusing on technical metrics, determine the acceptable effects on the user experience for the key flows.

There's a circular dependency. You can't measure what you haven't defined, and you can't define without measurement. So, it's also important to **measure the workload performance until you achieve a satisfactory definition of acceptable threshold** with collective agreement.

There's a strong correlation between performance and reliability targets, which help determine the quality of service in terms of performance, availability, and resilience. Without a clear definition, it's challenging to measure, alert for, and test performance. After you establish the targets and identify actual numbers through testing over time, you can implement automation for continuous testing against these targets.

Adhere to best practices in defining targets at the macro level, even if they're approximate or within a range.

Approach	Benefits
Prepare for effective negotiation by understanding technical concepts, exploring design possibilities with the available infrastructure, and using results from concrete experimentation, if available. **Use historical data** to get visibility into usage patterns and bottlenecks. Bring insight from external factors, such as **input from market analysis, experts, and industry standards**.	You can make **informed decisions** based on practical insights. The performance targets are focused on user experience that's based on what's feasible, industry best practices, and current market trends.
Collaborate with the business owners to understand user promises, in terms of quality and regulatory compliance, if applicable. **Maintain a broad perspective** and avoid diving into granular details at this stage. Be explicit about **what represents acceptable performance**, based on the investments. Understand the business context and **anticipated growth**.	You'll **avoid making assumptions** that might not align with the business goals. It also drives clarity and motivation within the workload team. Having a business context on functional and nonfunctional requirements might uncover design changes in other Azure Well-Architected pillars and **help you make informed tradeoffs**. Defining parameters early on helps avoid costs associated with potential solution

Approach	Benefits
	redesigns later. It enables you to ensure that **performance targets cover future projections**, so you can align current efforts with long-term goals.
Identify the workload flows and prioritize the flows in the architectural diagram. **Define each flow's performance tolerance** as a range from aspirational to unacceptable performance. **Evaluate the entry and exit points for each flow**, considering the path's criticality, usage frequency, and architectural intensity.	By prioritizing flows, you can **focus your resources on critical areas** that have the most effect on user and business outcomes. By breaking down the system into its parts and dependencies, you understand each component's function and influence on performance. You also become aware of potential issues. It helps establish a performance baseline and drive optimization.
Start building a performance model Consider whether usage patterns show seasonal or daily variations. Factor in the cost, operations, and criticality to the business. **Use industry standards to quantify metrics** and aggregation methods, such as using percentiles. **Evaluate the demand and supply expectations** and limitations that business constraints impose. Incorporate growth prospects.	A performance model provides **insight into optimal use of resources** and helps with strategic planning. Industry standards help with benchmarking. Future proofing ensures that the performance goals remain relevant and can adapt to changes.

Design to meet capacity requirements

 **Provide enough supply to address anticipated demand.**

It's important to proactively measure performance. Measuring performance involves **measuring baselines** and having a preliminary understanding of which components of the system are likely to pose challenges. You can achieve it without conducting a full performance test or through granular optimization. By taking these initial steps, you establish a foundation for effective performance management early in the development lifecycle.

Examine the system as a whole, rather than focusing on individual components. Avoid fine-tuning at this stage. Making granular performance improvements results in tradeoffs in other areas. As you progress through the lifecycle and begin user acceptance testing or move toward production, you can quickly identify which areas require further optimization.

Approach	Benefit
Evaluate the elasticity demands for the identified flows. **Explore design patterns** that can be implemented across the technology stack, considering the application and the underlying compute and data layers.	You're able to **define scalability requirements** on existing components that need more capacity and the areas where you need extra components to distribute load. You're aware of potential bottlenecks in the system and **design compensating controls**, such as adding caching capabilities to decrease latency and system load.
Choose the right resources across the technology stack, which enables you to meet performance goals and integrate with the system. **Consider features** that can fulfill the scalability requirements. **Find the right balance between resource allocation and system requirements**, to handle unexpected surges efficiently.	By analyzing the varying capabilities of the resources, you ensure that each component contributes to the **overall functionality and performance of the system**. You can **take advantage of the built-in capabilities** that automatically trigger scaling operations. Right-sizing resources can meet changes in demand without overprovisioning, which leads to cost savings.
Do capacity planning based on demand and the capability of selected resources to enrich your performance model. **Use predictive modeling techniques** to forecast anticipated changes in capacity that can occur with predictable and unexpected changes. **Define performance targets** that can be translated into technical requirements.	You can **efficiently use resources** and meet the demand without overprovisioning, thereby avoiding unnecessary costs. You understand how the design choices affect performance.
Implement a proof of concept that validates the technical requirements and design choices.	A proof of concept is instrumental in **validating the design** to determine if the system can meet the performance targets and if those targets are realistic. Based on the anticipated load, you can validate whether anticipated capacity can meet the performance targets.

Approach	Benefit
	Also, verify the cost implications of the design choices.
Document your performance testing strategy. Include use cases, different methodologies, and cadence of your test plans. Define a process for operation outlined by the performance test plan. Triage and prioritize the test cases in the plan. Focus on cases that offer valuable insights into performance targets and align capacity planning.	You ensure that the **right aspects of the system are tested**. You can allocate resources effectively and conduct tests in a manner that aligns with the business priorities and requirements.
Document your performance monitoring strategy. Assess metrics at different abstraction levels for each identified flow.	You can **track progress towards attainment** of performance targets throughout the development cycle.

Achieve and sustain performance

Protect against performance degradation while the system is in use and as it evolves.

Development isn't a one-time effort. It's an ongoing process. Expect changes in performance as features change. There's variance in user patterns and profiles, even changes from optimizations in other Azure Well-Architected pillars. Any change can strain workload resources.

Safeguard the system from changes so that it doesn't slide back on performance targets. **Integrate testing and monitoring in the development process**. Test the system's performance in production with real load and simulate that load with automated testing prior to production. In both cases, you should have monitoring practices in place for verification purposes.

Throughout the development lifecycle, **conduct various types of tests at different stages**. In the initial stages, test the proof of concept to make sure performance results aren't entirely unexpected. As development progresses, conduct **manual, low-effort**

tests to establish benchmarks. In the build stage, start developing **automated routine performance tests** that evaluate latency, stress levels, load capacity, and other characteristics defined in the test plans.

Monitoring must be an integral part of that effort, rather than being an isolated exercise. You can **see how the system and its resources perform over time. You can then fine-tune them to maximize their value**, and ensure they continue to meet performance standards.

Keep in mind that performance targets vary over time, in response to changes. Update the performance model based on tested and monitored metrics. Clearly indicate increased, reduced, or no effect on the performance of the flows.

Always be ready to renegotiate and reset expectations with business stakeholders.

Approach	Benefit
Integrate routine performance tests in Azure Pipelines. Choose pipelines that can integrate tests. Conversely, choose test tools that can be integrated into the pipelines.	Automated tests save time and provide consistency that makes it **easier to detect regressions or improvements**. These artifacts allow for continuous monitoring of any deviations or drift over time, so you can maintain consistent performance and quality.
Formalize performance tests as quality gates that can approve or deny release promotion and the final deployment to production.	These checkpoints ensure that **each stage of deployment meets the required performance standards** before you proceed to the next. The checkpoints help prevent unintended performance regression. For instance, if the performance is significantly below expectations, you might block a release until improvements are made.
Set up a repeatable process for monitoring real transactions in production and deviations against your performance targets. Use synthetic transactions in production. Set up monitoring alerts on performance regressions.	You want insight into the **actual performance of your system under real-world load** that couldn't be simulated through tests. Then you can proactively identify issues and areas of improvement such as potential bottlenecks, underutilized resources, and other concerns.
Review performance test results and monitoring data meticulously and optimize until you meet the	Based on test results, **you can capture and compare data** and start analyzing trends.

Approach	Benefit
performance targets. Prioritize actions derived from those reviews and add them to the backlog for planned execution.	Your optimization efforts are data driven.
Build coding skills that focus on performance. Have coding standards that exemplify performance-driven coding patterns.	Code that doesn't have performance issues can make **testing cycles more efficient** because tests can be focused on more significant issues. Coding patterns helps avoid rework and keeps your coding style consistent.
Address performance erosion as usage increases, features change, and data accumulates over time to sustain performance. Reset expectations and establish new targets, if fine-tuning brings only short-term benefits.	You can **preserve the performance state** before degradation develops into problems that negatively affect user experience beyond the acceptable range. Changing targets resets the performance model, and you don't waste time in optimizing the system that has already reached its capacity.

Improve efficiency through optimization

 **Improve system efficiency within the defined performance targets to increase workload value.**

The targets set during the initial phase are based on a reasonable level of user experience, considering various constraints. You should **reassess and adjust targets to further enhance the experience**. To further enhance the experience, it requires a clear understanding of how the system is used, how it has evolved, and how the platform or technology has changed over time. The cycle of monitoring, optimizing, testing, and deploying is a continuous process.

Efficiency optimization efforts allow a workload to work with lower resource consumption. They can cause the workload to be in an overprovisioned state with spare capacity. Use that capacity to improve reliability of the system. Eliminate capacity to improve the cost of the system. Or repurpose the capacity to support new product features on existing resources.

When the system gains efficiencies, take the opportunity to set and maintain new performance targets.

Approach	Benefit
Allocate dedicated cycles for performance optimization to address nonfunctional requirements and optimizations in functional areas. Targets for this optimization are resources, code, data retention, database queries, and others.	You can **build a culture of performance-driven optimization**. You keep the team accountable for proactively monitoring performance patterns and also fine-tune the application.
Enhance the architecture with new design patterns and components, which can boost performance, in ways that you previously didn't consider because of limited time or budget.	New design and components can optimize the system, **leading to better user experience**. For example, you can use caching or adding a content delivery network component. It can also lead to long-term cost benefits.
Use monitoring tools to analyze historical trends and to identify the flows and code implementation paths that would benefit the most from a performance optimization effort. We recommend application performance monitoring (APM) tools and profilers for this purpose. Identify operation hot paths and other potential bottlenecks in the system.	When you identify the recurring problematic areas, the team can **focus where gains are the highest**.
Get current and stay current with technology innovations that can improve performance. Take advantage of the new versions released for the dependent frameworks and libraries. Similarly, use the new features for platform resources as they're updated and patched.	Adopting new technology can often be the motivating factor to **look for opportunities to improve**. Code that might have been slow in the past can become faster with these updates. You also want to be aware of how certain updates negatively affect performance.

Next steps

Performance Efficiency checklist

Design review checklist for Performance Efficiency

Article • 11/14/2023

This checklist presents a set of recommendations for you to scale your system so it can grow and meet your workload usage demand. The goal of **performance** is to maintain the efficiency of every interaction with a healthy system as demand increases. When you design and implement for performance, focus on the **efficiency and effectiveness** of cost, complexity, supporting new requirements, technical debt, reporting, and toil.

For every system, there's a limit to how much you can scale it without redesigning, introducing a workaround, or incorporating human involvement. If you don't include performance efficiency practices and consider the tradeoffs, your design is potentially at risk. Carefully consider all the points covered in the checklist to instill confidence in your system's success.

Checklist

	Code	Recommendation
☐	PE:01	**Define performance targets.** Performance targets should be numerical values that are tied to workload requirements. You should implement performance targets for all workload flows.
☐	PE:02	**Conduct capacity planning.** Capacity planning should be done before there are predicted changes in usage patterns, such as seasonal variations, product updates, marketing campaigns, special events, or regulatory changes.
☐	PE:03	**Select the right services.** The services, infrastructure, and tier selections must support your ability to reach the workload's performance targets and accommodate expected capacity changes. The selections should also weigh the benefits of using platform features or building a custom implementation.
☐	PE:04	**Collect performance data.** Workload components and flows should provide automatic, continuous, and meaningful metrics and logs. Collect data at different levels of the workload, such as the application, platform, data, and operating system levels.
☐	PE:05	**Optimize scaling and partitioning.** Incorporate reliable and controlled scaling and partitioning. The scale unit design of the workload is the basis of the scaling and partitioning strategy.
☐	PE:06	**Test performance.** Perform regular testing in an environment that matches the production environment. Compare results against the performance targets and the

	Code	Recommendation
		performance benchmark.
☐	PE:07	**Optimize code and infrastructure.** Use code that's performant, and ensure that it offloads responsibilities to the platform. Use code and infrastructure only for their core purpose and only when necessary.
☐	PE:08	**Optimize data usage.** Optimize data stores, partitions, and indexes for their intended and actual use in the workload.
☐	PE:09	**Prioritize critical flows.** The allocation of workload resources and performance optimization efforts should prioritize the flows that support the most important business processes, users, and operations.
☐	PE:10	**Optimize operational tasks.** Monitor and minimize the effects of the software development lifecycle and other routine operations on workload performance. These operations include virus scans, secret rotations, backups, reindexing databases, and deployments.
☐	PE:11	**Respond to live performance issues.** Plan how to address performance problems by incorporating clear lines of communication and responsibilities. When a problematic situation occurs, use what you learn to identify preventive measures and incorporate them in your workload. Implement methods to return to normal operations faster when similar situations occur.
☐	PE:12	**Continuously optimize performance.** Focus on components that show deteriorating performance over time, such as databases and networking features.

Next steps

We recommend that you review the Performance Efficiency tradeoffs to explore other concepts.

Performance Efficiency tradeoffs

Performance Efficiency tradeoffs

Article • 10/10/2024

A workload that meets its performance targets without overprovisioning is efficient. The goal of performance efficiency is to have just enough supply to handle demand at all times. Key strategies for performance efficiency include proper use of code optimizations, design patterns, capacity planning, and scaling. Clear performance targets and testing underpin this pillar.

During the process of negotiating a workload's performance targets and designing a workload for performance efficiency, it's important to be aware of how the Performance Efficiency design principles and the recommendations in the Design review checklist for Performance Efficiency might affect the optimization goals of other pillars. Certain performance efficiency decisions might benefit some pillars but constitute tradeoffs for others. This article lists example tradeoffs that a workload team might encounter when designing workload architecture and operations for performance efficiency.

Performance Efficiency tradeoffs with Reliability

>  **Tradeoff: Reduced replication and increased density.** A cornerstone of reliability is ensuring resilience by using replication and limiting the blast radius of malfunctions.

- A workload that achieves efficiency by delaying scaling until the last responsible moment closely meets demand but is vulnerable to unforeseen node failures and scaling delays.
- Consolidating workload resources can use excess capacity and improve efficiency. However, it increases the blast radius of a malfunction in the co-located component or application platform.
- Scaling in or scaling down to minimize surplus capacity can leave a workload underprovisioned during usage spikes, which leads to service disruptions due to insufficient supply.

>  **Tradeoff: Increased complexity.** Reliability prioritizes simplicity.

- Using autoscaling to balance workload supply against demand introduces variability in the workload's topology and adds a component that must work correctly for the system to be reliable. Autoscaling leads to triggering more application lifecycle events, like starting and stopping.
- Data partitioning and sharding help avoid performance issues in large or frequently accessed datasets. However, the implementation of these patterns increases complexity because (eventual) consistency needs to be maintained across additional resources.
- Denormalizing data for optimized access patterns can improve performance, but it introduces complexity because multiple representations of data need to be kept synchronized.
- Performance-centric cloud design patterns sometimes necessitate the introduction of additional components. The use of these components increases the surface area of the workload. The components then must themselves be made reliable to keep the whole workload reliable. Examples include:
  - A message bus for load leveling, which introduces a critical, stateful component.
  - A load balancer for autoscaled replicas, which requires reliable operation and the enlistment of replicas.
  - Offloading data to caches, which requires reliable cache invalidation approaches.

>  **Tradeoff: Testing and observation on active environments.** Avoiding the unnecessary use of production systems is a self-preservation and risk avoidance approach for reliability.

- Performance testing in active environments, like the use of synthetic transactions, carries the risk of causing malfunctions due to the test actions or configurations.
- Workloads should be instrumented with an application performance monitoring (APM) system that enables teams to learn from active environments. The APM tooling is installed and configured in application code or in the hosting environment. Improper use, exceeding limitations, or misconfiguration of the tool can compromise its functionality and maintenance, potentially undermining reliability.

Performance Efficiency tradeoffs with Security

> **Tradeoff: Reduction of security controls.** Security controls are established across multiple layers, sometimes redundantly, to provide defense in depth.

One performance optimization strategy is to remove or bypass components or processes that contribute to delays in a flow, especially when their processing time isn't justified. However, this strategy can compromise security and should be accompanied by a thorough risk analysis. Consider the following examples:

- Removing encryption in transit or at rest to improve transfer speeds exposes the data to potential integrity or confidentiality breaches.
- Removing or reducing security scanning or inspecting tools to reduce processing times can compromise the confidentiality, integrity, or availability that those tools protect.
- Decreasing the frequency of security patching to limit the performance impact can leave a workload more vulnerable to emerging threats.
- Removing firewall rules from network flows to improve network latency can allow undesirable communication.
- Minimizing data validation or content safety checks for quicker data processing might compromise data integrity, especially if inputs are malicious.
- Using less entropy in encryption or hashing algorithms, for example, on the initialization vector (IV), is more efficient but makes the encryption easier to crack.

> **Tradeoff: Increased workload surface area.** Security prioritizes a reduced and contained surface area to minimize attack vectors and reduce the management of security controls.

Performance-centric cloud design patterns sometimes necessitate the introduction of additional components. These components increase the surface area of the workload. The new components must be secured, possibly in ways that aren't already used in the system, and they often increase the compliance scope. Consider these commonly added components:

- A message bus for load leveling
- A load balancer for autoscaled replicas
- Offloading data to caches, application delivery networks, or content delivery networks

- Offloading processing to background jobs or even client compute

>  **Tradeoff: Removing segmentation.** The Security pillar prioritizes strong segmentation to enable fine-grained security controls and reduce blast radius.

Sharing resources through increased density is an approach for improving efficiency. Examples include multitenancy scenarios or combining disparate applications in an architecture on a common application platform. The increased density can lead to the following security concerns:

- Increased risk of unauthorized lateral movement from one tenant to another.
- A shared workload identity that violates the principle of least privilege and obscures individual audit trails in access logs.
- Perimeter security controls, for example network rules, that are reduced to cover all co-located components, giving individual components more access than necessary.
- A compromise of the application platform host or an individual component due to a larger blast radius. This increase is caused by easier access to co-located components.
- Co-locating disparate components leading to more components in scope for compliance because of their shared host.

Performance Efficiency tradeoffs with Cost Optimization

>  **Tradeoff: Too much supply for demand.** Both Cost Optimization and Performance Efficiency prioritize having just enough supply to serve demand.

- Overprovisioning is a risk when teams try to mitigate performance issues in a workload. Some common causes of overprovisioning include:
  - Initial capacity planning was misjudged because the team focused only on peak load estimates, neglecting strategies for peak smoothing in the workload design.
  - Scaling a resource up or out during a troubleshooting step of an incident response.

- Autoscaling can be misconfigured. Some examples of misconfigured autoscaling include:
  - Scaling up with minimal changes in demand or an extended cooldown period can incur more cost than demand requires.
  - Using autoscaling without a set upper limit can lead to uncontrolled growth due to system malfunctions or abuse and exceed the expected workload requirements.

- Expanding into multiple regions can enhance performance by bringing workloads closer to the user and can avoid temporary resource capacity constraints. However, that topology also adds complexity and resource duplication.

>  **Tradeoff: More components.** One cost optimization technique is to consolidate with a smaller number of resources by increasing density, removing duplication, and co-locating functionality.

- Performance-centric cloud design patterns sometimes necessitate the introduction of extra components. These extra components usually lead to an overall cost increase for the workload. For example, you might include a message bus for load leveling or offload tasks to an application or content delivery network for improved response times.

- Resource segmentation allows different parts of a workload to have distinct performance characteristics, enabling independent tuning for each segment. However, it can increase the total ownership costs because it requires multiple optimized segments rather than a single, generalized component.

>  **Tradeoff: Increased investment on items that aren't aligned with functional requirements.** One approach to cost optimization is evaluating the value provided by any solution that's deployed.

- Premium services and SKUs can help a workload meet performance targets. These services usually cost more and can provide extra features. They might be underutilized if many of the premium features aren't used specifically for meeting performance targets.

- A performant workload requires telemetry data for observability that must be transferred and stored. An increase in the performance telemetry being collected can increase the cost of telemetry data transfer and storage.

- Performance testing activities add costs that aren't associated with the value of the production system. Examples of performance testing costs include:
  - Instantiating environments that are dedicated to performance-centric tests.
  - Using specialized performance tooling.
  - Spending time to run the tests.

- Training team members for specialized performance optimization tasks or paying for performance tuning services adds to the cost of a workload.

Performance Efficiency tradeoffs with Operational Excellence

>  **Tradeoff: Reduced observability.** Observability is necessary to provide a workload with meaningful alerting and help ensure successful incident response.

- Reducing log and metric volume to reduce the processing time spent on collecting telemetry instead of other tasks reduces the overall observability of the system. Some examples of the resulting reduced observability include:
  - It limits the data points that are used to build meaningful alerts.
  - It leads to gaps in coverage for incident response activities.
  - It limits observability in security-sensitive or compliance-sensitive interactions and boundaries.

- When performance design patterns are implemented, the complexity of the workload often increases. Components are added to critical flows. The workload monitoring strategy and performance monitoring must include those components. When a flow spans multiple components or application boundaries, the complexity of monitoring the performance of that flow increases. Flow performance needs to be correlated across all the interconnected components.

>  **Tradeoff: Increased complexity in operations.** A complex environment has more complex interactions and a higher likelihood of a negative impact from routine, ad hoc, and emergency operations.

- Improving performance efficiency by increasing density elevates the risk in operational tasks. An error in a single process can have a large blast radius.

- As performance design patterns are implemented, they influence operational procedures like backups, key rotations, and recovery strategies. For example, data

partitioning and sharding can complicate routine tasks when teams try to ensure that those tasks don't affect data consistency.

>  **Tradeoff: Culture stress.** Operational Excellence is rooted in a culture of blamelessness, respect, and continuous improvement.

- Conducting root cause analysis of performance issues identifies deficiencies in processes or implementations that require correction. The team should consider the exercise a learning opportunity. If team members are blamed for issues, morale can be affected.
- Routine and ad hoc processes can affect workload performance. It's often considered preferable to perform these activities during off-peak hours. However, off-peak hours can be inconvenient or outside of regular hours for the team members who are responsible for or skilled in these tasks.

Related links

Explore the tradeoffs for the other pillars:

- Reliability tradeoffs
- Security tradeoffs
- Cost Optimization tradeoffs
- Operational Excellence tradeoffs

Feedback

Was this page helpful? Yes No

Cloud design patterns that support performance efficiency

Article • 10/10/2024

When you design workload architectures, you should use industry patterns that address common challenges. Patterns can help you make intentional tradeoffs within workloads and optimize for your desired outcome. They can also help mitigate risks that originate from specific problems, which can impact reliability, security, cost, and operations. If not mitigated, risks will eventually lead to performance inefficiencies. These patterns are backed by real-world experience, are designed for cloud scale and operating models, and are inherently vendor agnostic. Using well-known patterns as a way to standardize your workload design is a component of operational excellence.

Many design patterns directly support one or more architecture pillars. Design patterns that support the Performance Efficiency pillar address scalability, performance tuning, task prioritization, and the removal of bottlenecks.

Design patterns for performance efficiency

The following table summarizes cloud design patterns that support the goals of performance efficiency.

Expand table

Pattern	Summary
Asynchronous Request-Reply	Improves the responsiveness and scalability of systems by decoupling the request and reply phases of interactions for processes that don't need immediate answers. By using an asynchronous pattern, you can maximize concurrency on the server side. You can use this pattern to schedule work to be completed as capacity allows.
Backends for Frontends	Individualizes the service layer of a workload by creating separate services that are exclusive to a specific frontend interface. This separation enables you to optimize in ways that might not be possible with a shared service layer. When you handle individual clients differently, you can optimize performance for a specific client's constraints and functionality.
Bulkhead	Introduces segmentation between components to isolate the blast radius of malfunctions. This design enables each bulkhead to be individually scalable to meet the needs of the task that's encapsulated in the bulkhead.

Pattern	Summary
Cache-Aside	Optimizes access to frequently read data by introducing a cache that's populated on demand. The cache is then used on subsequent requests for the same data. This pattern is especially useful with read-heavy data that doesn't change often and can tolerate a certain amount of staleness. The goal of this implementation is to provide better performance in the system overall by offloading this type of data to a cache instead of sourcing it from its data store.
Choreography	Coordinates the behavior of autonomous distributed components in a workload by using decentralized, event-driven communication. This pattern can provide an alternative when performance bottlenecks occur in a centralized orchestration topology.
Circuit Breaker	Prevents continuous requests to a malfunctioning or unavailable dependency. A retry-on-error approach can lead to excessive resource utilization during dependency recovery and can also overload performance on a dependency that's attempting recovery.
Claim Check	Separates data from the messaging flow, providing a way to separately retrieve the data related to a message. This pattern improves the efficiency and performance of message publishers, subscribers, and the message bus itself when the system handles large data payloads. It works by decreasing the size of messages and ensuring that consumers retrieve payload data only if necessary and at an opportune time.
Competing Consumers	Applies distributed and concurrent processing to efficiently handle items in a queue. This model supports distributing load across all consumer nodes and dynamic scaling that's based on queue depth.
Compute Resource Consolidation	Optimizes and consolidates compute resources by increasing density. This pattern combines multiple applications or components of a workload on a shared infrastructure. This consolidation maximizes the utilization of computing resources by using spare node capacity to reduce overprovisioning. Container orchestrators are a common example. Large (vertically scaled) compute instances are often used in the resource pool for these infrastructures.
Command and Query Responsibility Segregation (CQRS)	Separates the read and write operations of an application's data model. This separation enables targeted performance and scaling optimizations for each operation's specific purpose. This design is most helpful in applications that have a high read-to-write ratio.
Deployment Stamps	Provides an approach for releasing a specific version of an application and its infrastructure as a controlled unit of deployment, based on the assumption that the same or different versions will be deployed concurrently. This pattern often aligns to the defined scale units in your workload: as additional capacity is needed beyond what a single

Pattern	Summary
	scale unit provides, an additional deployment stamp is deployed for scaling out.
Event Sourcing	Treats state change as series of events, capturing them in an immutable, append-only log. Depending on your workload, this pattern, usually combined with CQRS, an appropriate domain design, and strategic snapshotting, can improve performance. Performance improvements are due to the atomic append-only operations and the avoidance of database locking for writes and reads.
Federated Identity	Delegates trust to an identity provider that's external to the workload for managing users and providing authentication for your application. When you offload user management and authentication, you can devote application resources to other priorities.
Gatekeeper	Offloads request processing that's specifically for security and access control enforcement before and after forwarding the request to a backend node. This pattern is often used to implement throttling at a gateway level rather than implementing rate checks at the node level. Coordinating rate state among all nodes isn't inherently performant.
Gateway Aggregation	Simplifies client interactions with your workload by aggregating calls to multiple backend services in a single request. This design can incur less latency than a design in which the client establishes multiple connections. Caching is also common in aggregation implementations because it minimizes calls to backend systems.
Gateway Offloading	Offloads request processing to a gateway device before and after forwarding the request to a backend node. Adding an offloading gateway to the request process enables you to use less resources per-node because functionality is centralized at the gateway. You can optimize the implementation of the offloaded functionality independently of the application code. Offloaded platform-provided functionality is already likely to be highly performant.
Gateway Routing	Routes incoming network requests to various backend systems based on request intents, business logic, and backend availability. Gateway routing enables you to distribute traffic across nodes in your system to balance load.
Geode	Deploys systems that operate in active-active availability modes across multiple geographies. This pattern uses data replication to support the ideal that any client can connect to any geographical instance. You can use it to serve your application from a region that's closest to your distributed user base. Doing so reduces latency by eliminating long-distance traffic and because you share infrastructure only among users that are currently using the same geode.

Pattern	Summary
Health Endpoint Monitoring	Provides a way to monitor the health or status of a system by exposing an endpoint that's specifically designed for that purpose. You can use these endpoints to improve load balancing by routing traffic to only nodes that are verified as healthy. With additional configuration, you can also get metrics on available node capacity.
Index Table	Optimizes data retrieval in distributed data stores by enabling clients to look up metadata so that data can be directly retrieved, avoiding the need to do full data store scans. Clients are pointed to their shard, partition, or endpoint, which can enable dynamic data partitioning for performance optimization.
Materialized View	Uses precomputed views of data to optimize data retrieval. The materialized views store the results of complex computations or queries without requiring the database engine or client to recompute for every request. This design reduces overall resource consumption.
Priority Queue	Ensures that higher-priority items are processed and completed before lower-priority items. Separating items based on business priority enables you to focus performance efforts on the most time-sensitive work.
Publisher/Subscriber	Decouples components of an architecture by replacing direct client-to-service or client-to-services communication with communication via an intermediate message broker or event bus. The decoupling of publishers from consumers enables you to optimize the compute and code specifically for the task that the consumer needs to perform for the specific message.
Queue-Based Load Leveling	Controls the level of incoming requests or tasks by buffering them in a queue and letting the queue processor handle them at a controlled pace. This approach enables intentional design on throughput performance because the intake of requests doesn't need to correlate to the rate in which they're processed.
Scheduler Agent Supervisor	Efficiently distributes and redistributes tasks across a system based on factors that are observable in the system. This pattern uses performance and capacity metrics to detect current utilization and route tasks to an agent that has capacity. You can also use it to prioritize the execution of higher priority work over lower priority work.
Sharding	Directs load to a specific logical destination to handle a specific request, enabling colocation for optimization. When you use sharding in your scaling strategy, the data or processing is isolated to a shard, so it competes for resources only with other requests that are directed to that shard. You can also use sharding to optimize based on geography.

Pattern	Summary
Sidecar	Extends the functionality of an application by encapsulating non-primary or cross-cutting tasks in a companion process that exists alongside the main application. You can move cross-cutting tasks to a single process that can scale across multiple instances of the main process, which reduces the need to deploy duplicate functionality for each instance of the application.
Static Content Hosting	Optimizes the delivery of static content to workload clients by using a hosting platform that's designed for that purpose. Offloading responsibility to an externalized host helps mitigate congestion and enables you to use your application platform only to deliver business logic.
Throttling	Imposes limits on the rate or throughput of incoming requests to a resource or component. When the system is under high demand, this pattern helps mitigate congestion that can lead to performance bottlenecks. You can also use it to proactively avoid noisy neighbor scenarios.
Valet Key	Grants security-restricted access to a resource without using an intermediary resource to proxy the access. Doing so offloads processing as an exclusive relationship between the client and the resource without requiring an ambassador component that needs to handle all client requests in a performant way. The benefit of using this pattern is most significant when the proxy doesn't add value to the transaction.

Next steps

Review the cloud design patterns that support the other Azure Well-Architected Framework pillars:

- Cloud design patterns that support reliability
- Cloud design patterns that support security
- Cloud design patterns that support operational excellence
- Cloud design patterns that support cost optimization

Feedback

Was this page helpful? Yes No

Recommendations for defining performance targets

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Performance Efficiency checklist recommendation:

> **PE:01** **Define performance targets. Performance targets should be numerical values that are tied to workload requirements. You should implement performance targets for all workload flows.**

This guide describes the recommendations for establishing and exposing performance targets. Performance targets are metrics that define performance objectives. These metrics are expressed as a single numerical value or a numerical range. They're clear and specific metrics that drive continuous improvement. Performance targets are a numerical foundation for improvements, and they help teams align their efforts toward specific goals. Without clear performance targets, teams might lack focus and lack of accountability for performance issues. By setting performance targets, teams can work toward specific objectives and drive continuous improvement.

Definitions

Term	Definition
Data flow	The movement of data within a system or between systems.
Dependency	A component that a workload relies on.
Flow	In a workload, a sequence of operations that performs a specific function. It involves the movement of data and the running of processes between components of the workload.
Metrics	Numerical values that are collected at regular intervals. Metrics describe some aspects of a system at a particular time.
Performance targets	Metrics that define performance objectives. These metrics are expressed as a single numerical value or a numerical range.
User flow	The paths or sequences of actions that users take within an application or system.
Workflow	The sequence of steps that a workload runs to accomplish a task.

Key design strategies

Establishing performance targets is an essential step for achieving workload performance efficiency. Performance targets define the desired level of performance for your workload and help you gauge its effectiveness in meeting those objectives. Performance targets provide a benchmark to measure and compare a workload's efficiency. This benchmark can help you highlight improvement areas. The targets also align tasks with your organization's objectives and enhance business outcomes. Additionally, performance targets offer guidance in resource allocation, helping you ensure that workloads can adapt to varying demands while maintaining optimal performance.

Set performance targets early

Set performance targets before you deploy your workload. For a workload in a design, performance targets require research. Conduct market research, competitive analysis, and surveys to generate your performance target ranges. For a production workload that has no performance targets, use production data and customer feedback to establish performance targets.

Determine performance requirements

Determining performance requirements is about identifying essential performance metrics like response time, throughput, and latency that are critical for your application. Aligning these performance targets with your organization's business goals ensures the workload meets the desired standards, whether for a best-in-class or average product. For example, you might aim to reduce response times, increase throughput rates, or optimize resource use.

When setting performance goals, it's important to align the organization's objectives with the distinct needs of the user base. Users ultimately determine the success of performance, emphasizing the need to align performance targets with their expectations. This balance ensures that performance targets capture the intended user experience and the overall efficiency of the workload. To comprehensively gauge and optimize workload performance, you should consider setting performance targets for the following list:

- *Individual components*: Individual components are the separate units or segments of the workload, each potentially having distinct performance attributes and demands.

- *User flows*: These pathways chart how users maneuver through the workload, and ensuring their fluidity directly enhances user experience.
- *Workflows*: Workflows defined internal processes are crafted to achieve particular results and often dictate operational efficiency.
- *Data flows*: Data flows refer to the movement and interaction of data within the workload, helping identify potential inefficiencies or bottlenecks.
- *External dependencies*: External dependencies are elements outside the primary workload (integrated third-party services or tools) that can significantly affect performance.
- *Scale units*: Scale units relate to the workload's scalable segments. Ensuring robust performance under increased loads is pivotal, especially in growth scenarios.
- *Technology levels*: Technology levels are direct performance indicators such as the speed of API access, database operation latencies, and potential network delays.
- *Business transactions*: Business transactions represent end-to-end user operations like completing a purchase or booking a service, their seamless execution is directly tied to user satisfaction.
- *Workload all up*: This holistic metric gives an overview of the collective performance encompassing all components and aspects of the workload.

Identify key metrics

Identifying key performance metrics involves determining the essential measurements that track the progress towards achieving workload performance goals. This identification provides a quantifiable way to measure and improve performance efficiency. When you identify key metrics to focus on, consider metrics related to availability, capacity, and response time:

- *Availability*: Error rate is an availability performance metric. Error rate represents the percentage of failed requests over a period. A common target for error rate is 0.1% percent of requests.
- *Capacity*: Throughput and concurrency are sample capacity metrics. Throughput refers to the ability to handle a specific number of transactions within a given time period. For instance, an application might need to sustain 100 million transactions per month. Concurrency is a measure of simultaneous users or actions.

- *Response time:* Latency and load time are common response time metrics. Latency is the time it takes to respond to a request (200 milliseconds). Load time is the time it takes for an application or web page to be interactive. A common target is 99% of sign-in requests completing less than 1 second.

Set specific targets

After you identify the key metrics, you need to specify performance targets or thresholds for each metric. Performance targets should be measurable, realistic, and aligned with your workload objectives. For example, you might set a target response time of less than 500 milliseconds (ms) or a target error rate of less than 1 percent. Avoid qualitative assessments of performance like *fast* or *slow*. By using numerical targets, you can objectively assess performance over time. As you set specific performance targets, consider these recommendations:

- *Consider the customer*: When you set performance targets, adopt a customer-centric perspective. Recognizing the customer as the ultimate judge of performance helps ensure that performance targets align with customer expectations. This alignment involves considering both organizational objectives and the distinct requirements of the customer base. When you integrate these two aspects, you can tailor performance targets to reflect the desired customer experience and overall workload effectiveness. By defining performance objectives that consider customer expectations, you can strive to provide a high-quality customer experience and meet the needs of your customers.

- *Use percentiles*: Percentiles, such as P99, P95, and P50, are the industry standard to represent the result of performance assessments. Percentiles are measures that indicate how much data the number includes. For example, P99 covers 99% of the data. Use percentiles, rather than simple averages, to provide a more comprehensive understanding of workload performance. To measure percentiles, collect performance data over a period of time, typically using monitoring tools or logging mechanisms. Then analyze this data to determine the response time values at different percentiles.

Document and expose performance target

Documenting and exposing performance targets is about recording all performance targets in a centralized location. Meeting performance targets is a shared responsibility between development and operations teams. To ensure that the workload consistently meets or exceeds these targets, provide teams with the information and access to take action. To document and expose performance targets, consider these recommendations:

- *Document performance targets*: Document all performance targets. Ensure that all performance targets are documented in a centralized location, easily accessible by both development and operations teams. It promotes alignment and aids in real-time decision-making.
- *Expose performance targets*: All responsible teams should be able to review and create actionable tasks from the performance targets. Use information radiators, such as dashboards and reports, to make the performance targets accessible.
- *Make it actionable*: The documentation and information radiators should suggest clear next steps. For example, a rise in errors might prompt an immediate check, or meeting a target consistently might suggest a reevaluation of that benchmark.

Evaluate customer feedback

Evaluating customer feedback involves actively seeking out and analyzing the responses and suggestions of your customers. Actively collecting and analyzing customer feedback offers valuable insights into their needs and expectations. Regular communication helps in adjusting performance targets in line with changing preferences and tech trends. A focus on customer needs means that the workload not only aligns with technical benchmarks but also undergoes continuous refinement. This approach, emphasizing customer satisfaction, ensures that the workload remains relevant and successful in the long run.

Azure facilitation

Setting performance targets: Azure Advisor provides performance recommendations that can inform your performance targets.

Azure Monitor is a full-stack monitoring service that provides a complete set of features to monitor your Azure resources and measure performance targets. It collects platform metrics and provides ready-to-use dashboards. It allows you to configure alerts based on metrics. It also stores and correlates metrics to ensure a single source of truth.

Related links

- Azure Advisor performance recommendations
- Azure Monitor

Performance Efficiency checklist

Refer to the complete set of recommendations.

Performance Efficiency checklist

Recommendations for capacity planning

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Performance Efficiency checklist recommendation:

> **PE:02** **Conduct capacity planning. Capacity planning should be done before there are predicted changes in usage patterns. Predicted changes include as seasonal variations, product updates, marketing campaigns, special events, or regulatory changes.**

This guide describes the recommendations for capacity planning. Capacity planning refers to the process of determining the resources required to meet workload performance targets. It involves estimating the amount of computing resources such as CPU, memory, storage, and network bandwidth needed to support the workload's performance requirements. Capacity planning helps avoid underprovisioning and ensures the workload has sufficient resources to handle the expected workload demands without experiencing performance degradation or bottlenecks. It also helps prevent overprovisioning and unnecessary costs. A lack of capacity planning can lead to performance issues, resource bottlenecks, increased costs, inefficient allocation, scalability challenges, and unpredictable workload performance.

Definitions

Term	Definition
Capacity planning	The process of predicting the resources a workload needs to meet its performance targets.
Functional requirements	The features and capabilities a workload must have to fulfill its intended purpose.
Technical requirements	The code and infrastructure needed to meet functional requirements.
Trend analysis	Historical data analysis to forecast future demand.

Key design strategies

Capacity planning is a forward-looking process that involves making decisions based on anticipated workload demands and patterns. Its goal is to optimize workload performance across both continuous and peak load scenarios. By understanding changes in usage, such as seasonal shifts or product releases, you can allocate resources

strategically, preventing system strain during high demand periods. This proactive strategy reduces disruptions and bolsters performance efficiency. By analyzing past usage trends and growth data, you can forecast short and long-term needs. You can pinpoint potential bottlenecks and scaling issues, ensuring consistent and efficient workload performance.

Gather capacity data

Gathering workload utilization data entails collecting and analyzing information on how a workload uses resources. You should collect data on historical patterns for existing workloads and predictive measures for new workloads. This process helps translate business objectives into technical requirements and is essential for forecasting capacity. Consider the following recommendations:

Understand an existing workload

Understanding an existing workload for capacity planning involves analyzing historical data related to how the workload utilizes resources. It encompasses metrics like resource utilization, performance data, and workload patterns. This understanding ensures efficient resource allocation, translates business goals into technical requirements, and helps identify potential bottlenecks.

- *Understand the data*: Review the available historical data and understand its structure, format, and relevance to capacity planning. The review might include resource utilization metrics, workload patterns, performance metrics, and other relevant data points. Understand the business processes and the criticality of the applications. Identify the peak usage times, user load, transaction rates, and other relevant metrics.

- *Clean and preprocess the data*: Prepare the data for analysis by removing any inconsistencies, errors, or outliers. Preparing the data might involve data cleaning techniques like data imputation, the handling of missing values, or normalization.

- *Identify key metrics*: Identify the metrics that are relevant for capacity planning. Metrics can include CPU utilization, memory usage, network throughput, and response times.

- *Identify bottlenecks*: Measure throughput and response times to identify the specific components of your system that might become bottlenecks as the workload grows. Requests per second and database CPU usage can be good indicators of capacity.

- *Visualize the data*: Create visualizations, like charts or plots, to gain better insights into historical data. Visualizations can help you identify patterns, trends, and anomalies in data to give you a clearer understanding of workload behavior.

Understand a new workload

Understanding a new workload for capacity planning refers to predicting the resource requirements of a future task without historical data. Predicting the future needs of new a workload without historical data can be more challenging. This process ensures you allocate resources efficiently and align allocations with workload objectives when the workload is introduced. Consider the following recommendations:

- *Market research*: Conducting market research to understand the demand for similar products or services can provide valuable insights into the potential demand for a new workload. The research can involve analyzing market trends, conducting surveys, or studying competitor offerings.

- *Expert judgment*: Input from subject matter experts or professionals who have experience in the industry can help you estimate the demand for a new workload. Their expertise and insights can provide valuable inputs for forecasting.

- *Pilot projects or prototypes*: Small-scale pilot projects or prototypes can help you gather real-time data and feedback. You can then use this data to inform the capacity planning process and adjust the forecasted demand.

- *External data sources*: External data sources like industry reports, market studies, or customer surveys can provide additional information for estimating demand for a new workload. These sources can offer valuable insights into customer preferences, market trends, and potential demand drivers.

Forecast demand

Forecasting demand involves using workload data to predict future needs for a service or product. It's essential for capacity planning to ensure efficient resource allocation, anticipate growth patterns, and prepare for potential surges in demand. When you forecast future demand, you use data to get a sense of future needs. You apply statistical analysis, trend analysis, or predictive modeling techniques to the data you have to forecast future demand. These methods take into account historical or anticipated patterns and project them into the future to provide estimates of the expected workload demand. To forecast demand, consider these strategies:

Account for various scenarios

When you perform capacity planning, you need to plan for different scenarios that might occur. This planning should include both predictable growth patterns and unexpected surges in demand. Usage patterns can grow or shrink. They can be organic (more or less users) or inorganic (an event or security incident). You need to conduct capacity planning before usage changes, at key times:

- Design (prediction)
- Regular spikes (8:00 AM sign-in rush)
- Launch (prediction validation)
- Business model change
- Acquisition or merger
- Marketing push
- Seasonal change
- Feature launch
- Periodically

Use prediction techniques

Forecasting future demand for a service or product involves using techniques like statistical analysis, trend analysis, and predictive modeling. Here's an overview of how you can use these techniques:

- *Statistical analysis*: Statistical methods can you help uncover patterns and relationships within historical data. You can use these patterns to forecast future demand. You can use techniques like time series analysis, regression analysis, and moving averages to identify trends, seasonality, and other patterns in the data.

- *Trend analysis*: Trend analysis involves examining historical data to identify consistent patterns and extrapolating those patterns into the future. For example, if workload demand increased by 10 percent during the past year, you might forecast a continuation of this trend. When you analyze historical demand data over a period of time, you can identify growth or reduction trends. Use these trends as a basis for forecasting future demand. Trend analysis can also identify the effects of one-time events that cause rapid shifts in traffic (inorganic). For example, feature releases might consistently increase demand by 5 percent. If you have four major releases a year, you should plan for 5 percent growth each time.

- *Predictive modeling*: Predictive modeling is the process of building mathematical models that use historical data and other relevant variables to make predictions about future demand. You can use techniques like machine learning algorithms,

neural networks, or decision trees. These models can take into account multiple factors and variables to provide more accurate forecasts.

Align forecasts with workload objectives

Aligning forecasts with workload objectives involves adjusting predictive capacity models to ensure they meet the specific goals and demands of a given workload. This alignment ensures resources are adequately provisioned, preventing both underutilization and potential workload overloads. For example, if you aim to support an API for 1 million users to upload 1-MB files in a second, but current data shows slow write speeds, you need to adjust your system. It's essential to talk with stakeholders to grasp the workload's requirements. Make sure your plans align with the promises (SLAs) of your service providers. This alignment ensures your capacity meets the expected demand and helps pinpoint areas of the system that might need changes.

Determine resource requirements

Determining resource requirements for capacity planning involves assessing the resources that you need to meet forecasted demand. For example, if an application anticipates a 50% increase in users during a promotional campaign, it might need to allocate more cloud instances or adjust its autoscaling parameters to handle the increased load.

A workload can have many resources, so there's no one metric to observe to determine resource requirements. You need to measure capacity at the resource level to get meaningful results. Estimate the expected demand for your resources based on historical data, market trends, and business projections. Consider the number of transactions, concurrent users, or any other relevant metrics.

Based on the forecasted demand, calculate the resources needed to meet that demand. Consider factors such as server capacity, network bandwidth, storage capacity, and personnel:

- *Server capacity*: Determine the required server capacity based on the estimated number of concurrent users or transactions. Consider factors like CPU, memory, and disk space requirements to ensure that your servers can handle the expected workload.
- *Network bandwidth*: Evaluate the network bandwidth that you need to support the anticipated level of traffic. You should include both inbound and outbound data transfer rates to ensure smooth and efficient communication between servers and clients.

- *Storage capacity*: Estimate the amount of data that the workload generates or processes during the forecasted demand. Consider factors like database size, file storage requirements, and any other data storage needs that are specific to your application.
- *Personnel*: Assess the human resources that are required to manage and maintain the infrastructure, handle customer support, perform system maintenance, and ensure smooth operations. Take into account factors like workload distribution, skill set, and required expertise.

Understand resource limitations

Resources in your workload have performance limitations. Performance limitations apply to services and SKUs within each service. You need to understand the limitations of the resources in your workload and factor those limitations into your design decisions. For example, you should know whether resource limitations require you to change SKUs or to change resources altogether.

You also need to identify reachable limits. It refers to pinpointing the maximum thresholds or boundaries of a workload. These limits usually apply to infrastructure (compute, memory, storage, network), application (concurrent database connections, response times, availability), service (requests per second), and scaling. When capacity planning identifies reachable limits, you need to modify the workload before the limit creates a performance problem. Performance baselines, continuous monitoring, and testing are essential to validating the limits and the solution.

> **Tradeoff**: Misjudged capacity planning can lead to over-provisioning or under-provisioning of resources. Over-provisioning leads to higher costs. Under-provisioning can result in poor performance. Try to find the right balance.

Azure facilitation

Gathering capacity data and forecasting demand: Azure Monitor enables you to collect and analyze telemetry data from your applications and infrastructure. It supports the monitoring of various Azure resources, including virtual machines, containers, and storage accounts. Key tools include Application Insights and Log Analytics. By configuring data collection and defining metrics and logs that you want to monitor, you can gather valuable workload data for analysis. For network monitoring, combine Azure Monitor with Azure Network Watcher, Azure Monitor network insights, and Azure ExpressRoute monitoring.

Azure Monitor allows you to analyze historical data and apply forecasting techniques to predict future workload trends and capacity requirements. You can generate forecasts that can help you with capacity planning. These forecasts help estimate server capacity, network bandwidth, storage capacity, and other resource needs by using predicted demand patterns.

Determining resource requirements: Because they provide a wide range of configurations, Azure tools and services can help you define technical requirements. You can align your workload requirements with available Azure resources, ensuring that you select the appropriate components and settings to meet your functional needs.

Understanding resource limitations: Azure provides documentation and resources to help you understand the performance limitations of various Azure services and SKUs. Taking into consideration these limitations can help you make informed design decisions and optimize your workload architecture for performance and cost-effectiveness.

Azure provides scalability options like autoscaling, which can automatically adjust resources based on workload demand. You can scale vertically by increasing the capacity of a resource by using a larger virtual machine size, or you can scale horizontally by adding new instances of a resource. Azure services that have autoscaling capabilities can automatically scale out to ensure capacity during workload peaks and return to normal when the load decreases. There are scaling limits within your configuration and services that you should be aware of. You can read the documentation or run tests. Azure provides tools like Azure Load Testing, which can simulate load and different usage patterns to help you gather relevant data about your workload.

Related links

- Azure Monitor
- Application Insights
- Log Analytics
- Network monitoring services

Performance Efficiency checklist

Refer to the complete set of recommendations.

Performance Efficiency checklist

Recommendations for selecting the right services

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Performance Efficiency checklist recommendation:

> **PE:03** **Select the right services. The services, infrastructure, and tier selections must support your ability to reach the workload's performance targets and accommodate expected capacity changes. The selections should also weigh the benefits of using platform features or building a custom implementation.**

This guide describes the recommendations for selecting appropriate services for your workload. The following recommendations help you choose services that best meet the requirements and demands of your workload. When you use services that are designed to handle your workload's requirements, you can ensure that your workload meets your performance targets. If you choose inappropriate services for your workload, the services might not be capable of handling your workload's demands. Insufficient services can lead to slow response times, bottlenecks, or workload failures.

Definitions

Term	Definition
Availability zone	A separated group of datacenters within a region. Each availability zone is independent of the others, with its own power, cooling, and networking infrastructure. Many regions support availability zones.
Compute service	A service that provides the infrastructure that you need to run an application.
Database service	A service that provides relational and nonrelational databases for your application.
Infrastructure	The physical components of cloud computing, and the geographic location of the components.
Infrastructure as a service (IaaS)	A service in which the customer is responsible for the operating system, identity, applications, and networking.
Platform as a service (PaaS)	A service in which the cloud service provider is responsible for the operating system. The cloud service provider shares responsibility with the customer for managing identity, applications, and networking.
Region	A geographic perimeter that contains a set of datacenters.

Term	Definition
Resource	A single entity or component that you can create, configure, and utilize within a cloud service provider.
Service	A product or offering from a cloud service provider.
Stock keeping unit (SKU)	A service tier for an Azure service.
Storage service	A service that provides storage for objects, blocks, and files.

Key design strategies

The services you choose should align with your workload's performance targets and be adaptable to future capacity needs. As the workload expands or evolves, the services you use should match your performance standards without requiring major adjustments. Consider the balance between platform features and custom implementations. Platform features provide immediate solutions, but custom-built options offer precise tailoring. Your service selections should be both forward-thinking and tailored to your specific needs, taking into account the trade-offs between convenience and customization.

Understand workload requirements

Understanding workload requirements refers to grasping the technical and functional demands of a workload. This analysis helps determine the resources, storage, compute, network, and other specifications needed to run the workload. Aligning services with the specific needs of a workload helps prevent overprovisioning or underutilizing resources.

Evaluate the needs and characteristics of your workload to determine the requirements, and align your workload requirements to your performance targets at every tier. You must account for constraints or dependencies. When you understand your workload requirements, you can make informed decisions. You can determine the right infrastructure and implement strategies to handle peak loads or variations in demand.

- *Meet performance targets.* Select services that enable you to meet the performance targets for your workload. Ensure that a service can support the performance needs and that you can monitor its performance. Collect performance data for critical components.
- *Consider organizational restrictions.* Be familiar with restrictions that your organization might have on services that you deploy. Consider these restrictions

when you design your solution.

- *Consider compliance and security requirements.* Compliance and security requirements can affect services and configurations that you select. Ensure that a service you choose meets the requirements that are related to storage, encryption, access controls, audit logs, and data locations.

- *Consider team skills.* Your team builds and maintains workloads. Different services require different skills. Choose services that your team knows how to use, or commit to training them before you choose a service. Ensure that team members possess the expertise and knowledge to effectively use services and to optimize their performance.

>  **Tradeoff**: Specialized services offer specific functionalities but might limit customization. Flexible resources require more management and configuration compared to specialized services. Managed services offer ease of management, but you might have less control over the underlying infrastructure compared to self-managed resources.

Understand services

Understanding services is about knowing the capabilities, limits, and functionalities of a vendor's tools and offerings. An understanding of services helps you use built-in features, reducing the need for complex custom solutions and improving performance efficiency.

Consider various factors and gain a comprehensive understanding of a service before you choose it. Research and assess services and tools that the provider offers. Determine which services and tools best align with your workload requirements. Consider factors like managed services, serverless options, and specialized services.

Understand service limits

Service limits are the predefined thresholds or boundaries that service providers set. Service limits define the maximum usage of resources or capabilities within that service. When you're familiar with service limits, you can avoid issues such as resource contention, performance degradation, or unexpected service interruptions. You can plan and scale the infrastructure appropriately. Your planning takes into account factors such as data volume, processing capacity, and data residency requirements.

Prefer platform features

Preferring platform features is about using built-in functionalities provided by a provider to handle specific tasks without custom code. Vendors design platform features to handle specific tasks efficiently at scale, and they regularly maintain these features. Platform features allow you to better take advantage of cloud infrastructure capabilities. Choose services that allow you to offload functionality to the platform instead of writing and maintaining your own custom code. In many cases, platform-as-a-service (PaaS) solutions provide better performance efficiency than custom code. Custom code adds complexity and makes the workload prone to performance issues. Only develop custom code when service features aren't sufficient.

>  **Tradeoff**: The best service for your workload might be a technology that your team isn't skilled at, can't afford, or it might require extra security layers. For example, a public load balancer might fit your performance needs. But if you don't have a web application firewall, you might have to deploy a firewall to secure the workload.

Evaluate infrastructure requirements

The performance efficiency of resources is tied to the infrastructure they reside on. It makes the selection of the right infrastructure critical to service performance efficiency. Evaluating infrastructure requirements means to identify the geographical region and availability zones best suited to support your workload. Key considerations in this decision-making include:

- *Understand regions and availability zones.* Every region corresponds to a distinct geographic location. Availability zones represent individual physical datacenters within a given region.

- *Understand available features.* Different regions have different available features, such as the number of services and availability zones. Understand the features that are available in a region before you select it. Ensure that a region meets your workload performance needs.

- *Consider latency.* Latency, the time data takes to travel from source to destination, increases the further services are from each other. Services communicating across regions or availability zones can face increased latency. Identifying services that frequently communicate and positioning them within the same region is recommended. Additionally, selecting a region proximate to your primary user base can minimize latency, offering a better user experience.

- *Understand datacenter mapping.* Availability zones might not map consistently to the same datacenters across different subscriptions. For instance, 'Zone 1' in

'Subscription A' might be different from 'Zone 1' in 'Subscription B'. When operating with multiple subscriptions, you should know these mappings to select zones that bolster performance optimally.

Evaluate networking requirements

Assess your network needs to determine the appropriate workload services and configurations. Ensure that the network can support your workload. To evaluate networking requirements, consider:

- *Understand network traffic.* Assess the expected network traffic for the workload. Understand the data transfer needs and the frequency of network requests.
- *Understand bandwidth requirements.* Determine the bandwidth requirements for the workload. Consider the amount of data transmitted and received over the network.
- *Understand network Latency.* Evaluate the desired latency for the workload. Use private virtual networks and backbone networks instead of traversing the public internet. This technique decreases the latency of the workload.
- *Understand throughput.* Consider the required throughput for the workload. Throughput refers to the amount of data that can be transmitted over a network in a given time. Configure the network routing options to take advantage of network throughput benefits.

>  **Tradeoff**: Private virtual networking limits public access and makes it difficult to deploy and manage resources.

Evaluate compute requirements

Evaluating compute requirements involves assessing the specific compute needs of a workload, including factors such as instance type, scalability, and containerization. Different compute services have varying capabilities and characteristics that can affect the performance of your workload. Select the optimal compute service to ensure that your workload runs efficiently. Consider the following strategies:

- *Understand instance types.* Different instance types are optimized for different workloads, such as CPU-optimized, memory-optimized, and GPU instances. Choose the instance type that aligns with your needs.

- *Consider automatic scaling*. If your workload has variable demand, consider a compute service with an autoscale feature that can automatically adjust the compute capacity based on demand. Automatically scaling helps ensure that you have enough resources during peak times and prevents overprovisioning during low demand periods.

- *Consider containerization*. Containers provide performance advantages compared to a noncontainerized workload. Consider using containerization if it suits your architectural needs. Containers improve compute performance through isolation, resource efficiency, fast startup time, and portability.

  When you use containers, consider design factors such as containerizing all application components. Use Linux-based container runtimes for lightweight images. Give containers short lifecycles to make them immutable and replaceable. Gather relevant logs and metrics from containers, container hosts, and the underlying cluster. Use this data to monitor and analyze performance. Containers are just one component of an overall architecture. Choose an appropriate container orchestrator, like Kubernetes, to further enhance performance and scalability.

Container benefit	**Description**
Isolation	Containers provide isolated environments for applications. Containers ensure that application resources don't interfere with each other. This isolation ensures compute resources assigned to a container are dedicated to running a specific application, resulting in better performance.
Resource efficiency	Containers are lightweight and share the host operating system's kernel, which allows for efficient resource utilization. Multiple containers can run on the same virtualized infrastructure, which maximizes the use of compute resources.
Fast startup time	Container images are prebuilt and are quickly started when needed. This fast startup time enables rapid scalability. It allows applications to scale up or down based on demand and avoid performance bottlenecks.
Portability	Containers encapsulate all the required dependencies and libraries within the image. With containers, it's easier to move applications across different operating systems or environments. This portability enables flexibility in deploying applications and allows for easy migration between cloud providers or on-premises environments.

- *Choose the appropriate tier*. Within each compute service, you can set the compute capacity, select features, and enable capabilities. Based on your performance targets, choose the appropriate service tier for your compute service.

- *Determine the instance count.* Determine the minimum instance count that your workload requires. Some workloads, even at minimal load, might require more than one instance of a compute resource. Set the minimum instance count accordingly.

Evaluate load balancing requirements

Load balancing ensures that network traffic is distributed evenly and prevents any single server from being overwhelmed with requests. Load balancing helps prevent bottlenecks and reduce response times. Evaluate the different load balancing services that your cloud provider offers. Review the cloud provider's documentation and comparison tools to understand the features. Select the most suitable service for your workload. To select a load balancing service, consider:

- *Understand traffic type*: Determine whether the load balancing service needs to handle web traffic, like HTTP and HTTPS, or other protocols, such as Transmission Control Protocol (TCP) or User Datagram Protocol (UDP).
- *Know global or regional routing*: Determine whether your workload requires load balancing within a specific region or across multiple regions.
- *Know service-level objectives (SLOs)*: Consider the service-level agreement (SLA). Different load balancing services offer different levels of performance.
- *Understand features*: Consider load balancing services that provide site acceleration, optimal traffic distribution, and low-latency layer-4 load balancing.

Evaluate data store requirements

Evaluating data store requirements is about assessing the specific needs and conditions for storing, retrieving, and managing data. This assessment considers factors like data volume, access speed, consistency, and durability. A workload might require multiple types of data stores based on varying business and technical requirements. Identifying the right data store services and proper implementation helps prevent bottlenecks and ensures quick data access.

Evaluate database requirements

The database can affect factors such as data storage and retrieval, transaction processing, consistency guarantees, and handling of large or rapidly changing data. Assess the needs and criteria for your database. Select a database system that can meet those requirements. Evaluate the database requirements before you choose a database.

To evaluate the database requirements and choose the appropriate database, follow these steps:

- *Identify the workload needs.* Understand the specific requirements of your workload, such as data volume, expected transaction rates, concurrency, data types, and expected growth. Evaluate different database systems based on your workload needs. For example, if your workload requires high-performance real-time data processing, you might choose a database system optimized for fast data ingestion and low latency.

- *Consider the data model.* Determine the data model that best suits your workload. Evaluate the database requirements to ensure that the chosen database supports the required data structures, relationships, and integrity constraints. For example, if your data has a highly relational structure, you might opt for a relational database management system (RDBMS) that provides robust support for transactions and referential integrity. The data model might be hierarchical, network, relational, object-oriented, or NoSQL. Assess the complexity of your data model. Ensure that the chosen database supports the required data structures and relationships.

- *Evaluate the capabilities.* Consider factors such as read/write patterns, query complexity, latency requirements, and scalability needs. Evaluate the performance capabilities of different database systems accordingly. Some databases excel in read-heavy workloads, while others are optimized for write-intensive or analytical workloads.

- *Assess the load.* Consider factors such as data volume, transaction rates, read/write ratios, and expected growth. Choose a database that can handle the anticipated workload to ensure smooth operation and prevent performance bottlenecks as your workload is scaled. Consider the scalability requirements of your workload. These requirements include anticipated data growth, concurrent user access, and the need for horizontal or vertical scaling. Evaluate the scalability options and availability features that different database systems provide.

Evaluate storage requirements

Choose storage services that align with your data access patterns, durability requirements, and performance needs. Most cloud workloads use a combination of storage technologies. This technique is known as the polyglot persistence approach. Determine the appropriate combination of storage services for your workload. You might also want to separate data to avoid contamination. For example, you might have separate storage accounts for monitoring data and business data. Choosing the right mix and correct implementation is important for optimizing application performance.

Evaluate cache requirements

A cache stores frequently accessed data. Caching reduces data access latency and lowers the load on data storage components. It allows the workload to handle more requests without scaling. It's common to cache workload data and static content. A Redis cache can store session data, database results, API responses, and reference data, such as configuration settings. A content delivery network or static web app can cache and serve static content. Consider caching data to improve your workload performance. Choose the right caching option for your workload, preferring the platform caching services, such as Azure Redis Cache, over custom or self-hosted ones.

Azure facilitation

Understanding requirements: Use Azure Monitor to collect and analyze data from your workload. Monitor provides insights into the performance and health of your workloads, allowing you to identify and troubleshoot issues.

Understanding and evaluating services: Review Azure services and products to determine if they meet your performance requirements. Azure offers several services that accomplish the same outcome. You have the flexibility to align your choice of service to your performance needs, team skill set, and cost requirements.

For a list of the most common Azure limits, see Azure subscription and service limits, quotas, and constraints.

 The Query limits and quotas sample shows how to query the limits and quotas for commonly used resources.

Azure has many services that can accommodate any workload. Review the selection guidance for each service type to help you streamline your selection based on your requirements. See the following guides to choose:

- A region
- Compute services
- Container services
- Data store services
- Load balancing services
- Storage services

Related links

- Azure regions with availability zone support
- Recommendations for defining performance targets
- Recommendations for using availability zones and regions
- What are availability zones?

Performance Efficiency checklist

Refer to the complete set of recommendations.

Performance Efficiency checklist

Recommendations for collecting performance data

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Performance Efficiency checklist recommendation:

> **PE:04** **Collect performance data. Workload components and flows should provide automatic, continuous, and meaningful metrics and logs. Collect data at different levels of the workload, such as the application, platform, data, and operating system levels.**

Collecting performance data is the process of gathering metrics and logs that provide information about the performance of a workload. This data includes numerical values, which are known as *metrics*. Metrics describe the state of the system at a particular point in time. It also includes logs that contain different types of data organized into records.

By collecting performance data, you can monitor and analyze the performance of a workload. You can use this information to identify performance bottlenecks, to troubleshoot issues, to optimize resource allocation, and to make data-driven decisions to improve the overall performance efficiency of the workload.

Without data-driven insights, you might be unaware of underlying performance issues or opportunities for optimization. Potential results include slower response times, decreased throughput, increased resource usage, and ultimately, a suboptimal user experience. Additionally, the lack of performance data makes it difficult to diagnose and troubleshoot issues in a timely manner, leading to prolonged downtime and reduced productivity.

Definitions

Term	Definition
Activity logs	Logs that track management operations on resources, such as deleting a resource.
Application logs	Logs that track information about application events, errors, and other activities, such use sign-ins and database connection failures.
Application performance monitoring (APM) tool	A tool that monitors and reports the performance of an application.
Code instrumentation	The direct or indirect capture of performance metrics from the perspective of the application code. Captured metrics include flow

Term	Definition
	metrics, resource use, and metrics specific to the language or runtime.
Distributed tracing	Gathering and correlating metrics across distributed workload components.
Metrics sink	A storage destination for your metrics that correlates time series data for analysis.
Platform logs	Diagnostic and auditing data that includes resource logs, activity logs, and audit logs.
Platform metrics	Numerical values that record workload performance at a particular time.
Resource logs	Data that a system generates. It provides information about the state of the system.
Rx/Tx errors	The number of receive errors and transmit errors on a network interface.
Structured logging	Defining a meaningful format to log messages, typically as key-value pairs.

Key design strategies

Performance optimization requires data to measure the current performance of a workload or a flow against its performance targets. You need to collect the right amount and diversity of data to measure the performance of the code and the infrastructure against performance targets. Ensure that every component and flow within the workload automatically generates continuous and meaningful metrics and logs. You need to source this data from diverse levels like the application, platform, storage, and operating system. Comprehensive performance data collection allows for a holistic understanding of performance, enabling precise identification of inefficiencies and avenues for improvement.

Centralize performance data

Centralizing performance metrics and logs is the process of collecting performance metrics and logs from various sources and storing them in a central location. Create a central metrics sink and a central log sink. This centralization allows for easy access, analysis, and monitoring of performance metrics and logs across different systems and components. By centralizing metrics and logs, you gain visibility into the performance of

your workload. Choose a suitable platform or tool that can aggregate and store workload performance metrics and logs.

>  **Tradeoff**: Understand the cost of collecting metrics and logs. In general, the more metrics and logs you collect, the higher the cost.

Segment performance data

Segmenting performance data involves organizing and categorizing metrics and logs based on their origin, purpose, or environment. For example, you should separate production data from nonproduction data or distinguish between performance targets and business metrics. Segmenting data helps with optimizing specific environments, facilitates troubleshooting, and limits inaccuracies in performance monitoring. By maintaining a clear distinction between different data types, you can capture, analyze, and respond to relevant metrics more efficiently and better align workload health with workload objectives. To segment performance data, consider the following recommendations:

- *Keep production data and nonproduction data separate*. By separating data by environment, you can ensure focused monitoring and optimization of each environment. In production environments, you can better identify and address performance issues that directly affect users and business operations. In nonproduction environments, the data separation facilitates effective troubleshooting and fine-tuning during the testing phase before you deploy to production.

- *Use one set of data within each environment*. Don't use one set of data for performance targets and another set of data for alerts related to the performance targets. Using different sets of data leads to inaccurate alerts that undermine the effectiveness of performance monitoring.

- *Separate performance targets and business metrics*. The operations and development teams use performance targets to monitor workload health and meet business targets. Business metrics relate to business goals or customer reporting. Capture business metrics in a separate data stream, even if the data directly overlaps. The separation gives you flexibility to capture the right data and independently analyze the data.

Define retention policies

Retention policies dictate how long performance data should be kept. Establishing these policies helps manage storage efficiently and ensures only necessary data is accessible for analysis. Such policies support better performance and meet compliance standards. You should configure retention policies for the log and metrics data to enable effective troubleshooting and monitoring in all environments. For example, the logs and metrics might need to be kept for longer time in a production environment than in the testing environment. The retention period should match your organization's requirements and compliance regulations. Decide how long to retain the data for analysis and audit purposes. Archive the data that you don't need for immediate analysis.

Collect application performance data

Collecting application data involves monitoring and analyzing an application's performance metrics, such as throughput, latency, and completion times, primarily gathered through instrumenting code. Application performance data provides valuable insights into the health and performance of an application. By monitoring and analyzing performance data, you can identify and troubleshoot issues, optimize application performance, and make informed decisions for your application.

Instrument code

Instrumentation refers to the process of embedding code snippets or integrating tools into an application code. The purpose of instrumentation is to capture performance data while the application runs. It's essential to gather metrics that highlight the application's critical operations. Focus on metrics like throughput, latency, and completion time. It's important to differentiate between business-related operations and operations that aren't. For data pertaining to business operations, make sure its metadata is structured in a way that allows distinct tracking and storage. The primary reason for code instrumentation is to collect data on how the application handles its workload. It provides the following benefits:

- *Identifying performance bottlenecks:* By tracking metrics such as CPU use and memory use, you can identify bottlenecks and optimize the code accordingly.
- *Evaluating system behavior under a load:* You can see how the application performs under different workloads and stress scenarios. This data can help you identify issues related to scalability, concurrency, and resource use.
- *Tracking application health and availability:* Because key performance indicators are monitored in real time, you can get alerts about potential issues that affect the application's performance and availability.

- *Improve user experience:* You can gain insights into how users interact with the application. Use this information to optimize the user experience and identify areas for improvement.

- *Plan capacity and allocate resources:* The performance data that instrumentation gathers can provide valuable insights into the resource requirements of an application. This information can inform your decisions about planning capacity and allocating resources.

When you instrument code for performance monitoring, consider the following strategies:

- *Use APM tools*: APM tools can collect and analyze performance data, including metrics, traces, and logs. APM tools offer features like code-level instrumentation, transaction tracing, and performance profiling.

- *Use logging and tracing frameworks*: Logging and tracing frameworks are tools or libraries that developers integrate into their applications to facilitate logging and tracing. These frameworks provide functions to generate logs, trace requests, and sometimes even format or transport the generated data. By incorporating logging and tracing frameworks into the code base, developers can capture relevant data during runtime. The data can include information about the running path, I/O, and performance.

- *Custom instrumentation*: Developers can add custom code to collect performance metrics that are unique to their application and workload. The custom instrumentation can measure runtimes, track resource usage, or capture specific events. Write custom code instrumentation only when platform metrics are insufficient. In some situations, the platform resource can measure aggregate or even granular perspectives of your application. Weigh the question of whether to duplicate that effort by using custom code against excess code tradeoffs or dependency on a platform feature.

- *Capture transaction times.* Capturing transaction times relates to measuring the end-to-end times for key technical functions as a part of performance monitoring. Application-level metrics should include end-to-end transaction times. These transaction times should cover key technical functions such as database queries, response times for external API calls, and failure rates of processing steps.

- *Use telemetry standards.* Consider using APM tool instrumentation libraries and tools that are built around a telemetry standard, such as OpenTelemetry.

Enable distributed tracing

Distributed tracing is a technique used to track and monitor requests as they flow through a distributed system. It allows you to trace the path of a request as it travels across multiple services and components, providing valuable insights into the performance and efficiency of your workload. Distributed tracing is important for performance efficiency because it helps identify bottlenecks, latency issues, and areas for optimization within a distributed system. You can pinpoint where delays or inefficiencies occur and take appropriate actions to improve performance by visualizing the flow of a request. Follow these steps to enable distributed tracing:

1. Start by instrumenting your applications and services to generate trace data. Use libraries or frameworks that support distributed tracing, such as OpenTelemetry.

2. Ensure that trace information is propagated across service boundaries. You should typically pass a unique trace ID and other contextual information with each request.

3. Set up a centralized trace collection system. This system collects and stores the trace data generated by your applications and services.

4. Use the trace data collected to visualize the end-to-end flow of requests and analyze the performance characteristics of your distributed system.

Collect application logs

When you instrument code, one of the primary outputs should be application logs. Logging helps you understand how the application runs in various environments. Application logs record the conditions that produce application events. Collect application logs across all application environments. Corresponding log entries across the application should capture a correlation ID for their respective transactions. The correlation ID should correlate application log events across critical application flows such as user sign-in. Use this correlation to assess the health of key scenarios in the context of targets and nonfunctional requirements.

You should use structured logging. Structured logging speeds up log parsing and analysis. It makes the logs easier to index, query, and report without complexity. Add and use a structured logging library in your application code. Sometimes log entries can help you correlate data that you couldn't correlate by other means.

Collect resource performance data

By collecting resource performance data, you can gain insights into the health and behavior of your workload. Resource performance data provides information about

resource use, which is key for capacity planning. This data also provides insights into the health of a workload and can help you detect issues and troubleshoot. Consider the following recommendations:

- *Collect metrics and logs for every resource.* Each Azure service has a set of metrics that's unique to the functionality of the resource. These metrics help you understand the resource's health and performance. Add a diagnostic setting for each resource to send metrics to a location that your workload team can access as they build alerts and dashboards. Metric data is available for short-term access. For long-term access or for access from a system that's outside of Azure Monitor, send the metric data to your unified sink to the access location.

- *Use platform tooling.* Gather inspiration from built-in and integrated monitoring solutions, such as Azure Monitor Insights. This tooling streamlines performance operations. Consider platform tooling as you select a platform and invest in custom tooling or reporting.

- *Monitor network traffic.* Monitoring network traffic means to track and analyze the flow and patterns of data as it moves across network pathways. Collect traffic analytics and monitor the traffic that traverses subnet boundaries. Your goal is to analyze and optimize network performance.

Collect database and storage data

Many database and storage systems provide their own monitoring tools. These tools collect performance data specific to those systems. Database and storage systems often generate logs that contain performance-related events and indicators. Collect database data and storage performance data so you can identify bottlenecks, diagnose issues, and make informed decisions to improve the overall performance and reliability of your workload. Consider collecting the following types of performance data:

- *Throughput*: Throughput measure the amount of data read from or written to the storage system over a period of time. Throughput data indicates the data transfer capabilities.

- *Latency*: Latency measures how long storage operations last. Latency data indicates the responsiveness of the storage system.

- *IOPS (I/O operations per second)*: Data about the number of read operations or write operations that the storage system can perform in a second. IOPS data indicates the storage system's throughput and responsiveness.

- *Capacity use*: Capacity use is the amount of storage capacity used and the amount that's available. Capacity-use data helps organizations plan for future storage needs.

For databases, you should also collect database-specific metrics:

- *Query performance*: Data about the execution time, resource usage, and efficiency of database queries. Slow or inefficient database queries can significantly slow down a workload. Look for queries that are slow and that run frequently.
- *Transaction performance*: Data about the performance of database transactions, such as transaction duration, concurrency, and lock contention.
- *Index performance*: Data about the performance of database indexes, such as index fragmentation, usage statistics, and query optimization.
- *Resource use*: Data that includes CPU, memory, disk space, I/O, and network bandwidth.
- *Connection metrics*: Metrics that track the number of active, aborted, and failed connections. High failure rates could indicate network issues or could indicate that the database reached its maximum number of connections.
- *Transaction rates*: The number of transactions that a database runs per second. A change in transaction rates can indicate performance issues.
- *Error rates*: Data that indicates a database performance. High error rates might indicate a performance issue. Collect and analyze database errors.

Collect operating system data (if applicable)

A platform as a service (PaaS) solution eliminates the need to collect operating system performance data. However, if your workload runs on virtual machines (infrastructure as a service), you need to collect performance data about the operating system. You need to understand the demand on your operating system and virtual machine. Frequently sample operating system performance counters. For example, you could sample the performance counters every minute.

At a minimum, collect data about the following performance areas.

Performance area	Process or function
CPU	- CPU usage (user mode or privileged mode) - CPU queue length (number of processes that are waiting for CPU time)

Performance area	Process or function
Process	- Process thread count - Process handle count
Memory	- Committed memory - Available memory - Pages per second - Swap space usage
Disk	- Disk read - Disk writes - Disk throughput - Disk space usage
Network	- Network interface throughput - Network interface Rx/Tx errors

Validate and analyze data

Your performance data should align with the performance targets. The data needs to represent workload or flow performance completely and accurately as it relates to performance targets. For example, the response time for a web service has a performance target of 500 ms. Make it a routine to analyze the data, as frequent evaluations allow for early detection and mitigation of performance issues.

- *Create alerts.* It's beneficial to have alerts that are actionable, enabling prompt identification and rectification of performance problems. These alerts should clearly indicate the breached performance threshold, the potential business effect, and the involved components. Start by setting common and recommended alert. Over time, you can modify these criteria based on your specific needs. The primary objective of these alerts should be to forecast potential performance drops before they escalate into significant issues. If you can't set an alert for an external dependency, consider devising a method to gather indirect measurements, like the duration of a dependency call.

- *Set data collection limits.* Determine and set logical limits on the volume of data you collect and its retention duration. Telemetry can sometimes produce overwhelming amounts of data. It's essential to focus on capturing only the most vital performance indicators or have an efficient system in place to extract meaningful insights from your performance data.

Azure facilitation

Centralizing, segmenting, and retaining performance data: Azure Monitor collects and aggregates data from every layer and component of your workload across multiple Azure and non-Azure subscriptions and tenants. It stores the data in a common data platform for consumption by a common set of tools that can correlate, analyze, visualize, and/or respond to the data.

You need at least one Log Analytics workspace to enable Azure Monitor Logs. You can use a single workspace for all your data collection. You can also create multiple workspaces based on requirements to segment performance data. It also allows you to define retention policies.

Collecting application performance data: Application Insights is a feature of Azure Monitor that helps you monitor the performance and availability of your application. It provides application-level insights by collecting telemetry data such as request rates, response times, and exception details. You can enable Application Insights for your application and configure it to collect the necessary performance data. Application Insights also supports distributed tracing. Configure distributed tracing for all flows. To build end-to-end transaction flows, correlate events that come from different application components or tiers.

Performance counters are a powerful way to monitor the performance of your application. Azure provides various performance counters that you can use to collect data about CPU usage, memory usage, disk I/O, network traffic, and more. If you configure your application to emit performance counter data, Azure Monitor collects and stores the data for analysis.

Collecting resource performance data: Most Azure services generate platform logs and metrics that provide diagnostic and auditing information. By enabling diagnostic settings, you can specify the platform logs and metrics to collect and store. For correlation purposes, enable diagnostics for all supported services and then send the logs to the same destination as your application logs.

Collecting database and storage performance data: Azure Monitor allows you to collect performance data for databases in Azure. You can enable monitoring for Azure SQL Database, Azure Database for MySQL, Azure Database for PostgreSQL, and other database services. Azure Monitor provides metrics and logs for monitoring database performance, including CPU use, memory use, and query performance. To be notified of issues, you can set up alerts based on performance thresholds.

Azure offers performance recommendations for databases, such as SQL Server on Azure Virtual Machines. These recommendations help you optimize the performance of your database workloads. They include suggestions for collecting performance counters, capturing wait statistics, and gathering performance data during peak hours.

Azure Storage Analytics allows you to collect performance data for Azure Storage services like Blob Storage, Table Storage, and Queue Storage. You can enable logging and metrics for your storage accounts to monitor key performance indicators, such as the number of read/write operations, throughput, and latency.

Collecting operating system performance data: The Azure Diagnostics extension enables you to collect detailed performance data from your virtual machines (VMs), including CPU, memory, disk I/O, and network traffic. This data can be sent to Azure Monitor or other storage services for analysis and alerting.

Validating and analyzing performance data: Within Azure Monitor, you can use Azure Monitor Logs to collect, analyze, and visualize log data from your applications and systems. You can configure Azure Monitor Logs to ingest logs from your application, including application-level logs and infrastructure logs. By aggregating logs, you can cross-query events and gain insights into the performance of your application. For more information, see Azure Monitor Logs cost calculations and options and Pricing for Azure Monitor.

In Azure Monitor, you can define alert rules to monitor specific performance metrics and trigger alerts based on predefined conditions. For example, you can create an alert rule to notify you when CPU usage exceeds a certain threshold or when response time goes above a specified limit. Configure the alert rule to send notifications to the desired recipients.

When you create an alert rule, you can define the criteria that determine when an alert should be triggered. You can set thresholds, aggregation methods, time windows, and the frequency of evaluation. Define the criteria based on your performance monitoring requirements. In addition to sending notifications, you can specify actions to be taken when an alert is triggered. Actions can include sending emails, calling webhooks, or running Azure functions. Choose the appropriate actions to respond to the specific alert scenario.

Examples

- Baseline highly available zone-redundant app services web application
- Monitor a microservices application in Azure Kubernetes Service (AKS)
- Enterprise monitoring with Azure Monitor

Related links

- Platform metrics

- Diagnostic settings
- Audit logs

Performance Efficiency checklist

Refer to the complete set of recommendations.

Performance Efficiency checklist

Recommendations for optimizing scaling and partitioning

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Performance Efficiency checklist recommendation:

> **PE:05** **Optimize scaling and partitioning. Incorporate reliable and controlled scaling and partitioning. The scale unit design of the workload is the basis of the scaling and partitioning strategy.**

This guide describes the recommendations for scaling and partitioning a workload. Scaling is the ability to increase or decrease the resources allocated to a workload based on demand. Partitioning involves dividing the workload into smaller, manageable units to distribute data and processing across multiple resources. A workload that doesn't scale or partition might experience poor performance in high-demand periods and underutilized capacity in low-demand periods.

Definitions

Term	Definition
Autoscale	A feature that automatically adjusts the capacity limits of a service based on predefined configurations, allowing it to scale up or down as needed.
Capacity	The upper limit or maximum capacity of a given service or feature.
Client affinity (session affinity)	The intentional routing of requests from a single client to a single server instance to help ensure consistent session management.
Consistency (distributed database)	The uniformity of data across multiple nodes in a distributed database, ensuring that all replicas have the same data at a given point in time.
Consistency (relational database)	The property of a transaction bringing a database from one valid state to another, maintaining data integrity.
Consistency level	A configuration that defines how and when data is replicated in a distributed database system, determining the tradeoff between consistency and performance.
Data locking	A mechanism used to prevent simultaneous updates to the same data.
Horizontal scaling	A scaling approach that adds instances of a given type of resource.

Term	Definition
Optimistic concurrency	An approach for updating databases that uses snapshots to make updates instead of traditional locking mechanisms.
Partitioning	The process of physically dividing data into separate data stores.
Scalability	The ability of a workload to dynamically change its capacity limits to accommodate varying levels of demand.
Scale unit	A group of resources that scale proportionately together.
State affinity	The storage of client session data on a single server so that the same server handles subsequent requests from the same client.
Vertical scaling	A scaling approach that adds compute capacity to existing resources.

Key design strategies

Both scaling and partitioning contribute to performance efficiency by ensuring that resources are used effectively and the workload can handle varying loads. These practices are especially important in cloud environments where applications need to be flexible and adaptable to changing demands. Scaling ensures you can expand workload capacity to meet increasing demands. Partitioning allows you to divide tasks or data efficiently to handle these growing needs. The foundation of both these processes is the scale unit design of the workload. It dictates how your workload should grow and distribute tasks. By incorporating a reliable and controlled approach to scaling and partitioning, you can sidestep potential workload inefficiencies.

Optimize scaling

Optimize scaling is the process of adjusting the number of servers, instances, or resources to meet the fluctuating demands of a workload. It ensures that the workload can handle increased traffic or demands without experiencing performance degradation or downtime.

Choose a scaling strategy

Choosing a scaling strategy involves deciding between vertical or horizontal methods to enhance the capacity of a workload based on its specific requirements. Selecting the right strategy ensures that resources are adjusted efficiently to meet workload demands without overuse or waste. To choose the right scaling strategy, you need to understand

the uses cases for vertical and horizontal scaling and how they meet the needs of your workload.

Understand vertical scaling. Using vertical scaling, you can increase the capacity of a single resource, such as upgrading to a larger server or instance size. Vertical scaling is useful when the workload can benefit from increased processing power, memory, or other resources within a single instance. Vertical scaling is appropriate for workloads that aren't easily divided into smaller parts or when the application architecture doesn't support horizontal scaling.

Understand horizontal scaling. Using horizontal scaling, you can add more instances or resources to distribute the workload across multiple servers. Horizontal scaling offers benefits such as improved resiliency, increased capacity, and the ability to handle increased traffic or workload demands. It's effective for cloud-native applications designed to run on multiple nodes. Horizontal scaling is appropriate for workloads that can be divided into smaller parts that run independently.

Understand the workload. The suitability of vertical or horizontal scaling depends on the specific characteristics and requirements of the workload. Regular performance monitoring and testing in the following areas can help optimize the scaling strategy over time:

- *Requirements*: Understand the specific requirements of the workload by considering factors such as resource demands, scalability needs, and the limitations of the workload.
- *Scale units*: Create a scale unit design for components expected to be scaled together. For example, 100 virtual machines might require two queues and three storage accounts to handle the extra workload. The scale unit would be 100 virtual machines, two queues, and three storage accounts. You should independently scale all the components that experience capacity-use fluctuation.
- *Architecture*: Assess the design of the application architecture. Some applications might be inherently designed to scale horizontally, with stateless components that can be easily distributed across multiple instances. Other applications might have stateful components or dependencies that make vertical scaling more appropriate. Evaluate the scalability and elasticity requirements of the workload.

Design infrastructure to scale

Designing infrastructure to scale is the process of creating an architecture that can handle increasing demands and workload by adding or adjusting resources as needed. It involves planning and implementing solutions that can scale horizontally or vertically to

accommodate growth. Strategies include avoiding singletons that can become bottlenecks and decoupling application components to ensure independent scalability. When you design a workload to be scalable, it can effectively distribute the workload across multiple resources, which prevents bottlenecks and maximizes resource utilization.

Avoid singletons. You should avoid the use of a single, centralized resource for the entire workload. Instead, distribute your workload across multiple resources for better scalability, fault tolerance, and performance. Explore some specific examples and design considerations to avoid singletons in workload resources:

- *Queue-based load leveling*: Instead of relying on a single queue to process messages, consider partitioning the workload across multiple queues to distribute the processing load. It provides better scalability and parallel processing.
- *Data processing*: Singleton patterns often appear in data processing scenarios where the processing doesn't fan out. Break long-running tasks into smaller tasks that can scale better to distribute the workload across multiple resources and take advantage of parallelism.
- *Design patterns*: Design patterns such as Fan-out/Fan-in or Pipes and Filters can help avoid singletons in workflows. These patterns enable the distribution of processing tasks across multiple resources and promote scalability and flexibility.

Decouple components. Decoupling application components is an important aspect of designing for scalability. It involves breaking down the application into smaller, independent components that can operate and scale independently based on specific workload requirements. For example, if one component requires more resources due to increased demand, you can scale that component without affecting the others. This flexibility ensures efficient resource allocation and prevents bottlenecks. By decoupling components, you can isolate failures and minimize the effect on the overall application. If one component fails, the other components can continue to function independently.

Decoupled components are easier to maintain and update. Changes or updates to one component can be made without affecting the others because they're independent. Follow these guidelines to decouple application components for scalability:

- *Separation of concerns*: Identify the responsibilities and functionalities of your application. Divide the responsibilities into separate components based on their specific tasks. For example, you might have separate components for user authentication, data processing, and UI.

- *Loose coupling*: Design the components to communicate with each other through well-defined interfaces and protocols. This design reduces dependencies between components and allows for easier replacement or scaling of individual components.

- *Asynchronous communication*: Use asynchronous communication patterns such as message queues or event-driven architectures to decouple components further. These patterns allow components to process tasks independently at their own pace, improving overall scalability.

- *Microservices*: Consider implementing microservices, which are small, independent services that focus on specific business functionalities. Each microservice can be developed, deployed, and scaled independently, providing greater flexibility and scalability.

Design application to scale

As you scale a workload, you should design the application to distribute the load. Just because you can add more replicas at the infrastructure level doesn't mean your application can use the replicas. Designing an application to scale is about structuring an application so it can handle increased demands by distributing its workload across resources. Avoid solutions that require client affinity, data locking, or state affinity for a single instance if possible. You want to route a client or process to a resource that has available capacity. To design for application scalability, consider the following strategies:

Eliminate server-side session state. You should design applications to be stateless where possible. For stateful applications, you should use a state store that's external to your server. Externalizing session state is the practice of storing session data outside of the application server or container. You can externalize session state to distribute session data across multiple servers or services, enabling seamless session management in a distributed environment. Consider the following when externalizing session state:

- *Evaluate your session requirements.* Understand the session data that needs to be stored and managed. Consider session attributes, session timeouts, and any specific requirements for session replication or persistence. Determine the size of your session state and the frequency of read and write operations.

- *Choose a solution.* Select a storage solution that aligns with your performance and scalability needs. Options include using a distributed cache, a database, or a session state service. Consider factors such as data consistency, latency, and scalability when making your choice.

- *Set up your application.* Update your application to use the chosen session state storage solution. You might need to change your application's configuration files or code to connect to the external storage service.

- *Update your logic.* Change your application's session management logic to store and retrieve session data from the external storage solution. You might need to use APIs or libraries provided by the storage solution to manage session state.

Eliminate client affinity. Client affinity is also known as session affinity or sticky sessions. When you eliminate client affinity, you distribute client requests evenly across multiple replicas or servers, without routing all requests from a client to the same replica. This configuration can improve the scalability and performance of applications by allowing any available replica to process the requests.

Review your load balancing algorithm. A load balancing algorithm can cause unintentional and artificial client pinning where too many requests are sent to one back-end instance. Pinning can happen if the algorithm is set up to always send requests from the same user to the same instance. It can also happen if the requests are too similar to each other.

Eliminate data locking. Data locking ensures consistency but has performance disadvantages. It can cause lock escalations and negatively affect concurrency, latency, and availability. To eliminate data locking, you should implement optimistic concurrency. Nonrelational databases should use optimistic concurrency control and have the right consistency level. Your data partitioning strategy should also support your concurrency needs.

Use dynamic service discovery. Dynamic service discovery is the process of automatically detecting and registering services in a distributed system. It allows clients to discover available services without being tightly coupled to specific instances. Clients shouldn't be able to take a direct dependency on a specific instance in the workload. To avoid these dependencies, you should use a proxy to distribute and redistribute client connections. The proxy acts as an intermediary between clients and services, providing a layer of abstraction that allows services to be added or removed without affecting clients.

Use background tasks. When an application is scaled, it can handle an increasing workload or a higher number of concurrent requests. Offloading intensive tasks as background tasks allows the main application to handle user requests without resource-intensive operations overwhelming it. Follow these steps to offload tasks as background tasks:

1. Find the CPU-intensive and I/O-intensive tasks in your application that you can offload. These tasks typically involve heavy computations or interactions with external resources such as databases or network operations.

2. Design your application to support background tasks. Decouple the intensive tasks from the main application logic and provide a mechanism to start and manage background tasks.

3. Implement background task processing with appropriate technologies or frameworks. Include features provided by your programming language or platform, such as asynchronous programming, threading, or task queues. Contain intensive operations in separate tasks or threads, these tasks can be run concurrently or scheduled to run at specific intervals.

4. Distribute background tasks if there are many of them, or if the tasks require substantial time or resources. For one possible solution, see the Competing Consumers pattern.

Configure scaling

Configuring scaling is the process of setting up and adjusting parameters to dynamically allocate resources based on workload demands. It encompasses strategies such as using autoscaling features, understanding service scaling boundaries, and implementing meaningful load metrics. Proper configuration ensures that an application can respond to varying demands while maximizing efficiency. When you configure scaling, consider the following strategies:

Use services with autoscaling. The autoscale feature automatically scales infrastructure to meet demand. Use platform as a service (PaaS) offerings with built-in autoscale features. The ease of scaling on PaaS is a major advantage. For example, scaling out virtual machines requires a separate load balancer, client-request handling, and externally stored state. PaaS offerings handle most of these tasks.

Constrain autoscaling. Set automatic scaling limits to minimize over-scaling that could result in unnecessary costs. Sometimes you can't set scaling limits. In these cases, you should set alerts to notify you when the component reaches the maximum scale limit and over-scaled.

Understand service scaling boundaries. When you understand service scaling limits, increments, and restrictions, you can make informed decisions when selecting a service. Scaling boundaries determine whether or not your chosen service can handle the expected workload, scale efficiently, and meet the performance requirements of your application. Scaling boundaries to consider include:

- *Scaling limits*: Scaling limits are the maximum capacity that a location or service can handle. It's important to know these limits to help ensure that the service can accommodate the expected workload and handle peak usage without performance degradation. Every resource has an upper scale limit. If you need to go beyond scale limits, you should partition your workload.

- *Scaling increments*: Services scale at defined increments. For example, compute services might scale by instances and pods while databases might scale by instances, transaction units, and virtual cores. It's important to understand these increments to optimize resource allocation and prevent resource flapping.

- *Scaling restrictions*: Some services allow you to scale up or out but limit your ability to automatically reverse scaling. You're forced to scale in manually, or you might have to redeploy a new resource. These limitations are often to protect the workload. Scaling down or scaling in can have implications on the availability and performance of the workload. A service might enforce certain limitations or constraints to help ensure that the workload has sufficient resources to operate effectively. These limitations can affect data consistency and synchronization, especially in distributed systems. The service might have mechanisms in place to handle data replication and consistency during scaling up or out but might not provide the same level of support for scaling down or in.

Use meaningful load metrics. Scaling should use meaningful load metrics as scaling triggers. Meaningful load metrics include simple metrics, like CPU or memory. They also include more advanced metrics, such as queue depth, SQL queries, custom metrics queries, and HTTP queue length. Consider using a combination of simple and advanced load metrics as your scaling trigger.

Use a buffer. A buffer is unused capacity that can be used to handle spikes in demand. A well-designed workload plans for unexpected spikes in workload. You should add a buffer to handle spikes for horizontal and vertical scaling.

Prevent flapping. Flapping is a looping condition that occurs when one scale event triggers an opposite scale event, creating a continuous back-and-forth scaling action. For example, if scaling in reduces the number of instances, it might cause the CPU usage to rise in the remaining instances, triggering a scale-out event. The scale-out event, in turn, causes the CPU usage to drop, repeating the process.

It's important to choose an adequate margin between the scale-out and scale-in thresholds to avoid flapping. You can prevent frequent and unnecessary scale-in and scale-out actions by setting thresholds that provide a significant difference in CPU usage.

Use Deployment Stamps. There are techniques that make it easier to scale a workload. You can use the Deployment Stamps pattern to easily scale a workload by adding one or more scale units.

>  **Risk**: While scaling helps optimize costs by adjusting capacity to meet demand, it can result in overall increased cost during long periods of high demand.

Test scaling

Testing scaling involves simulating various workload scenarios in a controlled environment to evaluate how a workload responds to different levels of demand. It helps ensure the workload scales efficiently, maximizing performance efficiency during varied loads.

You need to ensure that your workload scales efficiently under real-world conditions. It's essential to perform load and stress tests in an environment that mirrors your production setup. These tests, conducted in nonproduction environments, enable you to evaluate both vertical and horizontal scaling strategies and determine which one optimizes performance most effectively. Here's a recommended approach to testing scaling:

- *Define workload scenarios.* Identify the key workload scenarios that you need to test, such as increasing user traffic, concurrent requests, data volume, or resource use.
- *Use production-like test environment.* Create a separate testing environment that closely resembles the production environment in terms of infrastructure, configuration, and data.
- *Set performance metrics.* Define the performance metrics to measure, such as response time, throughput, CPU and memory utilization, and error rates.
- *Develop test cases.* Develop test cases that simulate different workload scenarios, gradually increasing the load to assess the performance at various levels.
- *Execute and monitor tests.* Run the tests using the defined test cases and collect performance data at each load level. Monitor workload behavior, resource consumption, and performance degradation.
- *Analyze and optimize scaling.* Analyze the test results to identify performance bottlenecks, scalability limitations, or areas for improvement. Optimize the configuration, infrastructure, or code to enhance scalability and performance. It takes time for scaling to complete, so test the effects of scaling delays.

- *Address dependencies.* Find potential dependency issues. Scaling or partitioning in one area of a workload might cause performance issues on a dependency. The stateful parts of a workload, such as databases, are the most common cause of dependency performance issues. Databases require careful design to scale horizontally. You should consider measures, such as optimistic concurrency or data partitioning, to enable more throughput to the database.

- *Retest after adjustments.* Repeat the scalability tests after implementing optimizations to validate the improvements and help ensure the workload can handle the expected workloads efficiently.

>  **Tradeoff**: Consider the budget constraints and cost-efficiency goals of your workload. Vertical scaling might involve higher costs due to the need for larger and more powerful resources. Horizontal scaling offers cost savings by using smaller instances that can be added or removed based on demand.

Partition workload

Partitioning is the process of dividing a large dataset or workload into smaller, more manageable parts called partitions. Each partition contains a subset of the data or workload and is typically stored or processed separately. Partitioning enables parallel processing and reduces contention. Dividing the workload into smaller units allows the application to process each unit independently. The result is better use of resources and faster processing times. Partitioning also helps distribute the data across multiple storage devices, reducing the load on individual devices and improving overall performance.

Understand partitioning

The specific partitioning approach you use depends on the type of data or workload you have and the technology you're using. Some common strategies for partitioning include:

- *Horizontal partitioning*: In this approach, the dataset or workload is divided based on specific criteria, such as ranges of values or specific attributes. Each partition contains a subset of the data that meets the defined criteria.

- *Vertical partitioning*: In this approach, the dataset or workload is divided based on specific attributes or columns. Each partition contains a subset of the columns or attributes, allowing for more efficient access to the required data.

- *Functional partitioning*: In this approach, the data or workload is divided based on the specific functions or operations that need to be performed. Each partition

contains the data or components necessary for a specific function, enabling optimized processing and performance.

Plan partitioning

It's important to consider factors such as data distribution, query patterns, data growth, and workload requirements when partitioning. Proper planning and design are essential to help ensure the effectiveness of partitioning and maximize performance efficiency. If you address partitioning as an afterthought, it's more challenging because you already have a live workload to maintain. You might need to change data access logic, distribute large quantities of data across partitions, and support continued usage during data distribution.

Implement partitioning

It's important to analyze the characteristics of your data, access patterns, concurrency requirements, and scalability goals when deciding which type of partitioning to use. Each type of partitioning has its own advantages and considerations. Here are some factors to consider for each type of partitioning:

- *Horizontal partitioning* is appropriate when you want to distribute the data across multiple resources or servers for better scalability and performance. It's effective when the workload can be parallelized and processed independently on each partition. Consider horizontal partitioning when multiple users or processes need to be able to access or update the dataset concurrently.
- *Vertical partitioning* is appropriate when certain attributes or columns are frequently accessed, while others are accessed less frequently. Vertical partitioning allows for efficient access to the required data by minimizing unnecessary data retrieval.
- *Functional partitioning* is appropriate when different functions require different subsets of the data and can be processed independently. Functional partitioning can optimize performance by allowing each partition to focus specific operations.

Test and optimize partitioning

Test the partitioning scheme to verify the effectiveness and efficiency of the strategy so you can make adjustments to improve performance. Measure factors such as response time, throughput, and scalability. Compare the results against performance goals and identify any bottlenecks or issues. Based on the analysis, identify potential optimization

opportunities. You might need to redistribute data across partitions, adjust partition sizes, or change the partitioning criteria.

>  **Tradeoff**: Partitioning adds complexity to the design and development of a workload. Partitioning requires conversations and planning between developers and database administrators.

>  **Risk**: Partitioning introduces some potential problems that need to be considered and addressed, including:
>
> - *Data skew*: Partitioning can lead to data skew, where certain partitions receive a disproportionate amount of data or workload compared to others. Data skew can result in performance imbalances and increased contention on specific partitions.
> - *Query performance*: Poorly designed partitioning schemes can negatively affect query performance. If queries need to access data across multiple partitions, it might require extra coordination and communication between partitions, leading to increased latency.

Azure facilitation

Optimizing scaling. Azure has the infrastructure capacity to support vertical and horizontal scaling. Azure services have different performance tiers known as SKUs. SKUs allow you to scale vertically. Many of Azure's resources support automatic scaling or other in-place scale options. Some resources support advanced metrics or custom input to support fine-tuning scaling behavior. Most scaling implementations in Azure can set limits and support the necessary observability to be alerted to change.

Azure Monitor allows you to monitor various metrics and conditions in your applications and infrastructure. You can use Monitor to trigger automated scaling actions based on predefined rules. For example, in Azure Kubernetes Service (AKS), you can use Monitor to enable horizontal pod automatic scaling (HPA) and cluster automatic scaling. Using Monitor's monitoring and alerting capabilities, you can effectively facilitate scaling in Azure and help ensure that your applications and infrastructure can dynamically adjust to meet demand.

You can also build custom automatic scaling in Azure. You can use alerts in Monitor for resources that don't have an autoscale feature. These alerts can be set up to be query-based or metric-based and can perform actions using Azure Automation. Automation provides a platform for hosting and running PowerShell and Python code across Azure,

the cloud, and on-premises environments. It offers features such as deploying runbooks on demand or on a schedule, run history and logging, integrated secrets store, and source control integration.

Designing application to scale: Here are some ways Azure facilitates application scaling design;

- *Eliminating data locking*: In Azure SQL Database, you can enable optimized locking to improve performance on databases that require strict consistency.
- *Using background tasks*: Azure offer services and guidance for implementing background jobs. For more information, see Background jobs.
- *Implementing load balancing*: Azure provides load balancers that don't require client affinity. These load balancers include Azure Front Door, Azure Application Gateway, and Azure Load Balancer.

Partitioning a workload: Azure offers various partitioning strategies for different data stores. These strategies help improve performance and scalability by distributing the data across multiple partitions. For more information, see Data partition strategies.

Related links

- Why partition data?
- Best practices for automatic scaling
- Overview of the autoscale feature in Azure
- Horizontal, vertical, and functional data partitioning
- Application design considerations

Performance Efficiency checklist

Refer to the complete set of recommendations.

Performance Efficiency checklist

Recommendations for performance testing

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Performance Efficiency checklist recommendation:

> **PE:06** **Test performance. Perform regular testing in an environment that matches the production environment. Compare results against the performance targets and the performance benchmark.**

This guide describes the recommendations for testing. Performance testing helps you evaluate the functionality of a workload in various scenarios. It involves testing the workload's response time, throughput, resource utilization, and stability to help ensure that the workload meets its performance requirements.

Testing helps to prevent performance issues. It also helps ensure that your workload meets its service-level agreements. Without performance testing, a workload can experience performance degradations that are often preventable. Workload performance can drift from performance targets and established baselines.

Definitions

Term	Definition
Chaos testing	A performance test that aims to test the resilience and stability of a system by deliberately introducing random and unpredictable failures or disruptions.
Load test	A performance test that measures system performance under typical and heavy load.
Performance baseline	A set of metrics that represent the behavior of a workload under normal conditions as validated by testing.
Stress test	A performance test that overloads a system until it breaks.
Synthetic test	A performance test that simulates user requests in an application.

Key design strategies

Performance testing helps you gather measurable data on a workload. When you run tests early enough, they also help you build workloads to the right specifications. You should conduct performance tests as early as possible in the software development

lifecycle. Early testing allows you to catch and fix performance issues earlier in development. You can use a proof of concept (POC) if production code isn't ready.

Prepare the test

Preparing performance tests refers to setting up and arranging the resources, configurations, and test scenarios that you need to conduct performance testing effectively.

Define acceptance criteria

Acceptance criteria specify the performance requirements that a workload needs to meet to be considered acceptable or successful. Define criteria that align with the performance targets.

Review performance targets. Performance targets define your desired level of performance for your workload. Review the performance targets that are established for the workload. Performance targets are metrics that can involve response time, throughput, resource utilization, or any other relevant performance indicators. For example, you might have a target for your response time to be under a certain threshold, such as less than 2 seconds.

Define acceptance criteria. Translate the performance targets into specific acceptance criteria that you can use to evaluate the performance of your workload. For example, suppose your performance target for response time is 2 seconds or less. Your acceptance criterion could be *The average response time of the workload should be less than 2 seconds*. Use these acceptance criteria to determine whether the workload meets the desired level of performance.

When you define acceptance criteria, it's important to focus on users and their expectations. Acceptance criteria help ensure that the delivered work meets user needs and requirements. Keep in mind the following considerations for incorporating the user perspective into acceptance criteria:

- *User requirements*: Understand the user needs and goals for the workload. Consider how the workload should perform to satisfy these requirements.
- *User experience*: Define acceptance criteria that capture the desired user experience. Include factors such as response time, usability, accessibility, and overall satisfaction.
- *Functional requirements*: Address the specific functionality that the user expects to see in the workload. Define acceptance criteria around these functional

requirements to help ensure that they're met.

- *Use cases*: Consider different scenarios or use cases that the user might encounter. Define acceptance criteria based on these use cases to validate the workload's performance in real-world situations.

Set acceptance thresholds. Determine the thresholds within the acceptance criteria that indicate whether the workload meets the performance targets. These thresholds define the acceptable range of performance for each metric. For example, suppose the acceptance criterion for response time is less than 2 seconds. You can set the threshold at 2.5 seconds. This level indicates that any response time over 2.5 seconds is considered a performance issue.

Define passing criteria. Establish the criteria for determining whether the workload passed or failed the performance test. You might define passing as meeting all the acceptance criteria or achieving a certain percentage of them.

Select the test type

To select the right type of performance test, it's important to align the test with your acceptance criteria. The acceptance criteria define the conditions that need to be met for a requirement or bug fix to be considered done. Performance tests should aim to verify whether a workload meets these acceptance criteria and performs as expected under specified conditions. Aligning the performance test type with the acceptance criteria helps ensure that the test focuses on meeting the performance expectations that the criteria define.

- *Understand acceptance criteria*. Review the acceptance criteria for the requirement or bug fix. The criteria outline the specific conditions and functionalities to be met.
- *Identify relevant performance metrics*. Based on the acceptance criteria, determine the performance metrics that are critical to achieving the desired outcomes. For example, if the acceptance criteria focus on response time, prioritizing load testing might be appropriate.
- *Select an appropriate test type*. Evaluate the available test types and choose the one that best aligns with the identified performance metrics and acceptance criteria.

The following table provides a sample of test types and their use cases.

Test type	Description	Use case
Load testing	Simulate realistic user loads to measure how your workload performs under expected	Determines load tolerance.

Test type	Description	Use case
	peak workloads.	
Stress testing	Push your workload beyond its normal limits to identify its breaking points and measure its ability to recover.	Determines resilience and robustness.
Soak testing (endurance testing)	Run your workload under sustained high loads for an extended period to identify performance degradation, memory leaks, or resource issues.	Evaluates stability and reliability over time.
Spike testing	Simulate sudden increases in user load to assess how your workload handles abrupt changes in demand.	Measures the ability to scale and maintain performance during peak periods.
Compatibility testing	Test your workload's performance across various platforms, browsers, or devices.	Helps ensure consistent performance across various environments.

Prioritize your selected test types based on the characteristics and requirements of your workload. Consider factors such as the criticality of performance metrics, user expectations, business priorities, and known issues or vulnerabilities.

Select testing tools

Choose appropriate tools based on the type of performance testing that you want to run. Evaluate the testing environment's infrastructure, resources, and constraints. Choose testing tools that support the desired test types and provide the necessary features for monitoring, measurement, analysis, and reporting.

An application performance monitoring (APM) tool provides deep insights into applications and is an essential testing tool. It helps you trace individual transactions and map their paths through various workload services. After testing, you should use the APM tool to analyze and compare testing data against your performance baseline.

Use profiling tools to identify performance bottlenecks in your code. Profiling helps identify areas of the code that consume the most resources and need optimization. It provides insights into the execution time and memory usage of different parts of the code.

The following steps can help you select the appropriate testing tools:

- *Identify testing requirements*. Begin by understanding the specific requirements of your performance testing. Consider various factors:

    - The type of workload
    - Performance metrics to measure, such as response time and throughput
    - The complexity of the workload architecture
    - The testing environment, such as cloud-based, on-premises, or hybrid

- *Research testing tools*. Conduct research to identify performance testing tools that align with your requirements. Consider commercial and open-source tools that are available in the market. Look for tools that support your desired types of performance testing, such as load testing or stress testing, and that provide features for measuring performance metrics.

- *Evaluate tool features*. Assess the features that each testing tool provides. Look for capabilities such as simulation of realistic user behavior and scalability to handle large user loads. Consider support for various protocols and technologies, integration with other testing tools or frameworks, and reporting and analysis capabilities.

- *Consider compatibility and integration*. Determine the compatibility of the testing tools with your existing infrastructure and technologies. Ensure that the tools can be easily integrated into your testing environment and can communicate with the necessary workload for monitoring and analysis.

- *Evaluate cost and licensing*. Assess the cost structure and licensing terms that are associated with the testing tools. Consider factors such as the initial investment, maintenance costs, and support costs. Also consider other licensing requirements that depend on the number of users or virtual users.

- *Conduct a POC*. Select a few tools that appear to be the most suitable based on your evaluation. Conduct a small-scale POC to validate the usability, features, and effectiveness of the tools in your specific testing scenario.

- *Consider support and training*. Evaluate the level of support and training that the tool's vendor or community provides. Determine the availability of documentation, tutorials, and technical support channels to assist with any challenges or issues that might arise during the testing process.

Create test scenarios

Creating test scenarios refers to the process of designing specific situations or conditions that are suitable for testing the performance of a workload. Test scenarios are created to emulate realistic user behavior and workload patterns. These scenarios provide a way for performance testers to evaluate how the workload performs under various conditions.

Test scenarios make it possible to replicate various workload patterns, such as concurrent user access, peak load periods, or specific transaction sequences. By testing the workload under different workload patterns, you can identify performance bottlenecks and optimize resource allocation.

- *Define user behavior*. Emulate realistic user behavior and workload patterns by identifying the steps and actions that users perform when they interact with the workload. Consider activities such as signing in, performing searches, submitting forms, or accessing specific features. Break down each scenario into specific steps and actions that represent the user's interaction with the workload. You can include navigating through pages, performing transactions, or interacting with various elements of the workload.

- *Determine data involvement*. Identify the test data required to run the test scenarios. You might include creating or generating realistic data sets that represent various scenarios, user profiles, or data volumes. Ensure that the test data is diverse and covers different use cases to provide a comprehensive performance evaluation.

- *Design test scripts*. Create test scripts that automate the execution of the defined test scenarios. Test scripts typically consist of a sequence of actions, HTTP requests, or interactions with workload APIs or user interfaces. Use performance testing tools or programming languages to write the scripts, considering factors such as parameterization, correlation, and dynamic data handling. Validate the test scripts for correctness and functionality. Debug any issues, such as script errors, missing or incorrect actions, or data-related problems. Test script validation is crucial to help ensure accurate and reliable performance test execution.

- *Configure test variables and parameters*. Configure variables and parameters within test scripts to introduce variability and simulate real-world scenarios. Include parameters such as user credentials, input data, or randomization to mimic different user behaviors and workload responses.

- *Iteratively refine scripts*. Continuously refine and enhance test scripts based on feedback, test results, or changing requirements. Consider optimizing script logic, parameterization, and error handling, or adding extra validation and checkpoints.

Configure the test environment

Configuring a test environment refers to the process of setting up the infrastructure, software, and network configurations that you need to create an environment that closely resembles your production environment.

To set up your testing environment in a way that boosts performance efficiency, include the following steps in your configuration process:

- *Mirror your production environment*. Set up your test environment to closely resemble your production environment. Consider factors such as infrastructure configuration, network settings, and software configurations. The goal is to ensure that the performance test results are representative of real-world conditions.
- *Provision sufficient resources*. Allocate adequate resources such as CPU, memory, and disk space to the test environment. Ensure that the available resources can handle the expected workload and provide accurate performance measurements.
- *Replicate network conditions*. Configure the network settings in the test environment to replicate the expected network conditions during the actual workload deployment. You need to include bandwidth, latency, and network protocols.
- *Install and configure dependencies*. Install the software, libraries, databases, and other dependencies that are required for the workload to run correctly. Configure these dependencies to match the expected production environment.

>  **Tradeoff**: There are costs associated with maintaining separate test environments, storing data, using tooling, and running tests. Know the cost of performance testing, and find a way to optimize spending.

>  **Risk**: Production data can contain sensitive information. Without a robust scrubbing and masking strategy, you risk leaking sensitive data when you use production data for testing.

Perform the tests

Run the performance tests by using the chosen testing tool. Testing involves measuring and recording performance metrics, monitoring health, and capturing any performance issues that arise.

Monitor and collect performance metrics such as response time, throughput, CPU and memory utilization, and other relevant indicators.

Use the defined test scenarios to put the workload under expected loads. Conduct tests under these varying load conditions. For example, use levels, such as normal, peak, and stress levels, to analyze the behavior of the workload in various scenarios.

Analyze the results

Analyzing the test results involves examining the collected data and metrics from the performance tests to gain insights into the performance of the workload. The goal is to identify performance issues and use the feedback to adjust priorities in application development. The following actions are key steps for analyzing test results.

Review performance metrics. Look at the performance metrics that you collect during performance testing, such as response times, throughput, error rates, CPU and memory utilization, and network latency. Analyze these metrics to understand the overall performance of the workload.

- *Identify bottlenecks*. Evaluate the performance metrics to identify any bottlenecks or areas of inefficient performance. The evaluation can include high response times, resource constraints, database issues, network latency, and scalability limitations. Pinpointing the root causes of these bottlenecks helps you prioritize performance improvements.

- *Correlate metrics*. Assess the relationships and correlations between various performance metrics. For example, analyze how increased load or resource utilization affects response times. Understanding these correlations can provide valuable insights into workload behavior under different conditions. Look for patterns and trends in the performance data over time. Analyze performance under different load levels or during specific periods. Detecting trends can help identify seasonal variations, peak usage times, or recurring performance issues.

Evaluate acceptance criteria. Compare the retest results against the predefined acceptance criteria and performance goals. Assess whether the workload meets the desired performance standards. If the workload doesn't meet the acceptance criteria, further investigate and refine the optimizations.

Iterate and refine the analysis. Make other adjustments and improvements as needed. Use the collected data and metrics to diagnose specific performance issues. The diagnosis might involve tracing through the workload components, examining log files, monitoring resource usage, or analyzing error messages. Dig deeper into the data to understand the underlying causes of performance problems.

Based on the analysis of the test results, prioritize identified performance issues and implement necessary improvements. The improvements can involve optimizing code, tuning database queries, improving caching mechanisms, and optimizing network configurations.

Establish baselines

Baselines provide a reference point for comparing performance results over time. Baselines should be meaningful snapshots of workload performance—you don't need to use every test as a baseline.

Consider the workload objectives, and document performance snapshots that allow you to learn over time and optimize. Use these baseline measurements as a benchmark for future performance tests, and use them to identify any degradation or improvement.

To establish baselines for performance testing and use them as a benchmark for future performance tests, follow these steps:

- *Identify performance metrics*. Determine the specific performance metrics that you want to measure and track. Examples include:
  - Response time, or how quickly the workload responds to requests.
  - Throughput, or the number of requests that are processed per unit of time.
  - Resource utilization, such as CPU, memory, and disk usage.
- *Record meaningful measurements*. Record the performance metrics that you obtain during the test as the baseline measurements. These measurements represent the starting point against which you compare future performance tests.
- *Compare future tests*. In subsequent performance tests, compare the performance metrics against the established baselines and thresholds. The comparison allows you to identify any improvements or degradation in performance.

Test continuously

Continuous testing involves the ongoing monitoring and refinement of your tests. Continuous testing helps you maintain consistent and acceptable levels of performance. A workload should provide a consistent and acceptable level of performance relative to the baseline. You should tune the workload over time to produce consistent performance that's within the acceptable limits of performance. Here are some key practices:

- *Set degradation limits*. Define numeric thresholds that specify the level of performance degradation that's acceptable over time. By setting these limits, you can monitor performance fluctuations and receive alerts when the performance falls below the defined threshold.
- *Include quality assurance*. Incorporate performance requirements, such as CPU utilization and maximum requests per second, into the quality assurance process.

Treat performance requirements with the same level of importance as functional requirements. This process helps ensure that the workload meets the defined performance requirements before you deploy it to production.

- *Automate alerting*. In live environments, rapid detection and response are crucial. Set up automated alerting systems that use the performance baseline as their reference. If there's a significant deviation in performance, the necessary teams are alerted immediately to act.

- *Test changes*. Some performance issues might only manifest in a live setting. Apply thorough testing practices for proposed code and infrastructure changes. Use code instrumentation to gain insights into the application's performance characteristics, such as hot paths, memory allocations, and garbage collection. This testing ensures that any change introduced doesn't degrade performance beyond the acceptable limits.

Azure facilitation

Perform the tests: Azure Pipelines makes it possible for you to integrate performance testing into your CI/CD pipeline. You can incorporate load testing as a step in your pipeline to validate the performance and scalability of your applications.

Azure Chaos Studio provides a way for you to inject real-world faults into your application so that you can run controlled fault injection experiments. The experiments help you measure, understand, and improve your cloud application and service resilience.

Azure Load Testing is a load testing service that generates high-scale load on any application. Load Testing provides capabilities for automating load tests and integrating them into your continuous integration and continuous delivery (CI/CD) workflow. You can define test criteria, such as average response time or error thresholds, and automatically stop load tests based on specific error conditions. Load Testing offers a dashboard that provides live updates and detailed resource metrics of Azure application components during a load test. You can analyze the test results, identify performance bottlenecks, and compare multiple test runs to understand performance regressions over time.

Analyzing the results: Azure Monitor is a comprehensive monitoring solution for collecting, analyzing, and responding to telemetry from your cloud and on-premises environments. Application Insights is an extension of Monitor that provides APM features. You can use Application Insights to monitor applications during development and testing and also in production.

> **Tradeoff**: Testing takes time and skill to perform and can affect operational efficiency.

Related links

- Recommendations for security testing
- Recommendations for designing a reliability testing strategy

Performance Efficiency checklist

Refer to the complete set of recommendations.

Performance Efficiency checklist

Recommendations for optimizing code and infrastructure

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Performance Efficiency checklist recommendation:

> **PE:07** **Optimize code and infrastructure. Use code that's performant, and ensure that it offloads responsibilities to the platform. Use code and infrastructure only for their core purpose and only when necessary.**

This guide describes the recommendations for optimizing code and infrastructure performance. To optimize your code and infrastructure, you should use your components only for their core purpose and only when necessary. When you overuse code and infrastructure, it creates unnecessary resource consumption, bottlenecks, and slow responses. To compensate for those inefficiencies, you must add more resources to accomplish the same tasks.

Definitions

Term	Definition
Concurrency	When multiple tasks or processes are performed at once but not necessarily at the exact same time.
CPU architecture	The components and principles that affect how the computer works.
Data compression	The action of reducing the size of files by minimizing redundant data.
Heap	An area in memory used for runtime memory allocations.
Memory leak	When a workload fails to release allocated memory after the memory is no longer needed.
Parallelism	When multiple tasks or processes are performed at the same time.

Key design strategies

Optimizing code and infrastructure entails fine-tuning the code and the supporting infrastructure to improve performance efficiency. It requires performant code that executes tasks quickly and doesn't waste resources. It requires a well-designed infrastructure that is streamlined to avoid unnecessary complexity. A workload should

use the inherent capabilities of the platform. It's an approach that helps ensure both code and infrastructure are used primarily for their core purposes and only when necessary.

Optimize code performance

To optimize code performance, modify code to reduce resource usage, minimize runtime, and enhance performance. You can modify code to improve the efficiency and speed of a software program. Don't mask performance issues with brute force. Brute force means adding compute resources to compensate for code performance, like adding extra capacity instead of addressing the source. You need to fix performance issues with optimization. When you optimize code performance, it helps maximize the utilization of system resources, improves response time, reduces latency, and enhances the user experience.

Instrument your code

Instrumenting code refers to the practice of adding code snippets or libraries to code that collect data and monitor code performance during runtime. Code instrumentation allows developers to gather information about key metrics such as resource consumption (CPU, memory usage) and execution time. By instrumenting code, developers can gain insights into code hot paths, identify performance bottlenecks, and optimize the code for better performance efficiency.

In an ideal environment, you should do code analysis early in the software development lifecycle. The earlier you catch a code issue, the cheaper it's to fix it. You want to automate as much of this code analysis as possible. Use dynamic and static code analysis tools to reduce the manual effort. However, keep in mind that this testing is still a simulation of production. Production provides the clearest understanding of code optimization.

>  **Tradeoff**: Code monitoring tools are likely to increase costs.

Identify hot paths

By instrumenting your code, you can measure the resource consumption for different code paths. These measurements help you identify hot paths. Hot paths have a significant effect on performance and resource usage. They're critical or frequently executed sections of a program that require high performance and low latency. To identify code hot paths, consider these steps:

- *Analyze runtime data*: Collect runtime data and analyze it to identify areas of the code that consume significant resources, such as CPU, memory, or I/O operations. Look for patterns or sections of code that are frequently executed or take a long time to complete.
- *Measure performance*: Use profiling tools or performance testing frameworks to measure the execution time and resource consumption of different code paths. It helps identify bottlenecks and areas for improvement.
- *Consider business logic and user effect*: Evaluate the importance of different code paths based on their relevance to the application's functionality or critical business operations. Determine which code paths are crucial for delivering value to users or meeting performance requirements.

Optimize code logic

Optimizing code logic is about refining the structure and design of code to perform tasks with fewer resources. Improved logic reduces unnecessary operations. It creates faster execution with less resource consumption. You should remove any unnecessary operations within the code path that might affect performance. Prioritize optimizing hot paths to see the greatest performance efficiency gains. To optimize code logic, consider the following strategies:

- *Remove unnecessary function calls*: Review your code and identify any functions that aren't essential for the desired functionality and might affect performance negatively. For example, if a function call performs a validation completed earlier in the code, you can remove the unnecessary validation function call.
- *Minimize logging operations*: Logging can be helpful for debugging and analysis, but excessive logging can affect performance. Evaluate the necessity of each logging operation and remove any unnecessary logging calls that aren't critical for performance analysis.
- *Optimize loops and conditionals*: Analyze loops and conditionals in your code and identify any unnecessary iterations or conditions that can be eliminated. Simplifying and optimizing these structures can improve the performance of your code. Minimize function calls within loops, and eliminate redundant calculations. Consider moving computations outside the loop or using loop unrolling.
- *Reduce unnecessary data processing*: Review your code for any unnecessary data processing operations, such as redundant calculations or transformations. Eliminate these unnecessary operations to improve the efficiency of your code.

- *Optimize data structures*. To efficiently store and retrieve data, select appropriate data structures, such as arrays, linked lists, trees, and hash tables. Choose the best data structure for a specific problem. A suitable data structure improves application performance.

- *Minimize network requests*: If your code involves making network requests, minimize the number of requests and optimize their usage. Batch requests when possible and avoid unnecessary round trips to improve performance.

- *Minimize allocations*: Identify areas where excessive memory allocation occurs. Optimize the code by reducing unnecessary allocations and reusing existing resources when possible. By minimizing allocations, you can improve memory efficiency and overall performance. Use the appropriate memory management and garbage collection strategies for your programming language.

- *Reduce data structure size*: Assess the size of your data structures, such as classes, and identify areas where reduction is possible. Review the data requirements and eliminate any unnecessary fields or properties. Optimize memory usage by selecting appropriate data types and packing data efficiently.

- *Use performance-optimized SDKs and libraries*. Use native SDKs or performance-optimized libraries. Native SDKs are designed to interact with the services and resources on a platform or within a framework. For example, cloud-native SDKs work better with cloud service data planes than with custom API access. SDKs excel at handling network requests and optimizing interactions. Performance-optimized libraries, such as Math.NET, contain performance-optimized functions. When you apply the functions appropriately, you can improve your workload's performance.

- *Cross-cutting implementation*: Consider the effects of cross-cutting implementations, such as middleware or token checks, and assess whether they negatively affect performance.

Review the performance recommendations specific to the programming language you're working with. Evaluate your code against these recommendations to identify areas for improvement.

> **Tradeoffs**:
>
> - Optimizing code and hot paths requires developer expertise in identifying code inefficiencies is subjective and might be highly skilled individual required for other tasks.
> - SDKs provide convenience and eliminate the complexities of interacting with APIs. But SDKs might limit your control and customization options for custom

> code.

Optimize memory management

Optimizing memory management involves refining the way a workload uses, allocates, and releases memory resources to improve efficiency. Proper memory management improves code performance because it reduces the overhead of memory operations. Efficient memory usage reduces latency, prevents system slowdowns or crashes, and maximizes the throughput of computational tasks. Consider the following strategies to optimize memory management.

Debug memory issues. Memory dumps are application memory snapshots. They capture the memory state of an application at a specific point in time. Memory dumps enable retrospective analysis of memory-related issues. Select the appropriate type of memory dump based on the nature of the problem you're trying to diagnose and the resources available. You should use miniature dumps for routine debugging and full dumps for complex, critical issues. This strategy provides a balance between resource usage and diagnostic capabilities. Many code hosting services support memory debugging. You should prefer services that support memory analysis over those services that don't. Here are the basic steps to debugging memory issues:

1. *Capture memory dumps*: Begin by setting up a mechanism to capture memory dumps during your application's runtime. The capture can be triggered manually, automatically, or when specific conditions (like excessive memory consumption) are met. Some cloud services might already offer this process.

2. *Analyze memory dumps*: After you collect the memory dumps, analyze them. Numerous tools can assist you in inspecting these dumps, such as WinDbg for Windows applications or GDB for Unix-based systems.

3. *Identify memory leaks*: Focus on identifying memory leaks during the analysis. Memory leaks arise when your application allocates memory but fails to release it when the memory is no longer required. Search for objects or data structures that remain in memory even when they should be deallocated.

4. *Fix and test*: Upon identifying the problematic code, concentrate on resolving the memory issues. Resolutions might involve releasing memory correctly, optimizing data structures, or reevaluating memory management practices. Confirm that your solutions undergo rigorous testing to ensure their efficacy.

5. *Iterate and monitor*: Memory management is a continuous process. Routinely monitor your application's memory usage and persist in collecting memory dumps

in production. Regularly revisit the analysis and optimization stages to make sure memory issues don't reappear with subsequent code modifications.

By incorporating memory dump analysis into your software development lifecycle, you can amplify the reliability and efficiency of your applications. It helps to reduce the likelihood of memory-related issues in production.

Reduce memory allocations. Minimize memory allocations to reduce the overall memory footprint of the code. Your workload can utilize the available memory efficiently. There's less need for the garbage collector to reclaim unused memory, and it reduces the frequency and duration of garbage collection cycles. Memory allocations can be costly, especially if you perform them frequently. Minimize memory allocations, so the code can run quickly and efficiently.

Caches store frequently accessed data close to the processor, which improves performance. When you minimize memory allocations, there's less contention for cache space, so you can effectively utilize the cache. A high number of memory allocations can degrade application performance and generate errors. Other ways to minimize memory allocations include:

- *Local variables*: Use local variables instead of global variables to minimize memory consumption.
- *Lazy initialization*: Implement lazy initialization to defer the creation of objects or resources until they're needed.
- *Buffers*: Manage buffers effectively to avoid allocating large memory buffers.
- *Object pooling*: Consider object pooling to reuse large objects instead of allocating and deallocating them.

For more information, see Reduce memory allocations and The large object heap on Windows systems.

Use concurrency and parallelism

Using concurrency and parallelism involves executing multiple tasks or processes either simultaneously or in an overlapping manner to make efficient use of computing resources. These techniques increase the overall throughput and the number of tasks that a workload can process. When you run tasks concurrently or in parallel, it reduces the runtime of the application and decreases latency and increases response times. Concurrency and parallelism enable efficient utilization of computing resources, such as

CPU cores or distributed systems. Concurrency and parallelism effectively distribute the workload among the computing resources.

Use parallelism. Parallelism is the ability of a system to simultaneously trigger multiple tasks or processes on multiple computing resources. Parallelism divides a workload into smaller tasks that are run in parallel. You can achieve parallelism by using techniques like multiprocessing or distributed computing. Distribute tasks across multicore processors to optimize workload management. Optimize code to take advantage of the CPU architecture, threading models, and multicore processors. When you run code in parallel, performance improves because the workload is distributed across multiple cores.

Use concurrency. Concurrency is the ability of a system to run multiple tasks or processes. Concurrency enables different parts of a program to make progress independently, which can improve overall performance. You can implement concurrency by using techniques like multithreading, in which multiple threads run concurrently within a single process. You can also use asynchronous programming, in which tasks are triggered concurrently.

- *Asynchronous programming*: Asynchronous programming is an approach to trigger tasks without blocking the main thread. Asynchronous programming enables a program to trigger tasks while waiting for long-running operations to finish. With asynchronous programming, the program can initiate multiple tasks and wait for them to complete asynchronously. The program doesn't have to wait for each task to finish before moving on to the next one.

  There are many asynchronous programming techniques and patterns, depending on the programming language and platform. One common approach is to use asynchronous keywords and constructs, such as `async` and `await`, in languages like C#. With these keywords, you can define asynchronous methods. For HTTP traffic, consider using the Asynchronous Request-Reply pattern.

  Many frameworks and libraries provide built-in support for asynchronous programming. For example, in the .NET platform, you can implement asynchronous operations by using patterns like Task-Based Asynchronous pattern and Event-Based Asynchronous pattern. The specific implementation of asynchronous programming varies depending on the programming language, platform, and requirements of the application.

- *Queues*: A queue is a storage buffer located between a requesting component (producer) and the processing component (consumer) of the workload. There can be multiple consumers for a single queue. As the tasks increase, you should scale the consumers to meet the demand. The producer places tasks in a queue. The

queue stores the tasks until a consumer has capacity. A queue is often the best way to hand off work to a processing service that experiences peaks in demand. For more information, see Queue-Based Load Leveling pattern and Storage queues and Service Bus queues.

Use connection pooling

Connection pooling is the practice of reusing established database connections instead of creating a new connection for every request. It can be expensive to establish a connection to a database. You have to create an authenticated network connection to the remote database server. Database connections are especially expensive for applications that frequently open new connections. Connection pooling reuses existing connections and eliminates the expense of opening a new connection for each request. Connection pooling reduces connection latency and enables high database throughput (transactions per second) on the server. You should choose a pool size that can handle more connections than you currently have. The goal is to have the connection pool quickly handle new incoming requests.

Understand connection pooling limits. Some services limit the number of network connections. When you exceed this limit, connections might slow down or terminate. You can use connection pooling to establish a fixed set of connections at startup time and then maintain those connections. In many cases, a default pool size might consist of only a few connections that perform quickly in basic test scenarios. Your application might exhaust the default pool size under scale and create a bottleneck. You should establish a pool size that maps to the number of concurrent transactions that are supported on each application instance.

Test the connection pool. Each database and application platform has slightly different requirements for setting up and using a pool. Test your connection pool to ensure it works efficiently under load.

>  **Risk**: Connection pooling can create pool fragmentation and degrade performance.

Optimize background jobs

Many applications require background tasks that run independently of the UI. The application can start the job and continue to process interactive requests from users. Examples of background jobs include batch jobs, processor-intensive tasks, and long-running processes, such as workflows. Background tasks shouldn't block the application or cause inconsistencies due to delayed operation when the system is under load. To

improve performance, you can scale compute instances that host background tasks. For more information, see Background jobs and Scaling and performance considerations.

Optimize infrastructure performance

Optimizing infrastructure performance means enhancing and adjusting infrastructure elements to ensure peak operation and the best use of resources for a workload. By fine-tuning infrastructure, you can minimize waste, reduce lags, and achieve more with the available resources. It ensures that workloads run reliably and swiftly, leading to improved user experiences and cost savings. To optimize infrastructure performance, consider the following strategies:

Add usage limits. You can implement usage limits on some workload components. For example, to remove unstable pods, you can define pod CPU and memory limits in Azure Kubernetes Service (AKS). To optimize performance, you can define memory limits in Java virtual machines (VMs).

Streamline infrastructure. Simplify your workload to reduce the potential for interaction, dependency, and compatibility issues. When you simplify your workload, you optimize resource utilization of memory, processing power, and storage.

Reduce load. To reduce load on a workload, minimize the demand placed on an application and enable resources to perform their primary tasks. For example, it's common practice to avoid running security solutions within your code or on individual compute instances. Instead, web servers should serve HTTP requests. Web application firewalls and gateway resources can handle security checks. The following strategies help reduce the load on your workload:

- *Eventual consistency*: Adopt an eventual consistency model to enhance performance by allowing data to be slightly dated. Eventual consistency reduces the immediate demand on CPU cycles and network bandwidth for constant data updates.

- *Delegate tasks*: Delegate server tasks to clients or intermediaries, such as search indexes and caches. Delegate tasks like sorting data, filtering data, or rendering views. When you offload these tasks, you reduce the workload on your servers and improve performance.

Optimize the network. To optimize a workload network for performance, configure and fine-tune the network infrastructure. Ensure that the workload can operate at its highest level of efficiency.

- *Network protocols*: Upgrade to modern protocols like HTTP/2, which enable multiple requests to be sent over a single connection. Modern protocols reduce the overhead of establishing new connections.

  >  **Tradeoff**: Modern protocols might exclude older clients.

- *Network chattiness*: Batch network requests together to reduce the number of requests. Instead of making multiple small requests, combine them into larger requests to reduce network overhead.

- *Database queries*: Ensure that database queries retrieve only the necessary information. Avoid retrieving large amounts of unnecessary data, which can lead to increased network traffic and slow performance.

- *Static data*: Utilize a content delivery network to cache frequently accessed static content that's close to the users. When you cache data, it doesn't have to travel over long distances. Caching improves response times and reduces network traffic.

- *Log collection*: Collect and retain only the log data that's necessary to support your requirements. Configure data collection rules and implement design considerations to optimize your Log Analytics costs.

- *Data compression*: Compress and bundle HTTP content and file data to allow fast transmission between clients and servers. Compression shrinks the data that a page or API returns and sends back to the browser or client app. Compression optimizes network traffic, which can accelerate application communication.

  >  **Tradeoff**: Compression adds server-side and client-side processing. The application must compress, send, and decompress data. Multicast communication, or communication to multiple recipients, can create decompression overhead. You need to test and measure the performance variations before and after implementing data compression to determine if it's a good fit for your workload. For more information, see Response compression in ASP.NET Core.

Azure facilitation

Instrumenting code: Azure Monitor Application Insights supports automatic instrumentation (autoinstrumentation) and manual instrumentation of application code. Autoinstrumentation enables telemetry collection without touching the application's code. Manual instrumentation requires code changes to implement the Application

Insights or OpenTelemetry API. You can use Application Insights Profiler to help optimize hot paths.

Optimizing code logic: Azure offers SDKs and libraries for various programming languages to interact with Azure services. Use SDKs to simplify interactions between applications and Azure resources. SDKs provide optimal interaction with Azure services, which reduces latency and enhances efficiency.

Optimizing memory management: Use the smart detection feature of Application Insights to analyze memory consumption and help to identify and address memory leaks.

Azure App Service has a profiler and memory dump collection and analysis feature. The App Service autohealing feature can automatically take memory dumps and profile traces of .NET and Java apps.

Using concurrency and parallelism: Different Azure services provide unique support for concurrency, such as Azure Cosmos DB, Azure Functions and Blob storage. For parallelism, services AKS supports deploying containerized applications, which improves parallel processing.

Azure Batch is a cloud-based job scheduling service that you can use to enable parallel and high-performance computing without the need for infrastructure setup. For more information, see Background jobs.

Optimizing infrastructure performance: Implement Azure Resource Manager templates to define and deploy infrastructure by using code. Use these templates to implement efficient, repeatable, and consistent resource deployments. Azure Policy provides governance capabilities to ensure that resource deployments adhere to organizational best practices and standards.

For asynchronous programming, use scalable queuing services, like Azure Queue Storage and Azure Service Bus, to facilitate asynchronous programming. You can queue tasks and independently process them. To support asynchronous operations, Azure Marketplace offers third-party queues and tools that you can integrate with Azure services.

Related links

- AKS
- Application Insights smart detection feature
- Asynchronous Request-Reply pattern
- Avoid memory allocations

- Azure Batch
- Azure Policy
- Azure Resource Manager templates
- Azure SDKs
- Background jobs
- Background jobs scaling and performance considerations
- Compress file data
- Compress HTTP content
- Define pod CPU and memory limits
- Event-Based Asynchronous pattern
- Java virtual machines (VMs)
- Large object heap
- Pool fragmentation
- Queue-Based Load Leveling pattern
- Response compression in ASP.NET Core
- Storage queues and Service Bus queues
- Task-Based Asynchronous pattern

Performance Efficiency checklist

Refer to the complete set of recommendations.

Performance Efficiency checklist

Recommendations for optimizing data performance

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Performance Efficiency checklist recommendation:

> **PE:08** **Optimize data performance. Optimize data stores, partitions, and indexes for their intended and actual use in the workload.**

This guide describes the recommendations for optimizing data performance. Optimizing data performance is about refining the efficiency with which the workload processes and stores data. Every workload operation, transaction, or computation typically relies on the quick and accurate retrieval, processing, and storage of data. When data performance is optimized, the workload runs smoothly. Compromised data performance creates a domino effect of poor performance efficiency. Failure to optimize data performance results in response delays, heightened latency, and curtailed scalability. It jeopardizes the efficiency of the entire workload.

Definitions

Term	Definition
CAP theorem	A framework used to consider consistency, availability, and partition tolerance to help explain the tradeoffs in data consistency.
Database index rebuilding	A maintenance activity that drops and recreates an index.
Database index reorganization	A maintenance activity that optimizes the current database index.
Data store	A resource that stores data such as a database, object store, or file share.
Eventual consistency	A data synchronization model that allows for temporary inconsistency in data replicas before they eventually sync.
Index	A database structure that provides quick access to items.
Online analytical processing (OLAP)	A technology that organizes large business databases, supports complex analysis, and performs complex analytical queries without negatively affecting transactional systems.
Online transaction processing (OLTP)	A technology that records business interactions as they occur in day-to-day operations of an organization.

Term	Definition
Optimistic concurrency	An approach for updating databases that uses snapshots to make updates instead of traditional locking mechanisms, improving performance and scalability.
PACELC theorem	A framework used to consider partition tolerance, availability, consistency, and latency to help explain the tradeoffs in data consistency.
Partitioning	The process of physically dividing data into separate data stores.
Query tuning	A process that optimizes the speed of a database query.
Read replica	A live copy of a primary database that enables you to offload read traffic from a write database.

Key design strategies

To optimize data usage, ensure that data stores, partitions, and indexes are optimized for their intended use and for their actual use in a workload. Optimized data usage can improve query performance, reduce resource consumption, and enhance overall system efficiency. Consider the following strategies:

- *Profile data.* Understand your data and ensure that your data model is well-suited for your workload. Consider factors such as data normalization, indexing strategies, and partitioning techniques. For efficient data retrieval, ensure that you select appropriate data types, define relationships between entities, and determine an optimal indexing strategy.

- *Fine-tune your data storage configuration.* Configure your data storage infrastructure to align with your workload requirements. Select an appropriate storage technology, for example relational databases, NoSQL databases, and data warehouses. Optimize storage settings, such as buffer size, caching mechanisms, and compression.

- *Optimize query performance.* Analyze and optimize queries that run in the workload. Use techniques such as query optimization, indexing, and caching. To identify bottlenecks, use query plans and performance monitoring tools, and then make necessary improvements.

- *Regularly monitor and tune the system.* Continuously monitor the performance of your workload and iterate on the data storage configuration and query optimizations. Based on performance tuning best practices, analyze system metrics, identify areas of improvement, and implement changes.

Profile data

Data profiling involves examining the data from a source and gathering information about it. The objective is to understand the quality, structure, and characteristics of workload data. This process allows for the identification of issues such as missing values, duplicates, inconsistent formats, and other anomalies. For effective data profiling, consider the following strategies:

- *Understand the data structure.* Examine the structure of your data, including tables, columns, and relationships. Determine the data types, lengths, and constraints that are applied to each column. Data structure evaluation helps you understand how the data is organized and how it relates to other data elements.
- *Analyze the data volume.* Assess the volume of your data to understand the overall size and growth patterns. Determine the number of records or documents and the size of individual tables or collections. This information helps you estimate storage requirements and identify scalability issues.
- *Identify data relationships.* Explore the relationships between data elements, such as primary and foreign key relationships. Understand how data is connected, so you can determine how changes in one table or document might affect related data.
- *Assess data quality.* Evaluate the quality of your data by examining factors such as completeness, accuracy, consistency, and uniqueness. Identify data anomalies, missing values, or duplicate records that might affect data integrity and query performance. This step helps you identify areas for data cleansing and improvement.
- *Capture data distribution.* Analyze the distribution of values within each column to determine data patterns. Identify frequent and rare values, outliers, and data skews. To optimize query performance, choose appropriate indexing strategies and query optimization techniques based on the distribution.

Monitor data performance

Data performance monitoring is the practice of consistently tracking the efficiency of data stores, partitions, and indexes in real-time. It involves collecting and analyzing performance metrics specific to data operations, using tools tailored for system-level, database-specific, or third-party monitoring solutions. Effective data performance monitoring allows you to proactively identify and mitigate potential bottlenecks,

ensuring that data-related processes and tasks are efficient. To monitor data performance, consider the following strategies:

- *Collect data-specific metrics.* Gather key metrics that directly relate to data performance. These metrics include query response times, data throughput, disk I/O related to data access, and the load times of specific data partitions.
- *Set up data alerts.* Set up alerts specifically for data metrics. Use predefined thresholds or anomalies in these metrics to trigger alerts. Alerts enable you to receive notifications when performance metrics exceed acceptable ranges or show abnormal behavior. For instance, if a database query takes longer than expected or if data throughput drops significantly, it would trigger an alert. You can set up these alerts using specialized monitoring tools or custom scripts.
- *Diagnose data performance issues.* Regularly review the collected data metrics to pinpoint potential performance bottlenecks or degradation in data operations. Visualization tools or dashboards can be invaluable in this process, helping to highlight trends, bottlenecks, and outliers in data performance. Once identified, delve into the root causes of these issues and strategize appropriate remediation steps.

Partition data

Partitioning involves dividing large datasets or high-volume workloads into smaller, manageable subsets. Partitioning enhances data performance efficiency by distributing the workload and improving parallel processing. It also ensures more effective data access based on specific needs and query patterns. You can partition data vertically or horizontally (also called sharding).

Strategy	Definition	Example	Use cases
Vertical partitioning	Divide a table into smaller tables by selecting specific columns or fields for each partition. Each partition represents a subset of the complete data.	If you have a table with columns A, B, C, and D, you could create one table with columns A and B and another with columns C and D.	- A table contains many columns, but queries don't access all columns together. - Some columns are larger than others and separating them can boost I/O performance. - Different data parts have diverse access patterns.

Strategy	Definition	Example	Use cases
Horizontal partitioning	Split data based on rows or ranges of values (also known as sharding). Each partition contains a subset of rows with similar characteristics.	If you have a table with rows 1 to 1000, you might create one partition with rows 1 to 500 and another with rows 501 to 1000.	- A dataset is too large for a single location or server. - Data is accessed based on specific ranges or filters. - Need to distribute the workload across physical nodes or servers for enhanced performance.

To partition your data, consider the following steps:

- *Analyze data and queries*. Analyze data and query patterns to identify suitable partitioning or sharding strategies. Understand the nature of the data, access patterns, and distribution requirements.
- *Determine a key*. Choose a partitioning or sharding key to distribute data across partitions or shards. Carefully select the key based on data characteristics and query requirements.
- *Determine logic*. Determine a partitioning or sharding logic based on the chosen key. Consider dividing the data into ranges, applying hashing algorithms, or using other partitioning techniques.
- *Configure the infrastructure*. Configure the database system to support partitioning or sharding. Consider creating the necessary infrastructure, defining the partitions or shards, and configuring the data distribution.

For more information, see Data partitioning guidance.

Optimize database queries

Optimizing database queries refines queries using techniques such index hints and caching. These adjustments increase efficiency and speed of data retrieval. As a result, the database has a lighter workload, resources work more effectively, and users enjoy smoother interactions. To optimize database queries, consider the following strategies:

- *Rewrite queries*. Review and analyze complex queries to identify opportunities to rewrite them. Consider restructuring query logic, eliminating redundant operations, or simplifying query syntax.

- *Avoid the N+1 query problem.* Minimize the number of roundtrips to the database by using joins and batch fetching to retrieve related data efficiently.
- *Reorder joins.* Evaluate the query plan and consider rearranging the join order to minimize the number of rows in each join operation. The order in which you join tables can affect query performance.
- *Use index hints.* Use index hints so a database engine can specify the use of indexes when it runs a query. Index hints guide the optimizer to select the most appropriate indexes.
- *Cache queries.* Store the results of frequently run queries in memory. Query caching eliminates the need for repeatedly running the same query, and it reduces query processing overhead.
- *Optimize locking.* Avoid unnecessary or restrictive lock hints in queries. Efficient locking strategies can enhance query performance and concurrency. Apply optimized locking mechanisms that the database system provides. Analyze and adjust isolation levels to balance data consistency and query performance.
- *Monitor and tune.* Monitor query performance metrics, such as runtime, resource utilization, and query throughput. Use database profiling tools and monitoring functionalities to identify poorly performing queries. Evaluate and fine-tune query plans based on collected performance data. Analyze query plans and wait statistics to identify bottlenecks. Use that information to optimize query performance.

Optimize index performance

Indexes enhance data retrieval speed by allowing databases to swiftly find data using specific columns or fields. When you optimize these indexes, sorting and join operations become more efficient, leading to faster queries. Well-optimized indexes cut down on the disk I/O operations required for queries. Removing unneeded or redundant indexes also frees up valuable storage space. To optimize index performance, consider the following strategies:

- *Analyze query patterns.* Understand the query patterns that run on your database. Identify the queries that run frequently and might degrade performance. Analyze query patterns to determine which indexes are beneficial for optimizing performance.
- *Evaluate existing indexes.* Review the existing indexes in your database. Evaluate their usage, performance effects, and relevance to the query patterns. Identify

redundant or unused indexes that you can remove to improve write performance and reduce storage overhead.

- *Identify columns for indexing.* Identify columns that are frequently used in the *where*, *join*, and *order by* clauses of your queries. These columns are potential candidates for indexing because they can enable fast data retrieval.
- *Choose an appropriate index type.* Select an appropriate index type based on your database system. Common options include b-tree indexes for equality and range queries, hash indexes for exact match queries, and full-text indexes for text search operations. Choose an index type that best matches your query requirements.
- *Consider index column order.* When you create composite indexes, or indexes with multiple columns, consider the order of the columns. Place the columns that are most frequently used in queries at the beginning of the index. Column order helps ensure that your workload is effectively using indexes for a wide range of queries.
- *Balance index size.* Avoid creating indexes on columns with low cardinality, or columns that have a low number of distinct values. Such indexes can be inefficient and increase the size of your database. Instead, index columns that have a high selectivity.
- *Maintain index usage.* Continuously monitor the usage and performance of your indexes. Look for opportunities to create new indexes or modify existing indexes based on changes in query patterns or performance requirements. Remove or update indexes that are no longer beneficial. Indexes have maintenance overhead. As data changes, indexes can fragment and affect performance. Regularly perform index maintenance tasks, such as rebuilding or reorganizing indexes, to ensure optimal performance.
- *Test and validate.* Before you revise indexes in a production environment, perform thorough testing and validation. Measure the performance effect of index revisions by using representative workloads. Verify the improvements against predefined benchmarks.

>  **Tradeoff**: B-tree indexes might have high storage overhead, and exact-match queries might be slow. Hash indexes aren't suitable for range queries or comparison operators. Full-text indexes might have high storage requirements, and nontextual data queries might be slow.

Consider data compression

Data compression is the process of reducing the size of data to optimize storage space and improve workload performance efficiency. Compressed data requires less storage space and less bandwidth for transmitting, which results in fast data transfer. You would compress data to reduce your storage footprint and improve data access times. When you compress data, it reduces I/O operations and network bandwidth requirements.

Lossless compression and lossy compression are data compression algorithms. Lossless compression algorithms reduce the size of data without losing any information. Lossy compression algorithms achieve high compression ratios by removing less important or redundant information.

>  **Tradeoff**: To compress and decompress data, you need computational resources, like CPU and memory. The more data that you compress, the more resources you need.

Archive and purge data

Archiving and purging are strategies that streamline data storage. Archiving relocates older, less-frequently accessed data to a more cost-effective storage. Purging data permanently removes redundant data. They contribute to performance efficiency by reducing data volume, increases data access speed, and reducing backup and recovery times:

- *Reducing data volume*: Less data means faster processing times, ensuring quick responses to user requests.
- *Increasing data access speed*: A trimmed dataset allows for swifter queries and data retrieval, optimizing system responsiveness.
- *Reducing backup and recovery times*: Smaller datasets expedite backup and restoration processes, minimizing downtime and ensuring consistent performance.

Archiving and purging are instrumental in maintaining peak performance efficiency in data-driven systems.

Optimize storage load

Optimizing storage load means streamlining requests to the storage system. It helps eliminate unnecessary requests. It also enhances data retrieval and prevents overwhelming the storage. Optimizing the storage load ensures the storage system remains responsive to legitimate requests and maintains peak performance. Implement

strategies to reduce the processing burden on the data store. To optimize data store load, consider the following strategies:

Use caching

Caching stores commonly accessed data in a fast-access storage area, making data retrieval quicker than pulling it from the main source. This technique boosts data performance by cutting down on access times and avoiding repetitive data fetches. Caching improves read speeds and user response times, especially for frequently accessed data This method is most effective on static data or data that rarely changes.

To ensure optimal caching efficiency, consider factors like expiration policies, eviction strategies, and managing cache size. Adjust settings, such as the time to live (TTL), for optimal performance. To use a cache to optimize storage load, consider the following strategies:

- *In-memory caching*: Perform in-memory caching to store frequently accessed data in memory for fast retrieval. You can use this technique for application data that's expensive to compute or retrieve from a database. In-memory caching is useful for data that you read frequently but don't change frequently.
- *Database query caching*: Use this technique to cache the results of database queries to avoid running the same query multiple times. Database query caching is useful for complex and time-consuming database queries. When you cache the results of a query, subsequent requests for the same query are returned quickly.
- *Content delivery network caching*: Use this technique to cache web content on distributed network servers to reduce latency and improve content delivery. Content delivery network caching is effective for static content, like images, CSS files, and JavaScript files. Content delivery networks store copies of content in multiple locations worldwide, so users can access the content from a server that's near them geographically.

Use read replicas

Many databases support multiple read replicas. Distribute read queries across replicas to minimize the demand on the write database. Each read replica can serve a subset of traffic, which can improve performance.

When you have a workload with multiple data replicas that you expect to stay in sync, it's helpful to model this distributed system by using the PACELC theorem. The PACELC theorem helps you understand latency versus constancy tradeoff choices in the

nonpartitioned state of the system. Use this information to help you choose a database engine and data sync strategy that best addresses the system in a partitioned and nonpartitioned state. For more information, see Command and Query Responsibility Segregation (CQRS) pattern.

Optimize data consistency

In a distributed workload, where data resides across multiple nodes or locations, the level of consistency you select determines how quickly changes in one location reflect in others. Opting for stricter consistency consumes more compute resources and can negatively affect performance efficiency. On the other hand, a less strict consistency level, like eventual consistency introduces temporary inconsistencies among nodes but can boost performance efficiency.

Eventual consistency strikes a balance between data accuracy and workload performance. Changes spread gradually instead of instantly, boosting workload responsiveness and data processing speed. Although it introduces short-lived inconsistencies, the workload eventually presents consistent data across all nodes. Choosing eventual consistency can elevate a workload's performance and further enhance its availability and scalability.

Optimize data updates

You can use optimistic concurrency to handle concurrent updates to the same data. Instead of locking data and preventing other updates, optimistic concurrency allows multiple users or processes to work concurrently and assumes that conflicts are rare.

With optimistic concurrency, each update operation includes a version or timestamp that represents the state of the data at the time of the update. When a conflicting update is detected, the system resolves the conflict by rejecting the update or merging the changes.

Optimistic concurrency minimizes contention and allows concurrent updates to proceed without unnecessary locking. It reduces wait time for resources and provides high throughput.

Optimize data movement and processing

Optimizing data movement and processing involves improving the efficiency and performance of operations related to data extraction, transformation, loading, and

processing. Consider the following key aspects of optimizing data movement and processing:

- *Extract, transform, and load (ETL) optimization*: Optimize ETL processes to minimize processing time. You can streamline the extraction process, implement efficient transformation algorithms, and optimize the loading process. When you make each step efficient, you can optimize the overall workflow.
- *Parallel processing*: Utilize parallel processing techniques to improve performance. When you distribute data processing tasks across multiple threads or nodes, you can divide and process the workload concurrently, which results in fast processing.
- *Batch processing*: Group similar tasks together to reduce overhead caused by repeated operations. Process multiple tasks in a batch to reduce overall processing time.

Optimize storage design

Optimizing storage design entails crafting a precise data storage architecture and selecting appropriate storage technologies. A streamlined storage design enhances data access, retrieval, and manipulation. Through strategic storage design, a workload achieves improved response times and overall functionality.

Design for data proximity

Data proximity refers to the strategic placement of data closer to the users or services that access it most frequently. By reducing the physical or logical distance between data and its users, data proximity ensures faster data access and improved responsiveness. To optimize design for close proximity, consider these strategies:

- *Evaluate data access patterns*: Assess your workload's access patterns and frequently accessed data. This analysis can help determine where to place data for maximum benefit.
- *Choose solutions that support data relocation*: Consider solutions that offer dynamic data relocation based on changing access patterns, ensuring optimal data positioning.
- *Choose solutions that support data synchronization*: If catering to a distributed user base, opt for solutions that facilitate data synchronization across various regions, ensuring that data replicas are available in proximity to users.

> **Tradeoff**: If underlying data changes frequently, implement a cache invalidation mechanism to ensure that the cached data remains up to date.

Use polyglot persistence

Polyglot persistence is the practice of using multiple data storage technologies to store and manage different types of data within an application or system. Different types of databases or storage solutions serve different data requirements.

Polyglot persistence takes advantage of the benefits of each data storage technology to ensure optimal performance and scalability for each type of data. For example, you might use a relational database to store structured, transactional data. And you might use a NoSQL database to store unstructured or semi-structured data.

Design a schema for each data storage technology based on the requirements of the data. For relational databases, you might create normalized tables with appropriate relationships. For NoSQL databases, you might define document structures or key-value pairs. Develop the necessary components to interact with each data storage technology, such as APIs, data access layers, or data integration pipelines. Ensure that the application can read and write data to the appropriate data stores.

> **Tradeoff**: A data structure that has low normalization can improve performance but introduce complexities.

Separate OLTP and OLAP systems

To separate OLTP and OLAP systems, design and deploy distinct systems for transactional processing and analytical processing tasks. This separation allows you to optimize each system for its specific workload and characteristics.

OLTP systems are used for real-time transactional processing. They efficiently and reliably handle individual transactions. OLTP systems are typically used to perform day-to-day operational tasks, such as online order processing, inventory management, and customer data management. OLTP systems prioritize responsiveness, consistency, and concurrency.

OLAP systems are used for complex analytical processing and reporting. They handle large volumes of data and perform intensive calculations and aggregations. OLAP systems are used for tasks such as business intelligence, data mining, and decision support. OLAP systems prioritize query performance, data aggregation, and multidimensional analysis.

When you separate OLTP and OLAP systems, you can allocate appropriate resources and optimize each system for its specific workload. Separation allows you to apply different data modeling techniques to each system. OLTP systems typically use normalized schemas for efficient transactional processing. OLAP systems might use denormalized schemas or data warehousing techniques to optimize query performance.

Azure facilitation

Profiling data: Azure offers tools and services that you can use to profile data, such as Azure Data Catalog, Azure Purview, and Azure Synapse Analytics. These tools enable you to extract, transform, and load data from various sources, perform data quality checks, and gain insights into the data.

Monitoring data performance: To monitor data performance, you can use Azure Monitor to collect and analyze infrastructure metrics, logs, and application data. You can integrate Monitor with other services like Application Insights. Application Insights provides application performance monitoring and supports many platforms.

Application Insights collects usage and performance data. You can use Log Analytics to correlate that data with configuration and performance data across Azure resources.

You can use the insights feature of Azure SQL and Azure Cosmos DB to monitor your database. This feature enables you to diagnose and tune database performance issues.

Partitioning data: Azure offers various partitioning strategies for different data stores. Each data store might have different considerations and configuration options for data partitioning. For more information, see Data partitioning strategies.

Optimizing database queries and index performance: Use the query performance insight feature of Azure SQL Database to optimize queries, tables, and databases. You can use this feature to identify and troubleshoot query performance issues.

For relational databases, you should follow the index design guidelines, SQL Server index guidance, and Azure Cosmos DB index guidance. Use SQL Database to perform automatic tuning for queries to improve their performance.

In SQL databases, you should regularly reorganize or rebuild indexes. Identify slow queries and tune them to improve performance. Many database engines have query-tuning features. For more information, see Best practices for query performance.

Azure Cosmos DB has a default indexing policy that indexes every property of every item and enforces range indexes for any string or number. This policy provides you with efficient query performance, and you don't have to manage indexes upfront.

Optimizing storage load: Many Azure database services support read replicas. The availability and configuration of read replicas vary depending on the Azure database service. Refer to the official documentation for each service to understand the details and options.

Optimizing storage design: Azure offers many different data stores to fit your workload needs. Understand data store types and select an Azure data store for your application.

Related links

- Automatic tuning in SQL Database
- Azure Cosmos DB
- Azure Cosmos DB index guidance
- Azure SQL
- Best practices for query performance
- CQRS pattern
- Data partitioning guidance
- Data partitioning strategies
- Default indexing policy
- Index design guidance
- OLAP overview
- OLTP overview
- Partitioning best practices
- Reorganize or rebuild indexes
- Select an Azure data store for your application
- SQL Server index guidance
- Understand data store types

Performance Efficiency checklist

Refer to the complete set of recommendations.

Performance Efficiency checklist

Recommendations for prioritizing the performance of critical flows

Article • 11/30/2023

Applies to this Azure Well-Architected Framework Performance Efficiency checklist recommendation:

Expand table

PE:09	**Prioritize the performance of critical flows. The allocation of workload resources and performance optimization efforts should prioritize the flows that support the most important business processes, users, and operations.**

This guide describes the recommendations for prioritizing the performance of critical flows in a workload. Critical flows represent crucial business processes that generate revenue or drive high-priority operations. When you prioritize the performance of critical flows, you ensure the flows that have the most impact get the resources they need before lower priority flows. Failure to do this prioritization can have disproportionate negative effects on workload priorities and the user experience.

Definitions

Expand table

Term	Definition
Flow	In a workload, the sequence of actions that performs a specific function. A flow involves the movement of data and the running of processes between components of the workload.
Priority queue processing	The act of processing high-priority tasks before low-priority tasks.
Rate limiting	The act of limiting how many requests can access a resource.
System flow	The flow of information and processes within a system. The system automatically follows this flow to enable user flows or workload functionality.
User flow	The sequence that a user follows to accomplish a task.

Key design strategies

Critical flows refer to the key user flows for customers or the system and data flows for operations that are crucial to the workload functionality. These flows can include actions such as user registrations, sign-ins, product purchases, accessing pages behind a paywall, or any other key path or process within your workload.

Critical flows significantly affect the user experience or business operations. Critical flows have higher performance targets and service-level agreements than noncritical flows. Where resources are limited, noncritical flows should yield resource usage to critical flows. You need to identify, monitor, and prioritize all flows before isolating and optimizing critical flows.

Identify all flows

The first step in prioritizing the performance of critical flows is identifying all the flows within your workload. Flow identification involves systematically mapping and understanding every user paths and component communication. The focus is on understanding the performance metrics and potential impact of flows on workload performance.

By dissecting the workload into discrete flows, you can find performance bottlenecks, inefficient resource utilization, and opportunities for performance optimization. This knowledge exposes areas of needed improvement and is the first step to identifying critical flows. For more information, see Identify and rate user and system flows.

Monitor flow performance metrics

After you identify all flows within your workload, you need to collect performance metrics on each flow and monitor those metrics. Flow metrics provide insights into response times, error rates, and throughput. The goal is to consistently observe and record performance-related metrics to further refine your understanding of each flow's impact on workload performance. To monitor flow metrics, you can use the following tools to collect data:

- *Analytic and tracking tools*: These tools provide insights into user behavior and interactions within your application. By analyzing user data, you can identify the most common flows, bottlenecks, or potential issues.

- *Application performance monitoring (APM) tools*: Use APM tools to monitor the performance of your application and track how flows run. These tools provide visibility into response times, errors, and other performance metrics, allowing you to identify critical flows and optimize their performance.

- *Logging and debugging tools*: Use these tools to capture and analyze logs and debug information while your application runs. Review logs and debugging information to trace how flows are running and identify issues or errors.

Identify critical flows

With the performance data available, you can begin ranking all the flows and identifying the critical flows. The identification of critical flows involves evaluating the performance impact and criticality of each flow. Effective flow prioritization ensures that the most important flows receive the resources needed before less critical flows. To prioritize flows in your application, consider these steps:

- *Identify business impact*: Start by assessing the importance of each flow within your operations. Focus on how each flow aligns with your business objectives, its impact on users, and the potential negative effects of poor performance. For instance, while a free service tier might attract more users, a paid tier could be more vital for your business goals.

  Additionally, consider the performance impacts of a flow across one or more business processes. Multiple flows might support a single business process, but often, one flow has a significant effect on the performance of that process. You want to identify the flows that the greatest performance impact. Conversely, a single flow might underpin several processes. In such cases, the performance of this flow directly influences the efficacy of all related processes, and it's likely a critical flow.

- *Analyze performance data*: Analyze the performance metrics associated with each flow. Look for patterns, anomalies, or standout metrics that can provide insights into the flow's efficiency and importance. For example, system flows with significant usage are likely important flows.

- *Assign criticality rating*: Based on the business impact and performance indicators, you should prioritize the flows. Use criticality ratings of *High*, *Medium*, and *Low*. Flows with a significant business impact or high performance demand should receive a "High" criticality rating. These flows are your critical flows. Focus on flows with high user traffic or have a direct effect on revenue generation. The following table provides characteristics of critical (*High*) and noncritical flows (*Medium* to *Low*).

Expand table

Critical flows	Noncritical flows
High usage	Low usage
Business critical	Not business critical
Expensive operations	Small operations
Time-sensitive	Not time-sensitive
Production	Preproduction
Real-time processing	Batch processing
Latency sensitive	Not latency sensitive
Paying user	Nonpaying user
Premium tier	Basic tier
Important tasks	Nonessential tasks
High-revenue accounts	Low-revenue accounts

Isolate critical flows

The process of isolating critical flows is about providing dedicated resources or capacity to support critical flows. You want to allocate resources and attention to those flows that are essential for optimal user experience or significant business outcomes. The goal is to ensure critical flows receive enough computing power, network bandwidth, and resources to operate efficiently and effectively. By isolating critical flows, you can more easily manage the resources that support critical flows. Here are recommendations for isolating critical flows:

- *Resource segmentation*: Create separate resources for critical flows, allowing them to operate independently without interference from other processes. For example, you can isolate critical flows on dedicated network segments or by using dedicated servers to handle the processing needs of these flows. This approach helps minimize how noncritical flows can negatively affect critical flows.
- *Logical segmentation*: Use virtualization and containerization tools like Docker or Kubernetes to isolate flows at the software level. You can separate critical flows into virtual machines (VMs). By doing so, you create an isolated environment, reducing dependencies and potential interference from other flows.
- *Capacity allocation*: For critical flows, explicitly allocate a fixed set of capacity such as CPU, memory, and disk I/O. This allocation ensures that critical flows always

have enough resources to operate efficiently. Set resource quotas or limits by using orchestration platforms. By explicitly allocating resources to critical flows, you prevent resource contention and prioritize how they run.

> **Tradeoff**: Resource segmentation affects costs. When you dedicate resources to a flow, you often increase the cost and leave some resources underutilized. To justify the performance enhancements to critical flows, the increase in business impact must outweigh the increase in cost.

Optimize capacity allocation

When you can't isolate critical flows, the next best option is to prioritize critical flows in accessing available capacity. The optimization of capacity allocation is about strategically distributing available capacity to different flows based on their criticality. Capacity includes CPU, memory, storage, and network bandwidth. The goal is to ensure that the most critical flows (highest priority) receive the necessary capacity to operate effectively. To decide how to allocate capacity, consider these strategies:

- *Assess resource capacity*: Evaluate how much resource capacity can be allocated to the flows. Capacity might include resources such as CPU, memory, storage, and network bandwidth. Understand the limitations and constraints of your infrastructure or environment.

- *Analyze flow requirements*: Analyze the resource requirements of each flow. Understand the resources the flow needs to operate efficiently. For each flow, identify the resource demands, such as CPU utilization, memory requirements, and network bandwidth.

- *Prioritize allocations*: Match the available resource capacity to the resource requirements of the flows. Allocate resources based on flow priorities, ensuring that higher-priority flows receive the necessary resources to meet their requirements. Understand where your tightest constraints are and optimize capacity allocations where they're needed. For example, queues can process only some messages per minute, but some storage limits are hard to reach.

- *Use rate limiting*: To ensure that critical flows can consume the resources they need to meet their performance targets, apply rate limits to noncritical flows and tasks. Rate limits cap the number of requests lower-priority flows and users can make to constrained resources. For example, you might rate-limit nonpriority requests to an API. For more information, see the Rate Limiting pattern and Rate limiting an HTTP handler in .NET.

- *Use priority queue processing*: Priority queue processing gives high priority to certain requests. Queues usually have a first in, first out (FIFO) structure, but you can update your application to assign a priority to messages it adds to the queue. Use this capability to prioritize critical flows and users. For more information, see the Priority Queue pattern.

>  **Risk**: It can be a challenge to balance the needs of critical flows with the overall performance of a workload. Although you should prioritize critical flows, you shouldn't neglect noncritical flows. The overall performance efficiency of a workload depends on all flows. Neglected noncritical flows could create issues that affect all users. Too much noise from nonessential items steals attention from critical items. But too little noise could harm the entire workload. The amount of data and the number of alerts should reflect these balanced priorities.

Azure facilitation

Identifying and monitoring flows: Azure provides different solutions to help you monitor the performance of critical flows in your workload. Azure Monitor, Azure Monitor Logs, and Azure Application Insights are some of the services that offer comprehensive monitoring capabilities for several types of applications and workloads.

Optimizing capacity allocations: Some Azure services support resource segmentation, logical segmentation, and capacity allocation techniques to allocate capacity and resources to critical flows. You can isolate critical flows through techniques such as creating separate resources, increasing density, using virtualization and containerization, and explicitly allocating resources to critical flows.

Some Azure services, such as Azure API Management, provide built-in policies for rate limiting. Azure provides detailed guidance and a sample implementation of the Rate Limiting design pattern.

Azure supports priority queue processing. Azure Functions provides event-driven functions that you can trigger in various ways, including by a new message in a queue or topic. Combine Azure Functions with Azure Queue Storage or Azure Service Bus to process messages based on their priority.

Related links

- Priority Queue pattern
- Rate Limiting pattern
- Rate limiting an HTTP handler in .NET

- Azure API Management
- Azure Service Bus

Performance Efficiency checklist

Refer to the complete set of recommendations.

Performance Efficiency checklist

Recommendations for optimizing operational tasks

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Performance Efficiency checklist recommendation:

> **PE:10** **Optimize operational tasks. Monitor and minimize the effects of the software development lifecycle and other routine operations on workload performance. These operations include virus scans, secret rotations, backups, reindexing databases, and deployments.**

This guide describes the recommendations for optimizing operational tasks. Optimizing operational tasks is the process of minimizing the effects of tasks that you perform as part of routing workload operations. Operations activities use the same compute resources as the workload itself. Failure to consider the effects of operations tasks can cause the workload to miss its performance targets. It can also negatively affect the performance of the workload for your customers.

Definitions

Term	Definition
Blue-green deployment	A deployment strategy that uses two identical environments and controls the direction of traffic to new deployments (green deployments).
Database index rebuilding	A maintenance activity that drops and recreates an index.
Database index reorganization	A maintenance activity that optimizes the current database index.
Database schema	The general structure of a database and its relationships to other data.
Deployment slot	A feature of Azure App Service that enables you to deploy live apps with their own host names.
In-place upgrades	The process of upgrading a component or an application without replacing it or migrating it to a new environment.
Infrastructure as code (IaC)	A descriptive model for defining and deploying infrastructure, including networks, virtual machines, load balancers, and connection topologies.

Key design strategies

You need to take measures to reduce the effects of the software development lifecycle and other routine operations on workload performance. The goal is to ensure that routine operations, like virus scans, secret rotations, backups, index optimization (reorganization or rebuilding), and deployments, don't significantly degrade the performance of the workload.

Account for operational tasks

It's important to consider operational tasks when you set performance targets. By incorporating routine, regular, and ad-hoc tasks into performance targets, you can ensure that the workload operates efficiently. To account for operational tasks in performance targets, here are some key points to consider:

- *Identify operational tasks.* Identify and include relevant operational tasks in performance targets. Examples of routine tasks can include virus scanning, database index reorganization, database index rebuilding, disk or database backups, certificate rotations, patching an operating system, rotating passwords, rotating API keys, penetration testing, and audit reviews in production.
- *Evaluate performance targets.* Evaluate current performance targets and adjust them to account for operational tasks that are specific to the workload. Doing so ensures that performance targets align with the workload's operational requirements.

Optimize deployments

Optimizing deployments refers to refining the process of releasing resources and code to guarantee seamless performance and minimal interruptions. It involves planning, effective resource distribution, and thorough testing of both the infrastructure-as-code (IaC) and the application code before they are introduced to a live environment. Deployment inadequacies can lead to reduced speed and efficiency of a workload, potential resource constraints, and a compromised user experience in the operational setting. To optimize deployments, consider these strategies:

Assess acceptable downtime. If downtime is acceptable, you can implement deployment strategies that prioritize speed and efficiency. However, it's important to carefully assess the effect of downtime on business requirements before you make that decision. On the other hand, if downtime isn't acceptable, you need to implement deployment strategies that ensure continuous availability of the workload. Consider using techniques like blue-green deployments or canary deployments, where you

gradually roll out new versions of the workload while you monitor for issues. These strategies help minimize the effect of downtime and ensure a seamless user experience.

Deploy at current instance count. You should also avoid deployments that cause immediate scale operations. You shouldn't deploy resources into a live system with an instance count so low that it forces the system to immediately perform a scale operation. For example, your infrastructure-as-code (IaC) template might not match the number of instances that you need at the time of deployment. It might have an instance count of two, even though the current deployed environment is running eight instances. The deployment would remove six instances and negatively affect performance.

Use a blue-green deployment strategy. Deployments can cause service interruptions and downtime. To mitigate these issues, select a deployment strategy that minimizes performance impact, like a blue-green deployment. These approaches allow for seamless transitions between environments and reduce the risk of service disruptions. When you use the blue-green deployment approach, you have two separate environments: the blue and green environments. If any issues or performance degradation is detected in the green environment, you can easily roll back to the stable blue environment. This strategy helps you ensure minimal downtime and allows you to maintain a high level of performance for your workload. To deploy by using the blue-green approach, follow these general steps:

- *Deploy the new environment.* Set up the new environment (green) alongside the existing environment (blue) with the updated version of your application.
- *Validate the new environment.* Deployments can introduce latency and increase response times. Consider prewarming instances before cutover. Prewarming involves preparing the new environment by simulating production-like traffic and workload to ensure that the environment is ready to handle the expected load. It helps minimize the effects on latency and response times. Thoroughly test and validate the new environment to ensure that it functions correctly and meets performance expectations. Testing helps warm up caches, establish database connections, and ensure that the environment is ready to handle the expected load.
- *Gradually shift traffic.* After the new environment is prewarmed and validated, gradually shift production traffic from the old environment (blue) to the new environment (green). Initially, direct a small percentage of traffic to the green environment and gradually increase it after verifying its stability and expected application health. You can use a global load balancer or traffic management mechanism. The controlled traffic shifting allows you to identify any performance

issues early and take corrective actions before fully transitioning the workload to the new environment.

- *Monitor and optimize.* Deployments might use shared computing resources. Continuously monitor the performance and health of the new environment after you shift traffic. Make any necessary optimizations or adjustments to ensure the desired performance and user experience.

- *Remove the old environment.* After you successfully transition all traffic to the green environment, remove the blue environment from existing connections. This step helps optimize the cost of maintaining the old environment and ensures that new environments are free of configuration drift.

- *Repeat the process.* For future deployments, reverse the roles of the blue and green environments. Deploy changes to the new blue environment, validate them, orchestrate traffic transition, and decommission the old green environment.

Use multiple builds. Different types of builds can help you optimize build times and ensure the quality of deployments. For example, you can have continuous integration (CI) builds that trigger with every code commit. You could have nightly builds that run automated tests regularly, and release builds that are used for deploying to production. Each type of build should have a specific purpose, like continuous integration, automated testing, or production deployment. Testing and validation of the workload before deployment help identify and address issues or bugs early in the development process.

Consider feature flags. Feature flags are used in software development to control the visibility and behavior of certain features in an application. By using feature flags, developers can enable or disable specific features without needing to redeploy the application. Feature flags work by introducing conditional logic in the code that determines whether a feature should be enabled or disabled. This logic can be based on various factors, like user roles, user preferences, or specific conditions that are defined by the development team. By using feature flags, developers can gradually roll out new features to a subset of users or enable features for specific groups for testing (canary testing).

Optimize upgrades

An in-place upgrade is an upgrade to an existing resource or application. In-place upgrades can temporarily slow down or interrupt a workload. It's important to ensure that upgrades are compatible with the workload. Before you apply an upgrade, we recommend that you test it in a separate environment to identify any potential issues.

Provide a rollback plan in case any issues arise during the upgrade process. It's crucial to take a complete backup of critical data and configurations before you apply the upgrade. Monitor the upgraded system closely after the upgrade to ensure that everything functions as expected. The backup allows you to restore to a good state if you need to. You should prioritize scheduling the upgrade during off-peak hours to minimize the effect on users and workload performance. Notify users in advance about the planned upgrade, including the expected downtime and any necessary actions they need to take.

>  **Tradeoff**: Waiting to perform operations activities during off-peak hours can affect operational efficiency. It might be less convenient to have the personnel with the right skill set work during off-peak hours.

Optimize tooling

Essential tools for file integrity monitoring, virus scanning, intrusion detection, and other operational tasks can affect workload performance. They consume compute resources and can add latency and performance overhead. You need to test and understand the effects your tools have on workload performance. Based on the test results, you should fine-tune tool configurations, adjust scan frequency, and reallocate compute resources. For virus scanning, you could create a relevant exclusion list to minimize the duration of scans.

Optimize database operations

Optimizing database operations refers to the process of refining and fine-tuning database tasks to ensure maximum efficiency and minimal resource utilization. These operations include tasks like backups, schema changes, performance tuning, and monitoring. Efficient database operations lead to faster query responses, reduced system overhead, and an overall smoother user experience.

Schema changes involve modifying the structure of a database, such as adding or altering tables, columns, or indexes. These changes might require extra processing and resource utilization during the deployment process, potentially affecting the overall performance of the workload. Schema changes can disrupt performance to active queries, indexes, or transactions or cause data to be unavailable.

To minimize these effects, you should plan and test schema changes in a nonproduction environment. You can use various deployment techniques to implement schema updates. You should also use available schema changing tools to optimize the process. Archiving data and partitioning can help reduce the effects of schema changes.

Optimize backups

Backups consume workload resources like processing power, network bandwidth, and disk I/O. You need to test and select a backup strategy that minimizes these effects. You should perform backups during off-peak hours when you can. Your strategy should include incremental backups instead of full backups each time. Snapshots can be less resource intensive than backups. You should consider built-in platform backup and restore features rather than building a custom solution. You need to test these options and use a combination that provides the best performance for your workload.

Optimize monitoring and debugging

Excessive or poorly implemented logging, telemetry, instrumentation, and distributed tracing capture and collection can affect performance. Likewise, convenience features like remote debugging can also affect performance. You need to measure and know their performance effects on the environment. You don't want these processes to degrade performance. You should configure or disable any processes whose performance effects outweigh their benefits.

Azure facilitation

Accounting for operational tasks: Azure DevOps is a set of development tools and services that enable teams to plan, develop, test, and deliver software efficiently. It includes features like version control, continuous integration and delivery, project management, and more.

Azure provides service-to-service integration that minimizes the effects of many operational tasks. For example, services that integrate with Azure Key Vault often support seamless certificate rotation or secret rotation that minimizes effects on performance.

Optimizing deployments: App Service provides deployment slots. You can use deployment slots to deploy code to a nonproduction environment. You can swap app content and configuration elements between two deployment slots. For example, you can switch app content from a nonproduction slot to the production slot.

Azure Front Door and Azure Traffic Manager enable you to implement a blue-green deployment strategy. Some Azure compute services also support advanced deployment strategies like blue-green deployments. You can combine those services with your traffic shifting or instance warming strategy to mitigate the performance effects of deployment.

Optimizing database operations: Azure SQL Database automatically takes full backups, differential backups, and transaction log backups. Azure Cosmos DB automatically takes backups of your data at regular intervals. The automatic backups are taken without affecting the performance or availability of database operations. Azure Cosmos DB stores the backups in a separate storage service.

Optimizing backups: Some Azure data services support low-to-no performance impact for point-in-time recovery and indexing. Azure Backup is a reliable and scalable cloud-based backup solution that enables you to protect your data and applications. It provides features like incremental backups, compression, and encryption to minimize the effects on performance during backup operations. Azure Site Recovery helps you protect your applications by replicating them to a secondary location. It provides continuous replication and automated failover capabilities to minimize the downtime and performance impacts during backup and disaster recovery operations.

Related links

- Deployment slots
- Blue-green deployment strategy
- Azure SQL Database
- Azure Cosmos DB

Performance Efficiency checklist

Refer to the complete set of recommendations.

Performance Efficiency checklist

Recommendations for responding to live performance issues

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Performance Efficiency checklist recommendation:

> **PE:11** **Respond to live performance issues. Plan how to address performance problems by incorporating clear lines of communication and responsibilities. When a problematic situation occurs, use what you learn to identify preventive measures and incorporate them in your workload. Implement methods to return to normal operations faster when similar situations occur.**

This guide describes the best practices for responding to live performance issues. Live performance issues refer to real-time challenges and bottlenecks that can hinder the optimal functioning of a workload. Addressing these issues promptly not only facilitates the immediate detection and rectification of performance hiccups but also ensures that the workload consistently meets its performance benchmarks. Failing to address them can lead to complications, including slowdowns, crashes, and system unresponsiveness, and degrade the user experience. They can also prevent users from completing their tasks efficiently, and, in turn, tarnish the reputation of the organization.

Definitions

Term	Definition
Data correlation	Aligning logs, metrics, and events from various parts of your workload to pinpoint underlying causes.
Root cause analysis	A process for identifying the underlying factors that are responsible for a problem.
Self-healing	The ability to automatically repair issues without human intervention.
Self-prevention	Implementations within a workload to prevent potential issues and failures.

Key design strategies

When you experience a live performance issue, you need to be prepared with the right data and a plan to respond to the issue. This plan should include clear lines of communication and responsibilities. The primary objective is to implement solutions that facilitate a quick return to regular operations and provide insights from the

incident. Integrating preventive measures into your workflow is a pivotal strategy. The goal is to either prevent the same issue from happening again or lessen its effects on performance if it isn't preventable.

Prepare for issues

The ideal response to live-site performance issues is precise and fast. Precision and speed in performance remediation require preparation. To effectively respond to live performance issues, it's crucial to monitor key performance metrics, identify the root cause of the issues, and implement appropriate solutions or optimizations. To take these steps, you might need to analyze workload logs, conduct performance testing, optimize code or configurations, and scale resources. The following examples outline a few critical areas of preparation:

- *Have accurate architecture diagrams.* Your architecture diagrams should include all components and show how they interact. Visual representation can help identify bottlenecks and single points of failure that can lead to performance degradation or unavailability. Ideally, you catch and remove these issues before they cause problems, but having an up-to-date diagram can help you pinpoint issues in high-stress moments.

- *Check data access.* Data and logs from monitoring processes are critical for responding to performance issues in real time and conducting root cause analyses. But it's important to maintain the integrity and confidentiality of the data. Responding to live-site performance issues often requires access to underlying data that might not be normally accessible. You need to ensure that personnel have access to the data that they need when issues arise. But you should only grant time-restricted, least-privilege access, and you should limit that access to authorized personnel.

- *Set automatic alerts.* Alerts can help you identify and address issues as soon as they occur. Alerts should generate notifications when workload performance deviates from performance baselines. Over time, you should tweak alert configurations to avoid generating too many or too few notifications. The monitoring solutions that you use need to collect enough data to generate alerts. These alerts should align with performance targets and established baselines. You should avoid generating alerts on issues that are relevant to your goals. Examples of alerts include degradations in CPU usage, memory, response times, and database performance.

Create a triage plan

Creating a triage plan involves devising a structured approach to identify, escalate, analyze, prioritize, and communicate live-site performance issues. A triage plan is a strategy for responding to live performance issues. It ensures that performance disruptions are addressed promptly and effectively, with clear roles and procedures. Most performance issues don't merit disaster recovery protocols, but they can affect workload functionality enough to require triage planning. A well-documented triage plan ensures all team members are aligned and can act swiftly, minimizing the impact on users and workloads. A triage plan should include the following components:

- *Identification and monitoring*: Implement a system to identify and monitor performance issues in real time. You should have a list of the contact information of people who are capable of making decisions or escalating issues to higher levels. The plan should also identify roles and responsibilities. It needs to document which accounts gain access to protected information and for how long.

- *Escalation process*: Define a clear escalation process to ensure that performance issues are escalated to the appropriate teams or individuals in a timely manner. The process definition should include contact information and guidelines for escalating issues.

- *Root cause analysis*: Develop a process for conducting a root cause analysis to identify the underlying cause of each performance issue. The process should involve analyzing logs and performance metrics and conducting diagnostic tests to pinpoint the source of each problem.

- *Prioritization*: Establish a prioritization framework to determine the severity of performance issues and prioritize them based on their effect on the workload and users.

- *Communication*: Create a communication plan to keep stakeholders informed about the status of performance issues and the progress of their resolution. Consider regular updates, status reports, and clear communication channels.

- *Documentation*: Document the triage plan, including all its steps, processes, and best practices. This documentation should be easily accessible to the team members who are involved in responding to performance issues.

Develop methods to identify and resolve issues

Resolving live performance issues involves identifying and addressing any factors that can cause performance degradation or inefficiencies in a live workload. Data that you collect during monitoring is invaluable when you investigate and resolve performance-related incidents. This data provides a historical record of performance metrics. When

you have monitoring data available, you can analyze root causes and identify contributing factors. You should use all relevant monitoring data to understand and fix each performance issue.

Use root cause analysis

Root cause analysis requires hypothesis testing. After you review monitoring data, you should list potential causes of the performance issue and test them. To conduct a root cause analysis on a live performance issue, you can follow these steps:

1. *Gather information.* Collect as much information as possible about the performance issue. Examples include error messages, logs, performance metrics, and any other relevant data.

2. *Define the problem.* Clearly define the problem by identifying the symptoms and the effect that the problem has on the workload or users.

3. *Investigate potential causes.* Narrow down the scope of the analysis by identifying the specific component or area of the workload where the performance issue is occurring. Identify potential causes of the performance issue based on the gathered information. This process can involve analyzing code, configuration settings, infrastructure, or external dependencies.

4. *Correlate data.* Dive deeper into the collected data to identify patterns, anomalies, or correlations that might contribute to the performance issue. Data correlation is key to identifying performance issues and causes. It can involve reviewing logs, analyzing performance metrics, and conducting tests.

5. *Test hypotheses.* Formulate hypotheses based on the potential causes that you identify. Conduct tests to validate or refute your hypotheses. You should use a test environment to see whether you can replicate the error.

6. *Implement solutions.* Once you identify a root cause, develop and implement solutions to address the performance issue.

7. *Monitor and validate.* After you implement the solutions, continuously monitor the workload to ensure that the performance issue is resolved. Validate the effectiveness of the solutions by monitoring performance metrics and user feedback.

>  **Tradeoff**: The steps of a root cause analysis, such as identifying possible causes, testing hypotheses, and documenting the analysis, can be time consuming. To correlate performance issues, you also need to collect and store data. The required

> time and infrastructure can add significant work to the operations teams and cost to the workload.

>  **Risk**: If you perform a root cause analysis without proper security guardrails, there's a risk that you expose sensitive information when you provide access to logs and data.

Engage vendor support

Vendor support can be an essential step when you deal with ongoing performance issues. Vendors have the expertise, tools, resources, and experience to help fix issues with their products. Your support agreement with your supplier determines the level of support a vendor provides.

It's often best to work in parallel with vendors. You should create a plan to have some team members collaborate with vendor support while others continue to triage and fix performance issues. Vendor support teams can also make suggestions on how to help prevent and automate responses to similar events.

You need to have contact information available for your personnel. Vendors might also need access to data to effectively engage in problem-solving. You need to have a plan in place for authenticating and authorizing external or guest accounts to access monitoring data.

Learn from findings

After you fix a live-site performance issue, you need to review what happened. The goal is to learn from performance issues, not just identify problems. The best way to learn is through documentation. Document each issue and explain how to fix it. If a vendor helped, work with the vendor to enhance your documentation, train your team, and modify your workload accordingly.

The documentation should indicate how to prevent each problem from happening again. One way to avoid recurring problems is to introduce automation to respond to common issues. Automation should add self-healing and self-prevention qualities to a workload. Along with the automation, you can create refined alerts that help you respond early to performance issue indicators.

Azure facilitation

Developing methods to identify and resolve issues: Azure provides several tools to help you respond to live performance issues:

- Azure Monitor is a comprehensive monitoring solution that provides insights into the performance and health of your applications and infrastructure. Monitor offers features such as metrics, logs, alerts, and dashboards to help you monitor and diagnose performance issues.
- Application Insights is an application performance management (APM) service that helps developers and DevOps professionals monitor live applications. It automatically detects performance anomalies, collects application-level logs and events, and provides analytics tools to diagnose issues.
- Log Analytics is a service that collects and analyzes log data from various sources, including applications, virtual machines, and Azure resources. When you use Log Analytics, you can query and analyze log data to gain insights into the performance and behavior of your applications.

Related links

- Recommendations for self-healing and self-preservation

Performance Efficiency checklist

Refer to the complete set of recommendations.

Performance Efficiency checklist

Recommendations for continuous performance optimization

Article • 11/14/2023

Applies to this Azure Well-Architected Framework Performance Efficiency checklist recommendation:

> **PE:12** **Continuously optimize performance. Focus on components that show deteriorating performance over time, such as databases and networking features.**

This guide describes the recommendations for continuous performance optimization. Continuous performance optimization is the process of constantly monitoring, analyzing, and improving performance efficiency. Performance efficiency adapts to increases and decreases in demand. Performance optimization needs to be an ongoing activity throughout the life of the workload. Workload performance often degrades or becomes excessive over time, and factors to consider include changes in usage patterns, demand, features, and technical debt.

Definitions

Term	Definition
Data tiering	A storage strategy that involves categorizing data based on its access frequency and storing it on storage tiers accordingly.
Technical debt	The accumulated inefficiencies, suboptimal design choices, or shortcuts intentionally taken during the development process to deliver code faster.
Time-to-live	A mechanism that sets an expiration time for data.

Key design strategies

Performance efficiency is when workload capacity aligns to actual usage. A workload that overperforms is as problematic as one that underperforms. The tradeoffs differ. Overperformance affects cost optimization. Poor performance affects users. The key to performance efficiency is monitoring, adjusting, and testing over time. You need to regularly review performance metrics and make adjustments as necessary to ensure that the workload is efficient. Testing all changes pre- and post-implementation is required to reach performance targets.

Develop a performance culture

A performance culture is an environment in which continuous improvement is expected and the team learns from production. Performance optimization requires specialized skills. Workload teams need the right skills and mindset to optimize their performance to meet increases and decreases in demand. You also need to allocate their time to support the required monitoring and remediation of performance issues as they arise. These teams need clear expectations. For example, performance targets, baselines, and deviation thresholds (how far from baseline is acceptable) need to be highly visible and socialized.

>  **Tradeoff**: Continuous performance optimizations require a team that has the right skills and time to find and fix performance issues. Dedicating personnel to performance adds operational cost. If you have limited personnel resources, continuous performance optimization could take time away from other operational tasks.

Evaluate new platform features

Evaluating new platform features involves examining the new functionalities and tools of a platform that can improve performance efficiency, such as optimized storage solutions, caching mechanisms, or resource management tools. New platform features can open avenues for enhancing performance efficiency. Keep your platform and tools up-to-date to ensure you're using the latest innovations and best practices. Consistently monitor feedback and performance metrics from these new additions to refine your approach.

Prioritize optimization efforts

Proactively optimizing performance means taking proactive measures to improve and enhance the performance of the workload before any performance issues arise. Using proactive measures involves identifying potential bottlenecks, monitoring performance metrics, and implementing optimizations to ensure that the workload operates efficiently and meets the desired performance goals. Based on the analysis of deteriorating components, critical flows, and technical debt, you can implement performance optimizations specific to each area. Improvements might involve code changes, infrastructure adjustments, or configuration updates.

Prioritize deteriorating components

A workload often has components such as databases and networking components that are prone to performance degradations over time. As the workload evolves and usage patterns change, these changes often affect the performance of individual components in the workload. Increased data in databases can lead to longer query run times and slower data retrieval. Changes in usage patterns might result in suboptimal query design. Queries that were once efficient can become inefficient as the workload evolves. Inefficient queries can consume excessive resources and degrade database performance. Increased workload usage can lead to higher network traffic, causing congestion and latency issues.

It's important to make continuous efforts to optimize the performance of these components. Proactively identify and address performance issues in your workload. By prioritizing known deteriorating components, you can proactively address potential performance issues and ensure the smooth operation of your workload. It might involve implementing performance tuning techniques, optimizing resource allocation, or upgrading hardware or software components as needed.

Prioritize critical flows

Critical flows are the most important and high-priority processes or workflows in the workload. By prioritizing these critical flows, you ensure that the most essential parts of the workload are optimized for performance. Knowing which flows are critical helps prioritize optimization efforts. Optimizing the performance efficiency of the most important areas of your application provides the highest return on investment. You should monitor critical flows and the most popular pages. Look for ways to make them more efficient.

Automate performance optimization

Automation can eliminate repetitive and time-consuming manual processes, allowing them to be performed efficiently. Automation reduces the chances of human error and ensures consistency in running optimization tasks. By automating these tasks, you can also free up people to focus on more complex activities and activities that add value. You can apply automation to various tasks, such as performance testing, deployment, and monitoring:

- *Automated performance testing*: Use automated performance testing tools like JMeter, K6, or Selenium to simulate different workloads and scenarios.
- *Automated deployment*: Implement automated deployment processes to ensure consistent and error-free deployments. Use CI/CD tools to automate the

deployment process. These tools can help you identify performance bottlenecks as you use them to test against endpoints, check HTTP statuses, and even validate data quality and variations.

- *Monitoring and alerting*: Set up automated monitoring and alerting systems to continuously monitor performance metrics and detect any deviations or anomalies. When performance issues are detected, automated alerts can be triggered to notify the appropriate teams or individuals.

- *Incident management*: Implement an automated incident management system that can receive alerts, create tickets, and assign tickets to the appropriate teams for resolution. These steps help ensure that performance issues are promptly addressed and assigned to the right resources.

- *Automated diagnostics*: Develop automated diagnostic tools or scripts that can analyze performance data and identify the root causes of performance issues. These tools can help pinpoint specific areas or components of the system that are causing performance problems.

- *Automated remediation actions*: Define and implement automated remediation actions that can be triggered when specific performance issues are detected. These actions can include restarting services, adjusting resource allocation, clearing caches, or implementing other performance optimization techniques.

- *Self-healing systems*: Build self-healing capabilities into your system by automating the recovery process for known performance issues. This capability can involve automatically fixing or adjusting the system configuration to restore optimal performance.

Address technical debt

Technical debt refers to the accumulated inefficiencies, suboptimal design choices, or shortcuts taken during the development process that can affect performance. Technical debt, unclear code, and overly complex implementations can make performance efficiency more difficult to attain. Addressing technical debt involves identifying and resolving these issues to improve the overall performance and maintainability of the workload. This work might include refactoring code, optimizing database queries, improving architectural design, or implementing best practices. Perhaps you introduced technical debt to meet a deadline, but you need to address the technical debt as you optimize performance efficiency over time.

Optimize databases

Continuously optimizing databases involves identifying and implementing optimizations to ensure that databases can handle loads, deliver fast response times, and minimize resource utilization. By regularly optimizing databases, you can improve application performance, reduce downtime, and enhance the overall user experience.

- *Optimize database queries*: Poorly written SQL statements can degrade database performance. Inefficient JOIN conditions can cause unneeded data processing. Complex subqueries, nested queries, and excessive functions can reduce running speed. Queries that retrieve too much data should be rewritten. You should identify your most common or critical database queries and optimize them. The optimization helps ensure faster queries.

- *Maintain indexes*: Evaluate your indexing strategy to ensure that indexes are properly designed and maintained. Index maintenance includes identifying unused or redundant indexes and creating indexes that align with the query patterns. Database indexes help accelerate data retrieval operations. For relational databases, you need to monitor index fragmentation. You should rebuild or reorganize indexes regularly. For nonrelational databases, you need to pick the correct indexing policy for your workload. Use automatic tuning on databases where available. These features include automatically creating missing indexes, dropping unused indexes, and plan correction. For more information, see Maintaining indexes to improve performance.

- *Review model design*: Review the data model to ensure you optimize it for the specific requirements of the application. Improving query performance and data retrieval might involve denormalization, partitioning, or other techniques.

- *Optimize database configuration*: Optimize database configuration settings such as memory allocation, disk I/O, and concurrency settings to maximize performance and resource utilization.

Optimize data efficiency

Optimizing data efficiency is the process of ensuring that data is stored, processed, and accessed in the most efficient way possible. Data tiering and using time-to-live (TTL) are techniques that can be used to optimize data efficiency. You can apply these techniques in various data storage scenarios, such as databases, file systems, or object storage.

- *Use data tiering*: Data tiering involves categorizing data based on its importance or frequency of access and storing data in different tiers accordingly. Setting up data tiering allows for more efficient use of storage resources and improves performance. Frequently accessed or critical data can be stored in high-

performance tiers, while less frequently accessed or less critical data can be stored in lower-cost tiers. The goal is to review data usage over time to ensure data is in the correct tier. As data priorities change, data should move from one tier to another.

- *Implement time-to-live*: Time-to-live is a mechanism that sets an expiration time for data. Time-to-live allows data to be automatically deleted or archived after a certain period, reducing storage requirements and improving data management. By setting an appropriate time-to-live, you allow unnecessary data to be removed, freeing up storage space and improving overall efficiency. Session data, temporary files, and cache data are frequent targets for the time-to-live. Database entries can also have a time-to-live.

>  **Risk**: A time-to-live that's too short can create performance issues.

Azure facilitation

Automating performance optimization: Azure Advisor provides automatic performance recommendations based on workload telemetry. You should review and address these recommendations regularly. Azure Monitor provides real-time insights into the performance of your system and allows you to set up alerts based on specific performance metrics. Azure Log Analytics provides automated diagnostics and analysis on collected logs and metrics. Tools like Azure Application Insights provide insights and recommendations for optimizing performance.

To automate remediation, use automation tools or scripts to execute remediation actions automatically when the alerts are triggered. You can use Azure Automation, Azure Functions, or custom automation solutions.

Azure lets performance testing to simulate different user scenarios and workloads. Automated testing can help you identify performance bottlenecks and optimize your system accordingly. Tools like Azure DevOps can automate performance testing.

Optimizing databases: The SQL family of products has many built-in features that allow you to monitor and remediate SQL database performance. You should use these features to maintain database performance. Azure SQL Database has an automatic tuning feature that continuously monitors and improves queries. You should use this feature to improve SQL queries automatically.

You can customize your indexing policies by using the features of Azure Cosmos DB. Customize the policies to meet the performance needs of your workload.

Optimizing data efficiency: Data tiering allows you to store data in different tiers based on its access frequency and importance. It helps optimize storage costs and performance. Azure provides different storage tiers, such as hot, cool, and archive tiers for blob data. Hot tiers are optimized for frequently accessed data, cool tiers are for infrequently accessed data, and archive tiers are for rarely accessed data. By using the storage access tier best suited to your data, you can ensure efficient data storage and retrieval.

Related links

- Optimize index maintenance to improve query performance and reduce resource consumption
- Improve the performance of Azure applications by using Azure Advisor
- Automatic tuning in Azure SQL Database and Azure SQL Managed Instance
- Indexing policies in Azure Cosmos DB

Performance Efficiency checklist

Refer to the complete set of recommendations.

Performance Efficiency checklist

Azure Well-Architected Framework workloads

Article • 11/01/2024

In the context of the Azure Well-Architected Framework, the term *workload* refers to a collection of application resources, data, and supporting infrastructure that function together to achieve defined business outcomes. A workload consists of components and also development and operational procedures.

Architects design workloads, and a workload team implements them. A workload is designed and implemented to achieve functional and nonfunctional business requirements. Workloads can be classified into many types.

Typical criteria for workload classification include:

- Utility, characteristics, and usage patterns of a workload, such as web applications, batch processing, and real-time analytics.
- Key influential drivers, such as technology platforms or alignment with an industry.
- Intended target audience. Examples of solutions with various audiences are internal line-of-business applications within enterprises, a purchased independent software vendor (ISV) solution, or a multitenant software as a service (SaaS) solution for public use.

Workloads that are in the same class can share similarities, including their target audience, compliance requirements, and technology stacks. The five pillars of the Well-Architected Framework, their principles, checklists, and tradeoffs are relevant for all workload classes.

The workload guidance of the Well-Architected Framework describes common priorities and tradeoffs as they pertain to specific workload classes. In the workload guidance, the pillar guidance applies to technical design principles and design areas that represent the priorities of a workload. Follow the recommendations to help set up a successful workload and to align it with the Well-Architected Framework.

What is a Well-Architected Framework workload?

The design and operations of any workload have to contend with the five architectural pillars: Reliability, Security, Cost Optimization, Operational Excellence, and Performance

Efficiency.

Expand table

To create a successful workload, develop it in accordance with the Well-Architected Framework principles, which are based on the following ideals.

A Well-Architected Framework workload:

- Has functional and nonfunctional requirements that are defined and prioritized to achieve a goal.
- Is designed so you can achieve those requirements by using resources and incorporating design patterns and tradeoffs.
- Is built and operated to the specifications of a design and purpose.
- Is measured by how adequately it achieves its purpose.
- Can adapt as its purpose is refined or changed.
- Is just as reliable as it needs to be.
- Is just as secure as it needs to be.
- Delivers a sufficient return on investment.
- Is developed and operated responsibly.
- Accomplishes its purpose within an acceptable time period.

A collaboration between the workload team and central teams of an organization must create a workload with the preceding characteristics. The following sections describe these teams and their functions.

Workload team

Create a workload team that has team members with a wide range of technical and business disciplines. The primary focus of all team members should be the success of the workload.

Expand table

Examples of workload team members	
Application security engineers	DevOps engineers
Business stakeholders	Infrastructure engineers
Cloud developer or software engineers	Product managers or owners
Cloud solution architects	Quality assurance (QA) engineers
Data scientists or analysts	Support team members
Database administrators	

Centralized teams and stakeholders

Centralized teams often support the workload team. They provide support functions and apply governance for many or all cloud workloads within an organization. Centralized teams focus on organizational success, which is achieved in part by the success of the organization's workloads. They provide services, guidance, and guardrails for workloads.

Expand table

Examples of centralized teams and team members	
Business intelligence analysts	Finance analysts
Business stakeholders	Infrastructure engineers
Cloud center of excellence (CCoE) board	Legal and compliance officers
Cloud platform team	Network engineers
Cybersecurity analysts	Procurement specialists
Database administrators	Project managers
Enterprise architects	

A Well-Architected Framework workload team focuses on workload outcomes. They coordinate with and benefit from the specialized support from centralized team members.

Shared responsibility model

A workload needs to be deployed and used in order to deliver value. As part of the workload team, you have a responsibility to design, implement, and deploy your workload in a way that creates value to your organization.

Workloads exist within the context of your organization. An organization often has regulated governing and authority roles. Your workload team has the responsibility to design, implement, and deploy a workload within the foundation of your organization.

In accordance with the Cloud Adoption Framework for Azure, standardize your workload's cloud resources. Rigorously apply standardization to provide a governed platform to help with onboarding workload teams. Apply this governance in accordance with your organization's cloud operating model.

You can use Azure landing zones to help you perform standardization. Platform landing zones and application landing zones are available in Azure. Deploy your workload in an application landing zone.

Your organization might have a cloud platform offering that's rigorously formalized and fully aligns with Azure landing zones. Or your organization might have a different

adoption strategy or no implementation. If there's no implementation, workload teams are nearly fully autonomous entities.

For any platform and governance that your organization uses, you must apply the principles of the Well-Architected Framework to your workloads. The Well-Architected Framework often references Azure landing zones, but it isn't dependent on a specific platform implementation. The Well-Architected Framework pillars, principles, checklists, and guides are for all cloud platforms and most workload types.

Fulfill requirements

Throughout the Well-Architected Framework, such as the core pillars and the workload guidance, recommendations coincide with the obligation of the workload. Recommendations don't usually imply what team member or team facilitates those obligations. You can determine who should perform each action. Perform workload-level mapping to determine your team's roles and responsibilities related to the topology, workload type, and criticality.

The direct workload team handles most workload requirements. Some requirements are handled as a joint effort with centralized teams. For example, the implementation choices might be based on guardrails that a centralized team sets. Or a centralized team might exclusively handle the implementation choices.

Your workload team must build a working relationship with other teams to help codeliver on workload goals. If you outsource components or responsibilities, you must successfully deliver on those obligations.

Learn the constraints

A centralized team supports diverse workloads based on the team's core capabilities and core infrastructure. To provide this support on an organizational scale, the centralized team might implement uniformity and constraints on the service offered or the infrastructure. As you design your workload, it's critical that you understand those constraints and, where possible, partner with enterprise architects who know those constraints. Learn from prior implementations as much as possible.

Every platform governance implementation is different, but the following constraints are common for many workloads:

- Allowlists for cloud resources
- Configuration mandates for cloud resources
- Regional allowlists for cloud resources and cross-premises connectivity availability

- Limited or no platform support outside of business hours
- Patching requirements
- Specific hub-spoke implementation, which drives Domain Name System (DNS) and private endpoint implementations
- Supply chain control requirements

Explicitly communicate requirements

If your workload requirement is faced with a constraint or a service-level agreement (SLA) that doesn't clearly define a core capability or infrastructure offering, treat that situation as a risk. To address this risk, your workload team must provide clarity to the other teams about how the concern affects the workload. You might have to change the workload requirements, design, or implementation, or change the infrastructure offering.

When you understand the platform team's obligations related to organizational directives and your workload team's obligations, you can communicate workload requirements with realistic expectations and recommendations.

Communicate common workload requirements

Every platform partnership is different, but the following areas are common topics in shared responsibility conversations:

- Compliance and legal requirements
- Networking specifics, such as the need for static ingress or egress IP addresses
- Observability requirements to provide live site triage that's effective
- Performance requirements, such as network throughput, availability of cloud resources, or regional availability
- Expectations for public internet access from an egress and ingress perspective
- Service-level objectives (SLOs) or SLAs that are offered to the workload's users
- The availability of technical support

Look for unified wins

Shared responsibility isn't just about tradeoffs, constraints, and compromise. Platform teams often have highly specialized skills and dedicated budgets that can augment beyond what an individual workload team can sustain. Consider the following examples.

Security specialists. Your workload might have a secure development lifecycle. As a centralized security team performs secure development tasks at scale across your

organization, it might perform routine penetration testing that's above and beyond your efforts. It might also help with planning and performing an incident response strategy.

Enterprise architecture guidance. You can save time and effort if you align with an enterprise architecture team's patterns and practices because the team has already streamlined the processes. You can also prevent rework if a solution isn't possible within the partnership without negotiation.

Big-ticket expenditures. Platform teams often host components or services that are too expensive or too extensively managed for an individual workload team. Platform teams can afford these components and services because they divide the cost across workloads.

Often these services or centralized platforms are offered as mere showback, so they help keep the workload cost optimized. And when they're offered as chargeback, they're often cheaper because of economies of scale and centralization.

Platform teams often provide self-service options to workload teams for various activities. For example:

- Providing a documentation repository for self-guided education
- Onboarding to cost management via specific resource tagging
- Offering subscriptions via a formal subscription-vending process

Explore self-service and platform engineering options that might be suitable for your workload.

Share successes and challenges

Shared responsibility with other teams also means sharing successes and challenges of a workload. When your workload meets its obligations and obtains the intended value, share that with your partnering teams. Tell them how they contributed to the workload's success. When your workload isn't meeting its obligations, share what isn't working and collaborate and recalibrate to get back on track.

Platform teams also have obligations and success criteria. You should expect your partners to tell you whether your workload works well with an offering or if it's at risk of being a noisy neighbor.

Strive for continuous improvement

A theme across all Well-Architected Framework pillars is continuous improvement. Adopt a progressive mindset. You might deal with new approaches to existing problems, adopt new technology, address new requirements, or operate under new constraints. As your workload improves over time, expect the same mindset from your partnering teams. However, every improvement opportunity also means changes and should be supported by a proper management process.

Workload teams have an obligation to communicate with platform teams about proposed changes to workload requirements that might have an effect on the platform team's services. Likewise, platform teams have an obligation to include their workload partners in change control processes and clearly communicate the impactful platform changes. Establish a regular communication cadence with partners to learn about and share how a product evolves.

Achieve a successful outcome

Workloads have many expectations from users, shareholders, regulatory bodies, employees, the center of excellence, and chief experience officers. Expectations can set the directional compass spinning. The Well-Architected Framework provides clarity related to the design and implementation by offering explicit rationalizations for architectural decisions to achieve a successful outcome. Develop a successful workload, and share in that success with your organization.

Feedback

Was this page helpful? Yes No

AI Workload Documentation

AI workloads use non-deterministic behavior, making them more adaptable to complex situations compared to traditional workloads. They combine code and data into models, creating unique experiences. This guidance focuses on the architectural challenges of designing AI workloads, including non-deterministic functionality, data and application design, and operations.

Get started

OVERVIEW

What is an AI workload?

CONCEPT

Design methodology

Design principles

Workload personas

Technical design areas

CONCEPT

Application design

Application platform

Training data design

Grounding data design

Data platform

Operational design areas

CONCEPT

Workload operations

MLOps and GenAIOps

Testing and evaluation

Responsible AI

Reference examples

ARCHITECTURE

Baseline OpenAI end-to-end chat reference architecture

Related services

ARCHITECTURE

Azure Machine Learning service guide

Azure OpenAI service guide

Assessment

HOW-TO GUIDE

AI assessment tool

Azure Virtual Desktop workload documentation

Run Windows desktops and applications on Azure from any device and location.

Get started

OVERVIEW

What is an Azure Virtual Desktop workload?

CONCEPT

Design principles

Integration with Azure landing zones

Design areas

CONCEPT

Application delivery

Networking and connectivity

Monitoring

Security and identity and access management (IAM)

Operational procedures

Business continuity

Storage

Reference examples

ARCHITECTURE

Migrate end-user desktops to Azure Virtual Desktop

Azure Virtual Desktop for the enterprise

Reference implementations

DEPLOY

Enterprise-scale support for Microsoft Azure Virtual Desktop

Learn

TRAINING

Deliver remote desktops and apps with Azure Virtual Desktop

Plan an Azure Virtual Desktop implementation

Implement an Azure Virtual Desktop infrastructure

Assessment

HOW-TO GUIDE

Azure Virtual Desktop assessment tool

Azure VMware Solution workload documentation

Relocate legacy application virtual machines to Azure VMware Solution as a staging area for the first phase of your migration and modernization strategy.

Get started

OVERVIEW

What is an Azure VMware Solution workload?

CONCEPT

Design principles

Integration with Azure landing zones

Design areas

CONCEPT

Infrastructure

Applications

Networking

Monitoring

Security

Operations

Reference examples

ARCHITECTURE

Baseline Azure VMware Solution reference architecture

Azure VMware Solution landing zone accelerator

Reference implementations

DEPLOY

Azure VMware Solution implementation options

Learn

TRAINING

Introduction to Azure VMware Solution

Migrate VMware resources on-premises to Azure VMware Solution

Run VMware resources on Azure VMware Solution

Assessment

HOW-TO GUIDE

Azure VMware Solution assessment tool

Mission-critical workload documentation

Learn about building highly reliable workloads on Microsoft Azure. The articles provide architectural guidance and a prescriptive approach for designing, building, and operating mission-critical workloads. For a given set of business requirements, an application should always be operational and available. While there are many approaches to achieving high reliability, one of the goals is to accelerate adoption toward cloud native solutions to derive maximum value from the Microsoft Cloud.

Get started

OVERVIEW

What is a mission-critical workload?

Design methodology

CONCEPT

Architecture pattern

Design principles

Design areas

CONCEPT

Application design

Application platform

Data platform

Networking and connectivity

Health modeling

Deployment and testing

Security

Operational procedures

Reference examples

ARCHITECTURE

Baseline reference architecture

Baseline with network controls

Baseline in Azure landing zones

Baseline with App Services

Learn

TRAINING

Challenge Project - Design a mission-critical web application

Design a health model for your mission-critical workload

Continuously validate and test mission-critical workloads

Video library

VIDEO

What is a mission-critical workload?

Global distribution

Define a health model

Continuously validate your workload

Use multiple subscriptions

Development environments

Zero downtime deployments

VIDEO

Demo - Continuous validation with Azure Load Test and Azure Chaos Studio

Demo - Ephemeral dev environments and automated feature validation

Demo - Monitoring and health modeling

Assessment

HOW-TO GUIDE

Mission-critical assessment tool

Architect perspectives

VIDEO

Azure Friday - Health modeling for mission-critical workloads on Azure

Azure Friday - Continuously validate and test your mission-critical Azure workloads

Azure Friday - Deploy your mission-critical workload in an Azure landing zone

Azure Enablement Show - Designing a mission-critical workload on Azure

Azure Enablement Show - Integrating a mission-critical workload with Azure landing zones

Oracle workloads on Azure IaaS documentation

When you create an Oracle workload on Azure IaaS, follow Azure Well-Architected Framework design principles and explore important design areas.

Get started

OVERVIEW

Oracle workloads on Azure infrastructure as a service (IaaS)

Design principles

Explore design areas

CONCEPT

Choose compute and storage

Design Oracle applications

Optimize business continuity and disaster recovery

Optimize security

Monitor workloads

Take the assessment

HOW-TO GUIDE

Take the assessment for Oracle on Azure IaaS

Software as a service (SaaS) Workload Documentation

Delivering SaaS on Microsoft Azure differs from other software types. This set of articles offers key insights for independent software vendors (ISVs) to build scalable, performant, reliable, and secure SaaS solutions. It provides actionable guidance based on Well-Architected best practices for building and operating SaaS on Azure at scale.

Get started

OVERVIEW

What is a SaaS workload?

Design methodology

Design principles

Technical design areas

CONCEPT

Resource organization

Identity and access management

Compute

Networking

Data

Operational design areas

CONCEPT

Billing and cost management

Governance

DevOps practices

Incident management

Assessment

HOW-TO GUIDE

Assessment tool

Additional guides

ARCHITECTURE

Multitenant solutions on Azure

Plan your journey to SaaS

SAP workload documentation

Learn how to build an SAP workload on Azure

Get started

OVERVIEW

SAP workload

Design principles

Design areas

CONCEPT

Application design

Application platform

Data platform

Networking and connectivity

Security

Operational procedures

Reference examples

ARCHITECTURE

SAP on Azure landing zone accelerator

Start small and expand with SAP HANA

Reference implementations

DEPLOY

SAP on Azure Deployment Automation Framework

Learn

TRAINING

How to use SAP on Azure Solutions

Assessment

HOW-TO GUIDE

Azure Well-Architected Review (SAP on Azure)

Sustainability workload documentation

In partnership with the Green Software Foundation, we've developed this set of recommendations for optimizing Azure workloads. This documentation helps you plan your path forward, improve your sustainability posture, and create new business value while reducing your operational footprint.

Get started

OVERVIEW

What is sustainability?

Design methodology

Design principles

Design areas

CONCEPT

Application design

Application platform

Testing

Operational procedures

Networking and connectivity

Storage

Security

AI design

Reference examples

ARCHITECTURE

Example scenario - Measure Azure app sustainability by using the SCI score

Learn

TRAINING

The Principles of Sustainable Software Engineering

Assessment

HOW-TO GUIDE

Sustainability assessment tool

Well-Architected Framework perspective on Azure services

Azure Well-Architected Framework is decision making tool to help solution architects build a technical foundation for their workloads. Consider WAF perspectives on the Azure services that are part of your solution.



GET STARTED
What is Well-Architected Framework?



ARCHITECTURE
Pillars



ARCHITECTURE
Workloads



ARCHITECTURE
Assessment tool

Browse the catalog of Azure services

Service guides help you choose and configure Azure components by highlighting key features needed for a well-architected design. While they don't list every feature, they focus on what matters most for building a solid baseline and addressing common workload challenges.

Popular



Azure App Service

Quickly create powerful cloud apps for web and mobile.



Azure Application Gateway

Build highly secure, scalable, and available web front ends in Azure.



Azure Cosmos DB

Use a fast NoSQL database with open APIs for any scale.



Azure ExpressRoute

Use dedicated private network fiber connections to Azure.



Azure Firewall

Use native firewall capabilities with built-in high availability, unrestricted cloud scalability, and zero maintenance.



Azure Front Door

Learn about the scalable, security-enhanced delivery point for global, microservice-based web applications.



Azure Kubernetes Service

Simplify the deployment, management, and operations of Kubernetes.



Azure Machine Learning

Build, train, and deploy machine learning models.



Azure OpenAI

Build intelligent apps with large language models from OpenAI with an enterprise-ready service.

Other resources

What's new?

Learn about the updates for the Azure Well-Architected Framework.

What is the Well-Architected Framework?

Learn how to use the Well-Architected Framework and audience profile.

Architecture best practices for Azure API Management

Article • 04/17/2025

Azure API Management is a management platform and gateway for APIs across all environments, including hybrid and multicloud. As a platform as a service (PaaS) solution, API Management helps support your workload's API lifecycle.

This article assumes that as an architect, you've reviewed the integration services decision tree and chosen API Management as the integration service for your workload. The guidance in this article provides architectural recommendations that are mapped to the principles of the Well-Architected Framework pillars.

> **Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern along with design strategies localized to the technology scope.
>
> Also included are recommendations for the technology capabilities that can help materialize those strategies. The recommendations don't represent an exhaustive list of all configurations that are available for API Management or the back-end API servers of your workload. Instead, they list the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept or to optimize your existing environments.
>
> Foundational architecture that demonstrates the key recommendations: API Management landing zone architecture.

Technology scope

This review focuses on the interrelated decisions for the following Azure resource:

- Azure API Management

The scope of this guide is the API Management service, primarily the gateway component (data plane), which proxies API requests from client applications to back-end APIs hosted on various platforms or cross-premises locations. The workload's architecture should account for the API Management control plane and related components such as client applications that access the gateway and the back-end APIs that process routed requests. It also integrates

supporting Azure services, including networking, monitoring, identity management, and other capabilities.

This guide doesn't cover Azure API Center. It addresses API-level topics as they relate to API Management instead of providing a well-architected perspective on API design considerations.

> **Note**
>
> Not all recommendations apply to all service tiers of API Management. Many recommendations in this guide focus on the Standard v2 and classic Premium tiers of API Management, which are the recommended production tiers for most enterprise workloads.

> **Important**
>
> The Premium v2 tier with enterprise capabilities is in preview. To determine whether your design should rely on early access features or generally available capabilities, evaluate your design and implementation timelines in relation to the available information about Premium v2's release and migration paths.

Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover fast from failures**.

Reliability design principles provide a high-level design strategy applied for individual components, system flows, and the system as a whole.

Design checklist

Start your design strategy based on the design review checklist for Reliability. Determine its relevance to your business requirements while keeping in mind the tiers and features of API Management and its dependencies. Extend the strategy to include more approaches as needed.

- ✓ **Evaluate gateway capabilities for reliability and redundancy:** Determine the API Management tier and features that are needed to meet the workload's reliability requirements for each environment.

Evaluate gateway redundancy features, including availability zones, multiple gateway units, multiple regions, and workspaces. These features are all supported in the Premium tier. The Developer tier, which doesn't include a service-level agreement (SLA), isn't suitable for production workloads. Consider the tradeoffs of adopting features such as external caching that can introduce potential points of failure and performance bottlenecks.

- ✓ **Review observability capabilities:** Consider the service's observability capabilities, including Azure Monitor logs and metrics, Application Insights, built-in analytics, and built-in diagnostics. Use these capabilities to monitor the reliability signals of your workload.

  For example, consider using Azure Monitor alerts to notify you of potential problems with the API Management gateway or its dependencies.

- ✓ **Review scaling strategies:** Define criteria for scaling out the gateway by adding units to maintain service reliability. Consider whether to scale based on requests, exceptions, or both. Evaluate the impact of scaling the gateway component on other components of the workload. For example, its effect on network address space and the scaling capabilities of the back ends.

- ✓ **Isolate critical workloads:** Determine if compute isolation is needed to fulfill workload requirements, like data sovereignty or performance optimization, in your APIs. Use dedicated gateways for critical APIs and shared gateways for less critical APIs.

  Choose an isolation approach, like using a dedicated workspace gateway or a separate API Management instance. For multiple instances, keep the environments synchronized as part of your safe deployment practices.

- ✓ **Service-level objective (SLO) alignment:** Factor in the gateway's SLA scope when you set your workload's SLOs. The service provides its own SLA, but you also need to account for the anticipated reliability of other workload components, such as the API back ends.

- ✓ **Address back-end faults:** Plan for both expected and unexpected back-end faults. Test client experiences in these scenarios. Evaluate gateway policies to improve resiliency and enhance the client experience. Focus on quotas, rate limits, retry policies, back-end circuit breakers, load balancing, and exception handling as documented in your API specifications.

- ✓ **Define testing strategies:** Use a testing solution such as Azure Load Testing from within your network to reflect actual production workloads. Don't rely on published throughput or other estimates which might not apply to your workload.

- ✓ **Plan for disaster recovery (DR):** Review options for backing up and restoring the gateway infrastructure and APIs. Built-in backup and restore capabilities might be useful in some scenarios. But customer-managed options such as APIOps tooling and infrastructure as

code (IaC) solutions can also be considered. Develop strategies for maintaining other system data, including user subscriptions.

Recommendations

These reliability recommendations can apply either to the service itself or to the traffic that flows through APIs and their policies. The *(Service)* or *(API)* designators indicate whether a recommendation targets the service or the APIs.

Expand table

Recommendation	Benefit
(Service) Enable zone redundancy in the Premium tier and have a minimum of three units deployed. In this configuration, under normal operating conditions, all scale units in all configured zones are active and serve gateway traffic. In any active-active scenario, plan to support scaling out in the remaining active zones to handle traffic that units originally processed in the faulted zone.	Resiliency is ensured during a datacenter outage within a region. During a complete datacenter loss, API traffic will continue to flow through the remaining units deployed in other zones.
(Service) Enable automatic scale-out based on traffic demands. In multiregion deployments, secondary regions don't have automatic scale-out or scale-in capabilities. You need to implement a custom function or Logic App that activates in response to Azure Monitor metric alerts to manage unit adjustments based on demand.	Sufficient resources are guaranteed to meet demand from API clients, which prevents failures because of insufficient capacity.
(Service) Use a multiregion topology to support resiliency against a complete regional failure. This approach requires you to coordinate with other components in your workload and to understand their planned failover characteristics. In any active-active scenario, plan to support scaling out in remaining active regions to handle traffic that the now-inactive region originally processed. Ensure that your multiregion topology aligns with any compliance requirements for data in transit or data in cache residency.	Robust resiliency is provided during a complete regional outage, which helps ensure uninterrupted API traffic through units deployed in other regions.

Recommendation	Benefit
(Service) Isolate unrelated APIs with workspaces or additional API Management instances.	The impact of failures caused by misconfiguration or outages is minimized by separating APIs across different compute instances.
(API) Thoroughly test policy expressions and logic and pair them with resilient error handling techniques in API management policies.	Client experience is enhanced by preventing new sources of failures at the gateway and providing graceful degradation or safe transient fault handling.
(Service & API) Collect reliability metrics. Typical API reliability metrics include: - Rate limits and quotas violations. - Error rate based on HTTP status codes. - Request rate deviations from baseline. - Health checks, including dependency health.	Deviations from expected behavior and past baselines are identified. This data can be used to trace root causes such as changed user behavior, interference from routine operations, unexpected impacts from new features, or unplanned faults in the workload.
(Service & API) Use built-in backup and restore capabilities built into API Management as part of your DR playbook. Supplement those features with your IaC artifacts and APIOps processes for a robust solution that can handle various scenarios, including recovery coordination with dependencies and back ends.	Business continuity is ensured by allowing recovery of the API gateway and restoration of defined APIs after a complete system loss.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload.

The Security design principles provide a high-level design strategy for achieving those goals by applying approaches to the technical design in protecting the API Management gateway.

> ⓘ **Note**
>
> The checklist and recommendations in this section focus on securing the API Management gateway resource. Securing the APIs themselves is only briefly addressed. For more information, see Mitigate OWASP API security top 10 in API Management.

Design checklist

Start your design strategy based on the design review checklist for Security and identify vulnerabilities and controls to improve the security posture. Extend the strategy to include more approaches as needed.

- ✓ **Establish a security baseline:** Review the security baseline for API Management and incorporate applicable measures in your baseline.
- ✓ **Protect the deployment pipeline:** Identify the individuals and role-based access control roles that are required to manage the service platform, continuous integration and continuous deployment (CI/CD) pipelines, and the individual APIs. Ensure that only authorized individuals have access to manage the service and its APIs.
- ✓ **Evaluate data sensitivity:** If sensitive data flows through API requests and responses in the API Management gateway, ensure its protection throughout its entire lifecycle. Account for varying data protection requirements across different regions. Evaluate service features such as multiple regions to isolate specific data. Also, confirm whether your caching strategy aligns with these isolation requirements.
- ✓ **Develop segmentation strategies on shared gateways:** If your API Management instance hosts APIs for multiple workload teams, use workspaces to segregate roles, networks, and possibly gateways. This approach ensures that each team has appropriate access and control over the APIs that they manage while restricting access to the APIs of other teams.
- ✓ **Consider network controls:** Identify requirements and options for isolating or filtering inbound and outbound gateway traffic by using virtual networks. Determine whether access to the gateway can be restricted through Azure Private Link or if public access to the gateway is necessary. Assess whether the architecture should include a web application firewall, such as Azure Web Application Firewall in Azure Application Gateway or Azure Front Door, to achieve the required network isolation and filter network traffic.
- ✓ **Consider capabilities for API authentication and authorization:** Evaluate the use of identity providers like Microsoft Entra ID, which includes built-in groups, or Microsoft Entra External ID to screen gateway requests and enable OAuth authorization for back-end APIs.
- ✓ **Encrypt network traffic:** Identify the most secure Transport Layer Security (TLS) protocol versions and ciphers that your workloads can support. Don't require insecure TLS versions. Use TLS 1.3 when supported by your clients.
- ✓ **Perform threat modeling on API Management and reduce its surface area:** Determine whether specific API Management components, such as the direct management API or public access to the developer portal, can be disabled, restricted, or removed.
- ✓ **Identify secrets that workloads require:** Gateway operation might require certificates, API keys, or other secret values. Review the requirements and capabilities of Azure Key Vault to

store secrets and certificates. Or review the built-in API Management capabilities such as named values and managed certificates.

Recommendations

These security recommendations can apply either to the service itself or to the traffic that flows through APIs and their policies. The *(Service)* or *(API)* designators indicate whether a recommendation targets the service or the APIs.

Expand table

Recommendation	Benefit
(Service) Disable the direct management API, which is deprecated. Use Azure Resource Manager as your control plane.	The surface area of the service is reduced by removing a control plane access point.
(Service) Limit exposure of the gateway based exclusively on where legitimate clients connect from. - Use virtual network injection without a public IP address when all clients can route traffic through a virtual network. Use a network security group (NSG) to further restrict traffic to only the expected client origin IP addresses. - Use virtual network integration with Application Gateway or Azure Front Door if any traffic must come from the internet. Configure the NSG to only accept traffic from the single point of entry.	Confidentiality of network traffic is protected by restricting exposure to the source IP address ranges expected to contain legitimate clients. These restrictions block access from sources that shouldn't initiate legitimate client communication. Limiting exposure to legitimate traffic sources enhances the confidentiality, integrity, and availability of the service.
(Service) Disable developer portal if it's not in use. If the portal is in use, disable the sign-up experience, disable anonymous access, and restrict access to only trusted network locations.	The surface area of the service and the chance for misconfiguration or neglect are reduced.
(Service) Explicitly set the narrowest supported TLS versions, protocols, and ciphers for your clients and your back ends.	Versions and supported ciphers are reduced to only those options that clients and back ends support. This approach helps ensure that connections prioritize the highest grade connections possible for confidentiality.

Recommendation	Benefit
(Service) Store custom certificates in Key Vault.	Certificate management functionality is provided by using Key Vault, which supports routine rotation and audits access to certificates.
(Service) Back ends should only accept traffic from the API gateways and should block all other traffic.	Malicious traffic is prevented from bypassing the security and other crosscutting concerns offloaded to the gateway.
(Service) For API Management instances that host APIs for multiple teams or segmented workloads, you must design a robust access control isolation strategy. Use workspaces or implement a rigorous APIOps process to achieve this strategy. Teams must only have access to the APIs that they own. They shouldn't have access to other APIs that might be hosted in the same instance.	Lateral movement by attackers from one compromised API set into other unrelated APIs is reduced.
(API) Store secrets in Key Vault and expose them to policies through named values. Don't use Key Vault to store nonsecrets. Use named value properties directly for those values.	Secret rotation is encouraged through a single management plane in Key Vault, similar to the approach used for certificates. This process ensures that API Management is updated accordingly. Named values also ensure a consistent experience for policy configuration for both secrets and nonsecrets. All secret access is also audited in Key Vault to provide an access history. Only storing secrets in Key Vault minimizes the dependency on Key Vault and doesn't turn Key Vault into an application configuration service.
(API) Use different user-assigned managed identities for different APIs by using the authentication-managed-identity policy.	Each API is enabled to have an independent identity, which supports segmentation goals through least privilege access for each API. It also provides better auditability in the back-end systems.
(API) When possible, require clients to authenticate with OAuth 2.0 flows instead of only using preshared keys, including subscription keys or client certificates.	Client identification for auditing purposes is improved, key rotation is eliminated, and the burden on clients to protect secrets is reduced compared to using preshared keys.
(API) Suppress HTTP headers from API responses by using the set-header policy that might expose implementation details.	The cost to attackers is increased by withholding implementation information.
(API) Don't use API tracing in	Sensitive data is prevented from being exposed in request

Recommendation	Benefit
production.	traces.
(API) Use Defender for APIs.	API security insights, recommendations, and threat detection are provided.
(API) Protect back-end resources by delegating key security checks in API policy, such as validate-jwt, ip-filter, validate-headers, validate-content.	The amount of nonlegitimate traffic that reaches back-end services is reduced by performing security checks at the gateway. This offloading approach helps protect the integrity and availability of those resources.
(API) Apply security development lifecycle (SDL) practices to API policy changes the same way that you apply proposed changes to application code in your workload. Policies execute with a highly privileged view of the API traffic.	Circumventing back-end protections for confidentiality, integrity, and availability by a compromised API gateway is prevented.
(Service & API) Use managed identities for service and API dependencies.	Connections to Key Vault, Azure Event Hubs, and other dependencies, such as certificates and named values, are established without relying on preshared secrets.
(Service & API) Connect to dependencies such as Key Vault, Event Hubs, and back ends over private network connections when possible.	Confidentiality of traffic is protected by not exposing the traffic beyond the private network.
(Service & API) Client traffic that targets internet-exposed APIs should first pass through a web application firewall, such as Web Application Firewall, before it reaches API Management. Also, protect public endpoints by using Azure DDoS Protection.	The amount of nonlegitimate traffic that reaches the gateway and back-end services is reduced by conducting security checks with a web application firewall. Reducing this traffic helps protect the integrity and availability of those resources.
(Service & API) Evaluate and enable all of the Azure Policy Regulatory Compliance controls that are relevant to your workload.	The API Management instance aligns with the desired posture and remains aligned as the workload evolves by having security policies in place.

Cost Optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget while meeting business requirements.

The Cost Optimization design principles provide a high-level design strategy for achieving those goals and making tradeoffs as necessary in the technical design related to API Management and its environment.

Design checklist

- ✓ **Consider the API Management cost model:** Use the Azure pricing calculator with your organization's account benefits and criteria for SLA and scalability to develop accurate cost estimates of using an API Management service tier. Determine whether a charge-back model is necessary and determine how to calculate it based on metrics, tags, and tokens.

  The service cost model is mainly influenced by the service tier, number of units, and number of gateways. Evaluate the extra costs associated with maintaining a reserve unit or implementing an active-passive DR configuration.

  If you implement workspaces, evaluate the costs of using separate versus shared workspace gateways to address the distinct API flow requirements of various API teams or stakeholders.

- ✓ **Estimate scaling costs:** Develop scaling criteria to maintain high usage of the gateway resources. Evaluate baseline costs in a pre-production environment and perform testing to model costs of scaling out based on anticipated workload traffic.

  Design a mitigation strategy to prevent misuse of your gateways, which might cause expensive scaling beyond predicated usage.

- ✓ **Evaluate service configurations and policies:** Capabilities such as rate-limit and limit-concurrency can be used as cost optimization techniques to manage request loads.

- ✓ **Optimize logic placement:** Assess whether back-end servers are more cost-effective for processing logic or if the gateway should handle the resource usage. The gateway is a strong component for implementing cross-cutting concerns, but it might perform excessive functions in certain scenarios. Identify resource-heavy request processing tasks that the gateway performs, and determine whether moving that logic to back-end servers can reduce costs.

Recommendations

These cost optimization recommendations can apply either to the service itself or to the traffic that flows through APIs and their policies. The *(Service)* or *(API)* designators indicate whether a recommendation targets the service or the APIs.

Expand table

Recommendation	Benefit
(Service) Use the least expensive tier that supports your workload's requirements. For example, choose a Standard tier instead of a Premium tier if your workload won't benefit from the added functionality in a way that justifies the return on investment (ROI).	Purchasing unused or underused capabilities is avoided.
(Service) Use nonproduction tiers or transient infrastructure in lower environments, such as development environments, proof-of-concept deployments, and initial cost modeling activities.	Resource costs are saved for environments that can remain useful without fully mirroring the production's exact configuration or uptime requirements.
(Service) Scale in when demand decreases. Configure autoscale rules or other automated processes to reduce units when gateway capacity drops below defined thresholds.	Unnecessary costs are reduced by matching capacity to demand.
(Service) Calculate any cost advantages of a federated model for API management by using workspaces to serve APIs while also providing team isolation.	The deployment and management surface is reduced. This approach aims to achieve economies of scale from time and resource purchases.
(Service) Decommission workspaces when they're no longer in use.	Spending on unused resources is avoided.
(Service) Use the built-in cache if your workload's cached data fits within the constraints of the built-in cache in your tier.	Costs involved in purchasing and maintaining an external Redis-compatible cache are avoided.
(Service) Block malicious traffic before it reaches the gateway by using network controls, DDoS protection, and web application firewalls. Specific tiers of API Management incur charges based on the number of HTTP request operations that the gateway processes. As a result, undesirable traffic, such as bot-generated requests, can increase costs. Evaluate the cost of the blocking mechanism versus the estimated cost of HTTP operation reduction to evaluate whether this approach has a ROI.	Charges incurred because of excessive malicious or nuisance HTTP operations against the gateway are reduced.
(API) Optimize the policy expressions and processing and code to avoid excessive compute resource usage, such as processor, networking, or memory.	Unnecessary deployment of more units to provide capacity for unoptimized policy implementations and code is avoided.
(API) Evaluate the cost of logic placement between your gateway, your back end, or your public entry point, such as Azure Front Door.	Load on high-cost resources is minimized by placing capabilities

Recommendation	Benefit
The same processing can often occur at any of these layers, each with its own tradeoffs. However, some layers might provide cost savings because of unused capacity available at that level. For instance, caching logic originally implemented in the back end might be more cost-effectively implemented at the gateway level by using the built-in cache. This approach reduces extra network and compute overhead on back-end services.	in the most cost-effective layer. This approach shifts workloads to layers that have spare capacity or lower-cost compute options.

Operational Excellence

Operational Excellence primarily focuses on procedures for **development practices, observability, and release management**.

The Operational Excellence design principles provide a high-level design strategy for achieving those goals for the operational requirements of the workload.

Start your design strategy based on the design review checklist for Operational Excellence for defining processes for observability, testing, and deployment related to API Management.

Design checklist

- ✓ **Review key knowledge and skills needed to operate the service:** Areas include API lifecycle, DevOps processes, networking and connectivity, monitoring and observability, and software development. This approach encompasses policy configuration, unit testing, and the creation of CI/CD pipelines.
- ✓ **Consider documentation needs:** Organizations should commit to documenting processes for service-level and API-level configuration, lifecycle operations, and the different access patterns for API teams.
- ✓ **Evaluate standard tooling** to support service operations. For example, adopt APIOps methods, such as GitOps and CI/CD, to publish APIs and manage API configurations. Use IaC tooling for service-level configuration changes. Design reusable artifacts that can seamlessly transition from development environments to production. Consider integrating a linter like Spectral, either self-managed or as integrated into an Azure service like Azure API Center, into API approval pipelines.
- ✓ **Determine key operational metrics and logs:** Review the metrics for production. These metrics include gateway capacity, CPU usage, memory usage, and the number of requests. Review logs and observability tools, such as Azure Monitor and Application Insights. Determine the strategies and tools needed to effectively manage large volumes of

observational data in production environments. Determine queries that deliver actionable insights for both the gateway operator and stakeholders that monitor specific hosted APIs.

- ✓ **Plan regular testing of production workloads** by using load testing.
- ✓ **Identify operational tasks beyond CI/CD or IaC processes** that require automation. Plan automation in areas such as API Management policy source management, Azure policies, and specific developer portal configurations.

Recommendations

These operational excellence recommendations can apply to either the service itself or to the traffic that flows through APIs and their policies. The *(Service)* or *(API)* designators indicate whether a recommendation targets the service or the APIs.

Expand table

Recommendation	Benefit
(Service) Configure Azure diagnostics resource logs for API Management.	Platform-generated logs are available to query for routine operations, on-demand needs, or urgent scenarios, such as security audits.
(Service) Use Azure Event Grid for automation based on meaningful events that your API Management instance raises, such as `APICreated`.	Automation or notification tasks can be built around key lifecycle events that occur in the API Management instance.
(Service) Avoid using a self-hosted gateway if a Microsoft-hosted gateway unit works in your scenario.	Automation or notification tasks can be built around key lifecycle events that occur in the API Management instance.
(Service) Apply all of the built-in Azure Policy policies that help you govern your instance and constrain it to align with workload requirements, including security requirements. For example, use *deny* policies to prevent APIs from being exposed over HTTP or to prevent the enabling of the direct management REST API. Use *audit* policies if deny policies aren't available, or create custom deny policies if it's important to your workload.	The API Management instance aligns with the design and remains aligned as the workload evolves.
(Service) Familiarize yourself with the Diagnose and solve problems capability in the Azure portal. Use the Network status blade in the portal to troubleshoot network connectivity.	Site reliability engineering individuals can identify and troubleshoot gateway performance, service availability, and network connectivity problems across all environments.

Recommendation	Benefit
(API) Use Event Hubs to handle log or event streams from API invocations that require near real-time reactions or short-timeframe windowed operations.	Near real-time availability of log entries is provided. The log data ingestion delay that occurs when logging to Azure Monitor within APIs is avoided.
(API) Support the usage of API tracing in development to help developers understand their policy implementation.	Developer productivity is optimized by providing insight into the implementation of policies within an API. Without this capability, developers need to introduce hacks in policy implementation to gain insight.
(API) Always define back ends by using the backends feature of API Management. Reference these back ends by ID in policy by using set-backend-service. Load balanced back ends should be defined as a backend pool.	Back-end connection changes apply to all APIs that use the back end without requiring updates to individual policy code. The risk of misconfigurations or overlooked API policy changes is reduced, and scenarios where multiple APIs share the same back end are suited through this layer of configuration abstraction.
(API) Design the versioning approach of your APIs to align with API Management's versioning and revision capabilities. Factor this approach into your API deployment operations.	An API versioning strategy supported out of the box by API Management is used to take advantage of built-in capabilities instead of building custom solutions.
(Service & API) Define a consistent and sustainable operational process for adding, modifying, and deleting APIs. Determine whether to manage this experience manually through the portal or implement it through an APIOps process. Prefer using an IaC-based approach instead of a portal based approach. API Management's representation in the Resource Manager API consists of many child resources, so it's important to adopt a layered approach for IaC-based management of this resource collection.	API specifications and their gateway implementations are integrated into the workload's change control processes. This approach prevents handling changes to back-end APIs differently from how they're exposed to API clients through the gateway.

Performance Efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance.

The Performance Efficiency design principles provide a high-level design strategy for achieving those capacity goals against the expected usage.

Start your design strategy based on the design review checklist for Performance Efficiency for defining a baseline based on key performance indicators for API Management.

Design checklist

- ✓ **Define performance targets:** Key metrics to evaluate the performance of the API Management gateway include capacity metrics, such as CPU and memory usage percentages for gateway infrastructure, request duration, and request throughput. In multiregion deployments, define performance targets for each region. The customer needs to define appropriate scaling thresholds based on traffic patterns, API workloads, and scaling times.
- ✓ **Collect performance data:** Review capabilities of built-in analytics, Azure Monitor metrics, Azure Monitor logs, Application Insights, and Event Hubs to collect and analyze performance at different levels of granularity.
- ✓ **Review how to identify live performance problems:** Indicators for live performance problems include Azure service availability, HTTP response errors, and errors raised in the **Diagnose and solve problems** section in the portal. Troubleshoot performance and availability problems by using Application Insights, Kusto queries, and recommendations from API Management Diagnostics in the Azure portal.
- ✓ **Test performance:** Test performance under production conditions by using load testing.
- ✓ **Evaluate adjacent services that might improve performance:** Caching policies or an external cache might improve the performance of specific API operations. Azure Front Door and Application Gateway are common choices for TLS offloading.
- ✓ **Review the documented limits and constraints:** API Management has limits and constraints. Review the documented constraints and compare them against your workload's requirements to see if you need to design a solution that avoids these constraints.

Recommendations

These performance efficiency recommendations can apply either to the service itself or to the traffic that flows through APIs and their policies. The *(Service)* or *(API)* designators indicate whether a recommendation targets the service or the APIs.

Expand table

Recommendation	Benefit
(Service) Dynamically scale to match demand. Configure autoscale rules or other automated processes to adjust gateway units to match a target usage capacity.	Elasticity is achieved based on concurrent usage, which prevents resource depletion of currently deployed units or over-allocation of capacity.
(API) Minimize expensive processing tasks, such as generating large buffered payload sizes, by using the validate-content policy on large request bodies or by maintaining a high number of active WebSockets.	The load on scale units is reduced, which enables them to process more requests concurrently before needing to scale out.
(Service & API) Collect performance metrics. Common API performance metrics include: - Request processing time for the gateway itself and for the full operation. - Gateway unit resource usage metrics. - Throughput measurements like requests per second or megabits per second. - Cache hit ratio.	Actual performance is measured against the targets of the workload.
(Service & API) Define a sampling percentage for Application Insights that provides sufficient visibility without affecting performance.	Observability data needs are satisfied without affecting overall performance.
(Service & API) Evaluate the performance impact of logic placement between your gateway, your back end, or your public entry point, such as Azure Front Door. The same processing tasks can often occur at any of these layers, with performance tradeoffs and limitations in optimization opportunities for each. If a task, such as an API management API policy, introduces too much latency, consider whether that task can run somewhere else in a more optimized way.	Tasks that add latency to APIs are run on compute optimized to meet workload requirements.

Azure policies

Azure provides an extensive set of built-in policies related to API Management and its dependencies. Some of the preceding recommendations can be audited through Azure Policy. For example, you can check whether:

- The gateway is configured for zone redundancy.
- Proper network controls are in place for the API Management gateway, such as deployment in a virtual network.

- The service configuration endpoints aren't publicly accessible.
- The direct Management REST API is disabled.

For comprehensive governance, review the Azure Policy built-in definitions and other policies that might affect the security of the API Management gateway.

Azure Advisor recommendations

Azure Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments.

For more information, see Azure Advisor.

Advisor might surface other recommendations in your production system as well, such as:

- Failure to require long JWT key sizes in the validate-jwt policy.
- You used a legacy Resource Manager API version to deploy the resource.
- API key tokens are close to their expiration date.
- Failure in a certificate rotation operation.

Tradeoffs

You might have to make design tradeoffs if you use the approaches in the pillar checklists. Here are some examples of advantages and drawbacks.



High availability through redundancy and isolation

- **High availability.** Adding redundancy to an architecture affects costs. For example, provisioning at least three units to avoid zonal outages might not be financially feasible for your workload. Costs increase further with a multiregion architecture, which requires at least six units, or three units per region. A multiregion setup also adds operational costs for coordinating safe deployments, reliable scaling, and failover coordination with back ends.

- **Isolation.** Isolating workloads across workspaces or API Management instances adds operational complexity because it includes managing a multitenant system that has compute isolation.



Scale to match demand

- When you automatically scale out to meet the demand from well-behaved clients, those costs are already factored into your cost models. However, this scaling capability also

allows the service to scale to handle traffic from nuisance and malicious traffic.

- Mitigating undesired traffic incurs costs. Adding services like a web application firewall and DDoS protection increases expenses. Scaling your service to handle traffic increases costs because of added units. Setting upper scaling limits can cap spending but might result in reliability problems for legitimate users if malicious or harmful traffic overwhelms your API.



Federated versus distributed approach

A fundamental decision in API Management is whether to colocate disparate workloads within a single API Management instance or to isolate workloads across multiple instances in a fully autonomous topology.

API Management workspaces enable efficient operation as a multitenant colocation platform for multiple API teams. The tiered pricing models are designed to allow service costs to be shared across all tenants that use the gateways. However, like any colocation platform, outages or misconfigurations can result in widespread effects on unrelated workloads that share the same infrastructure.

A fully distributed model, where each workload team manages its own instances, introduces duplicative costs and redundancy in routine operations. However, it provides inherent blast radius mitigation for reliability, security, and performance-related incidents.

Next steps

API Management is often combined with the following services. Be sure to review their service guides or product documentation if your workload includes the following services.

- Application Gateway
- Azure Cache for Redis
- Azure Front Door
- Key Vault
- Azure OpenAI Service
- Azure Virtual Network
- Web Application Firewall

The following articles demonstrate some of the recommendations discussed in this article.

- Access Azure OpenAI and other language models through a gateway
- Automated API deployments using APIOps
- API Management architecture in an application landing zone and Azure landing zone platform considerations for API Management deployments

Architecture best practices for Azure Application Gateway v2

Article • 03/28/2025

Azure Application Gateway v2 is a web traffic load balancer that operates at the application layer. Application Gateway manages traffic to your web applications based on the attributes of an HTTP request. Use Application Gateway for scenarios that have advanced routing capabilities and require enhanced security and scalability.

This article assumes that as an architect, you've reviewed the networking options and chose Application Gateway as the web traffic load balancer for your workload. The guidance in this article provides architectural recommendations that are mapped to the principles of the Well-Architected Framework pillars.

> **ⓘ Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern along with design strategies localized to the technology scope.
>
> Also included are recommendations for the technology capabilities that can help materialize those strategies. The recommendations don't represent an exhaustive list of all configurations that are available for Application Gateway and its dependencies. Instead, they list the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept or to optimize your existing environments.
>
> Foundational architecture that demonstrates the key recommendations: **Baseline highly available, zone-redundant web application architecture**.

Technology scope

This review focuses on the interrelated decisions for the following Azure resources:

- Application Gateway v2
- Web Application Firewall (WAF) on Application Gateway

Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover fast from failures.**

Reliability design principles provide a high-level design strategy applied for individual components, system flows, and the system as a whole.

Design checklist

Start your design strategy based on the design review checklist for Reliability. Determine its relevance to your business requirements while keeping in mind the features of Application Gateway and its dependencies. Extend the strategy to include more approaches as needed.

- ✓ **Use Application Gateway v2** in new deployments unless your workload specifically requires Application Gateway v1.
- ✓ **Build redundancy in your design.** Spread Application Gateway instances across availability zones to improve fault tolerance and build redundancy. Traffic goes to other zones if one zone fails. For more information, see Recommendations for using availability zones and regions.
- ✓ **Plan extra time for rule updates** and other configuration changes before you access Application Gateway or make further changes. For example, you might need extra time to remove servers from a back-end pool because they have to drain existing connections.
- ✓ **Implement the Health Endpoint Monitoring pattern.** Your application should expose health endpoints, which aggregate the state of the critical services and dependencies that your application needs to serve requests. Application Gateway health probes use the endpoint to detect the health of servers in the back-end pool. For more information, see Health Endpoint Monitoring pattern.
- ✓ **Evaluate the impact of interval and threshold settings on a health probe.** The health probe sends requests to the configured endpoint at a set interval. And the back end tolerates a limited number of failed requests before it's marked as unhealthy. These settings can conflict, which presents a tradeoff.
    - A higher interval puts a higher load on your service. Each Application Gateway instance sends its own health probe, so 100 instances every 30 seconds equals 100 requests every 30 seconds.
    - A lower interval increases the amount of time before the health probe detects an outage.

- A low, unhealthy threshold increases the chance of short, transient failures shutting down a back end.
- A high threshold increases the amount of time that it takes a back end to go out of rotation.

✓ **Verify downstream dependencies through health endpoints.** To isolate failures, each of your back ends might have its own dependencies. For example, an application that you host behind Application Gateway might have multiple back ends, and each back end connects to a different database, or replica. When such a dependency fails, the application might work but doesn't return valid results. For that reason, the health endpoint should ideally validate all dependencies.

If each call to the health endpoint has a direct dependency call, that database receives 100 queries every 30 seconds instead of one query. To avoid excessive queries, the health endpoint should cache the state of the dependencies for a short period of time.

✓ **Consider Application Gateway limitations and known issues that might affect reliability.** Review the Application Gateway FAQ for important information about by-design behavior, fixes under construction, platform limitations, and possible workarounds or mitigation strategies. Don't use UDRs in the Application Gateway dedicated subnet.

✓ **Consider Source Network Address Translation (SNAT) port limitations in your design** that can affect back-end connections on Application Gateway. Some factors affect how Application Gateway reaches the SNAT port limit. For example, if the back end is a public IP address, it requires its own SNAT port. To avoid SNAT port limitations, you can do one of the following options:

- Increase the number of instances for each Application Gateway.
- Scale out the back ends to have more IP addresses.
- Move your back ends into the same virtual network, and use private IP addresses for the back ends.

  If Application Gateway reaches the SNAT port limit, it affects the requests per second (RPS). For example, Application Gateway can't open a new connection to the back end, and the request fails.

Recommendations

Expand table

Recommendation	Benefit
Deploy Application Gateway instances in a zone-aware configuration. Check regional support for zone redundancy because not all regions offer this feature.	When you spread multiple instances across zones, your workload can withstand failures in a single zone. If you have an unavailable zone, traffic automatically shifts to healthy instances in other zones, which maintains application reliability.
Use Application Gateway health probes to detect back-end unavailability.	Health probes ensure that traffic only routes to back ends that can handle the traffic. Application Gateway monitors the health of all the servers in its back-end pool and automatically stops sending traffic to any server that it considers unhealthy.
Configure rate-limiting rules for Azure WAF so that clients can't send too much traffic to your application.	Use rate limiting to avoid problems like retry storms.
Don't use UDRs on Application Gateway so that the back-end health report functions properly and generates the correct logs and metrics. If you must use a UDR in the Application Gateway subnet, see Supported UDRs.	UDRs on the Application Gateway subnet can cause some problems. Don't use UDRs on the Application Gateway subnet so that you can view the back-end health, logs, and metrics.
Configure the *IdleTimeout* settings to match the listener and traffic characteristics of the back-end application. The default value is four minutes. You can configure it to a maximum of 30 minutes. For more information, see Load balancer Transmission Control Protocol (TCP) reset and idle timeout.	Set the *IdleTimeout* to match the back end. This setting ensures that the connection between Application Gateway and the client stays open if the back end takes more than four minutes to respond to the request. If you don't configure this setting, the connection closes, and the client doesn't see the back-end response.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload.

The Security design principles provide a high-level design strategy for achieving those goals by applying approaches to the technical design of Application Gateway.

Design checklist

Start your design strategy based on the design review checklist for Security and identify vulnerabilities and controls to improve the security posture. Extend the strategy to include more approaches as needed.

- ✓ **Review the security baseline for Application Gateway.**
- ✓ **Block common threats at the edge.** WAF integrates with Application Gateway. Enable WAF rules on the front ends to protect applications from common exploits and vulnerabilities at the network edge, which is close to the attack source. For more information, see WAF on Application Gateway.

  Understand how WAF affects Application Gateway capacity changes. When you enable WAF, Application Gateway:

  - Buffers every request until it fully arrives.
  - Checks if the request matches with any rule violation in its core rule set.
  - Forwards the packet to the back-end instances.

  Large file uploads that are 30 MB or more can introduce significant latency. Application Gateway capacity requirements change when you enable WAF, so we recommend that you properly test and validate this method first.

  When you use Azure Front Door and Application Gateway to protect HTTP or HTTPS applications, use WAF policies in Azure Front Door and lock down Application Gateway to receive traffic only from Azure Front Door. Certain scenarios can force you to implement rules specifically on Application Gateway. For example, if you require ModSec CRS 2.2.9, CRS 3.0, or CRS 3.1 rules, you can only implement these rules on Application Gateway. Conversely, Azure Front Door supports rate limiting and geo filtering, and Application Gateway doesn't support these features.
- ✓ **Allow only authorized access to the control plane.** Use Application Gateway role-based access control (RBAC) to restrict access to only the identities that need it.
- ✓ **Protect data in transit.** Enable end-to-end Transport Layer Security (TLS), TLS termination, and end-to-end TLS encryption. When you re-encrypt back-end traffic, ensure that the back-end server certificate contains both the root and intermediate certificate authorities (CAs).

  Use a well-known CA to issue a TLS certificate of the back-end server. If you don't use a trusted CA to issue the certificate, Application Gateway checks until it finds a

trusted CA certificate. It establishes a secure connection only when it finds a trusted CA. Otherwise, Application Gateway marks the back end as unhealthy.

- ✓ **Protect application secrets.** Use Azure Key Vault to store TLS certificates for increased security and an easier certificate renewal and rotation process.
- ✓ **Reduce the attack surface and harden the configuration.** Remove default configurations that you don't need, and harden your Application Gateway configuration to tighten security controls. Comply with all network security group (NSG) restrictions for Application Gateway.

  Use an appropriate Domain Name System (DNS) server for back-end pool resources. When the back-end pool contains a resolvable fully qualified domain name (FQDN), the DNS resolution is based on a private DNS zone or custom DNS server (if configured on the virtual network), or it uses the default Azure-provided DNS.
- ✓ **Monitor anomalous activity.** Regularly review logs to check for attacks and false positives. Send WAF logs from Application Gateway to your organization's centralized security information and event management (SIEM), such as Microsoft Sentinel, to detect threat patterns and incorporate preventative measures in the workload design.

Recommendations

Expand table

Recommendation	Benefit
Set up a TLS policy for enhanced security. Ensure that you use the latest TLS policy version.	Use the latest TLS policy to enforce the use of TLS 1.2 and stronger ciphers. The TLS policy includes control of the TLS protocol version and the cipher suites and also the order in which a TLS handshake uses ciphers.
Use Application Gateway for TLS termination.	Performance improves because requests that go to different back ends don't have to reauthenticate to each back end. The gateway can access the request content and make intelligent routing decisions. You only need to install the certificate on Application Gateway, which simplifies certificate management.

Recommendation	Benefit
Integrate Application Gateway with Key Vault to store TLS certificates.	This approach provides stronger security, easier separation of roles and responsibilities, support for managed certificates, and an easier certificate renewal and rotation process.
Comply with all NSG restrictions for Application Gateway.	The Application Gateway subnet supports NSGs, but there are some restrictions. For instance, some communication with certain port ranges is prohibited. Make sure you understand the implications of those restrictions.

Cost Optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget while meeting business requirements.

The Cost Optimization design principles provide a high-level design strategy for achieving those goals and making tradeoffs as necessary in the technical design related to Application Gateway and its environment.

Design checklist

Start your design strategy based on the design review checklist for Cost Optimization for investments. Fine-tune the design so that the workload is aligned with the budget that's allocated for the workload. Your design should use the right Azure capabilities, monitor investments, and find opportunities to optimize over time.

- ✓ **Familiarize yourself with Application Gateway and WAF pricing**. Choose appropriately sized options to meet your workload capacity demand and deliver expected performance without wasting resources. To estimate costs, use the pricing calculator.

- ✓ **Remove unused Application Gateway instances, and optimize underused instances.** To avoid unnecessary costs, identify and delete Application Gateway instances that have empty back-end pools. Stop Application Gateway instances when they're not in use.

- ✓ **Optimize the scaling cost of your Application Gateway instance.** To optimize your scaling strategy and reduce your wokload's demands, see Recommendations for optimizing scaling cost.

To scale the service in or out based on application traffic requirements, use autoscaling in Application Gateway v2.

- ✓ **Monitor Application Gateway consumption metrics**, and understand their cost impact. Azure charges for metered instances of Application Gateway based on tracked metrics. Evaluate the various metrics and capacity units, and determine the cost drivers. For more information, see Microsoft Cost Management.

Recommendations

Expand table

Recommendation	Benefit
Stop Application Gateway instances when they're not in use. For more information, see Stop-AzApplicationGateway and Start-AzApplicationGateway.	A stopped Application Gateway instance doesn't incur costs. Application Gateway instances that continuously run can incur unnecessary costs. Evaluate usage patterns, and stop instances when you don't need them. For example, expect low usage after business hours in dev/test environments.
Monitor key cost driver Application Gateway metrics, like: - Estimated billed capacity units. - Fixed billable capacity units. - Current capacity units. Make sure you account for bandwidth costs.	Use these metrics to validate whether the provisioned instance count matches the amount of incoming traffic, and ensure that you fully utilize the allocated resources.

Operational Excellence

Operational Excellence primarily focuses on procedures for **development practices, observability, and release management.**

The Operational Excellence design principles provide a high-level design strategy for achieving those goals for the operational requirements of the workload.

Design checklist

Start your design strategy based on the design review checklist for Operational Excellence for defining processes for observability, testing, and deployment related to Application Gateway.

- ✓ **Enable diagnostics on Application Gateway and WAF.** Collect logs and metrics so you can monitor the health of the workload, identify trends in the workload performance and reliability, and troubleshoot problems. To design your overall monitoring approach, see Recommendations for designing and creating a monitoring system.

  Use capacity metrics to monitor the use of the provisioned Application Gateway capacity. Set alerts on metrics to notify you of capacity problems or other problems either at Application Gateway or the back end. Use diagnostic logs to manage and troubleshoot problems with Application Gateway instances.

- ✓ **Use Azure Monitor Network Insights** to get a comprehensive view of health and metrics for network resources, including Application Gateway. Use centralized monitoring to quickly identify and resolve problems, optimize performance, and ensure the reliability of your applications.

- ✓ **Monitor Application Gateway recommendations in Azure Advisor.** Configure alerts to notify your team when you have new, critical recommendations for your Application Gateway instance. Advisor generates recommendations based on properties, such as the category, impact level, and recommendation type.

Recommendations

Expand table

Recommendation	Benefit
Configure alerts to notify your team when capacity metrics, like CPU usage and compute unit usage, cross recommended thresholds. To configure a comprehensive set of alerts based on capacity metrics, see Application Gateway high-traffic support.	Set alerts when metrics cross thresholds so that you know when your usage increases. This approach ensures that you have enough time to implement necessary changes to your workload and prevents degradation or outages.
Configure alerts to notify your team about metrics that indicate problems either at Application Gateway or the back end. We recommend that you evaluate the following alerts: - Unhealthy host count - Response status, such as 4xx and 5xx errors - Back-end response status, such as 4xx	Use alerts to help ensure that your team can respond to problems in a timely manner and facilitate troubleshooting.

Recommendation	Benefit
and 5xx errors - Back-end last byte response time - Application Gateway total time For more information, see Metrics for Application Gateway.	
Enable diagnostic logs on Application Gateway and WAF to collect firewall logs, performance logs, and access logs.	Use logs to help detect, investigate, and troubleshoot problems with Application Gateway instances and your workload.
Use Advisor to monitor Key Vault configuration problems. Set an alert to notify your team when you get the recommendation that states **Resolve Azure Key Vault issue for your Application Gateway**.	Use Advisor alerts to stay up to date and fix problems immediately. Prevent any control plane or data plane-related problems. Application Gateway checks for the renewed certificate version in the linked Key Vault instance every 4 hours. If the certificate version is inaccessible because of an incorrect Key Vault configuration, it logs that error and pushes a corresponding Advisor recommendation.

Performance Efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance.

The Performance Efficiency design principles provide a high-level design strategy for achieving those capacity goals against the expected usage.

Design checklist

- ✓ **Estimate capacity requirements for Application Gateway to support your workload requirements.** Take advantage of the autoscaling functionality in Application Gateway v2. Set appropriate values for the minimum and maximum number of instances. Appropriately size the dedicated subnet that Application Gateway requires. For more information, see Recommendations for capacity planning.

  Application Gateway v2 scales out based on many aspects, such as CPU, network throughput, and current connections. To determine the approximate instance count, factor in these metrics:

- *Current compute units:* This metric indicates the CPU usage. One Application Gateway instance equals approximately 10 compute units.
- *Throughput:* An Application Gateway instance can serve approximately 500 Mbps of throughput. This data depends on the type of payload.

Consider this equation when you calculate instance counts.

$$\text{Approx. instance count} = \boldsymbol{max}\left(\frac{\text{Current compute units}}{10}, \frac{\text{Throughput in Mbps}}{500}, \frac{\text{Current connections}}{25000}\right)$$

After you estimate the instance count, compare that value to the maximum instance count. Use this comparison to determine how close you are to the maximum available capacity.

✓ **Take advantage of features for autoscaling and performance benefits.** The v2 SKU offers autoscaling, which scales up Application Gateway as traffic increases. Compared to the v1 SKU, the v2 SKU has capabilities that enhance the performance of the workload. For example, the v2 SKU has better TLS offload performance, quicker deployment and update times, and zone-redundancy support. For more information, see Scaling Application Gateway v2 and WAF v2.

If you use Application Gateway v1, consider migrating to Application Gateway v2. For more information, see Migrate Application Gateway and WAF from v1 to v2.

Recommendations

Expand table

Recommendation	Benefit
Set the minimum instance count to an optimal level based on you estimated instance count, actual Application Gateway autoscaling trends, and your application patterns. Check the current compute units for the past month. This metric represents the gateway's CPU usage. To define the minimum instance count, divide the peak usage by 10. For example, if your average current compute units in the past month is 50, set the minimum instance count to five.	For Application Gateway v2, autoscaling takes approximately three to five minutes before the extra set of instances are ready to serve traffic. During that time, if Application Gateway has short spikes in traffic, expect transient latency or loss of traffic.
Set the maximum autoscale instance count to the maximum possible, which is 125 instances. Make	Application Gateway can scale out as needed to handle increased traffic to your

Recommendation	Benefit
sure that the Application Gateway dedicated subnet has sufficient available IP addresses to support the increased set of instances.	applications. This setting doesn't increase cost because you only pay for the consumed capacity.
Appropriately size the Application Gateway dedicated subnet. We highly recommend a /24 subnet for an Application Gateway v2 deployment. If you want to deploy other Application Gateway resources in the same subnet, consider the extra IP addresses that you require for the maximum instance count. For more considerations about sizing the subnet, see Application Gateway infrastructure configuration.	Use a /24 subnet to provide support for all IP addresses that your Application Gateway v2 deployment needs. Application Gateway uses one private IP address for each instance and another private IP address if you configure a private front-end IP. The Standard_v2 or WAF_v2 SKU can support up to 125 instances. Azure reserves five IP addresses in each subnet for internal use.

Azure policies

Azure provides an extensive set of built-in policies related to App Service and its dependencies. A set of Azure policies can audit some of the preceding recommendations. For example, you can check whether:

- You should enable WAF for Application Gateway. Deploy WAF in front of public-facing web applications to add another inspection layer for incoming traffic. WAF provides centralized protection for your web applications. It helps prevent common exploits and vulnerabilities, such as SQL injections, cross-site scripting, and local and remote file executions. You can also use custom rules to restrict access to your web applications based on countries or regions, IP address ranges, and other HTTP or HTTPS parameters.
- WAF should use the specified mode for Application Gateway. Ensure that all WAF policies for Application Gateway use *Detection* or *Prevention* mode.
- You should enable Azure DDoS Protection. Enable DDoS Protection for all virtual networks that have a subnet that contains Application Gateway with a public IP.

For comprehensive governance, review the Azure Policy built-in definitions and other policies that might affect networking.

Azure Advisor recommendations

Azure Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments. Here are some recommendations that can help you improve the reliability, security, cost effectiveness, performance, and operational excellence of Application Gateway.

- Reliability
- Security
- Cost Optimization
- Performance
- Operational Excellence

Next steps

- Use API gateways in microservices
- Azure Firewall and Application Gateway for virtual networks
- Protect APIs with Application Gateway and Azure API Management
- Securely managed web applications
- Zero Trust network for web applications with Azure Firewall and Application Gateway
- Quickstart: Direct web traffic with Application Gateway via the Azure portal

---

Feedback

Was this page helpful? Yes No

Architecture best practices for Application Insights

Article • 11/27/2024

Application Insights is an extensible application performance management service that you can use to monitor live applications and automatically detect performance anomalies. It includes powerful analytics tools to help you diagnose problems and understand how users interact with your application. This architecture guide provides best practices for Application Insights based on the five pillars of the Azure Well-Architected Framework pillars.

> **Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern and design strategies that are localized to the technology scope.
>
> Also included are recommendations on the technology capabilities that can help materialize those strategies. The recommendations aren't an exhaustive list of all configurations that are available for Application Insights and its dependencies. Instead, they provide the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept or to optimize your existing environments.

Technology scope

This guide focuses on the interrelated decisions for the following Azure resources:

- Application Insights
- Log Analytics workspaces

> **Note**
>
> Application Insights stores and exposes data in a Log Analytics workspace. When you evaluate your metrics, keep Log Analytics as a critical dependency on Application Insights. For more information, see Well-Architected Framework perspective on Log Analytics.

Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover from failures quickly**.

The **Reliability design principles** provide a high-level design strategy applied for individual components, system flows, and the system as a whole.

Design checklist for Reliability

Start your design strategy based on the design review checklist for Reliability. Determine its relevance to your business requirements and keep in mind the dependency on Log Analytics. Extend the strategy to include more approaches as needed.

- ✓ **Align your application monitoring design with business objectives and avoid unnecessary complexity.** Determine the number of Application Insights resources that you need and where to deploy them.
- ✓ **Configure sampling based on the criticality of the data.** Identify the user and system flows of your workload, determine the events that you want to gather, and configure sampling accordingly.
  - For manually instrumented applications, see Azure Monitor OpenTelemetry Distro or Application Insights SDKs (Classic API).
  - For autoinstrumentation, sampling is described in environment-specific articles. For more information, see Configure sampling in Azure Functions.
- ✓ **Create a comprehensive resiliency plan.**
  1. Start by conducting a failure mode analysis to identify potential scenarios when Application Insights might fail or become unreachable. These reasons can include network problems, authentication problems, or service disruptions.
  2. Determine how your workload should behave if Application Insights is unreachable at boot or during runtime based on the criticality of application monitoring to your business goals. Define and document the expected behavior of your workload.
  3. Test your resiliency plan. For example, you can validate network failures by using a network security group rule, and authenticate failures by changing the connection string.
- ✓ **Plan for workspace resiliency and recovery by defining targets for your data collection components.**

1. Assess the criticality of the data that you collect and determine if it needs to be recoverable.
2. Review service limits for Application Insights and Log Analytics workspace to understand restrictions on data collection and retention, and other aspects of the service.
3. Consider moving to a dedicated cluster to take advantage of workspace resilience where the availability of Application Insights telemetry within the retention period defined is business critical.
4. Use diagnostic settings to export platform logs and metrics to the destination of your choice (for example, a storage account) for backup and recovery purposes.

- ✓ **Implement a timely and reliable scaling strategy to plan for data ingestion growth.** Monitor and adjust limits on sampling and data ingestion as traffic grows to prevent data loss because of sampling or exceeding the daily cap. This helps ensure that your data ingestion process scales effectively with traffic increases.

- ✓ **Ensure quick recovery of your application monitoring solution in case of service failure.** Adopt infrastructure as code and use Bicep templates to create or re-create your user experience in Application Insights, including alerts, dashboards, and queries. This approach helps ensure that all critical components are quickly restored, which minimizes downtime and maintains service reliability.

Recommendations

Expand table

Recommendation	Benefit
Use one Application Insights resource per workload per environment, such as one for development, one for staging, and one for production.	Using multiple Application Insights resources prevents mixing telemetry from different application versions and helps ensure that a misconfiguration of one resource doesn't affect the logging for another environment.
Deploy your Application Insights resource in the same region as the underlying Log Analytics workspace.	Because Application Insights requires Log Analytics to function properly, having both resources in different regions doubles the chance for a regional failure to cause problems.
Implement a resilient workspace design by using best practices for Azure Monitor Logs.	Best practices help ensure continuous and robust monitoring by minimizing disruptions.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload. Security is one of the most important aspects of any architecture. Azure Monitor provides features to employ both the principle of least privilege and defense in depth.

The **Security design principles** provide a high-level design strategy for achieving those goals by applying approaches to the technical design of Application Insights. Use the following information to maximize the security of Application Insights and help ensure that only authorized users access collected data.

Design checklist for Security

Start your design strategy based on the design review checklist for Security. Identify vulnerabilities and controls to improve the security posture. Extend the strategy to include more approaches as needed.

- ✓ **Review the Azure Monitor security baseline.** It provides guidance on security best practices to help secure your cloud solutions on Azure, including Application Insights.
- ✓ **Keep Application Insights instrumentation up to date to help ensure that your application monitoring solution is secure.** Follow our SDK update guidance and update Application Insights SDKs (Classic API) at least once a year. We recommend that you follow similar practices for the Azure Monitor OpenTelemetry Distro.
- ✓ **Define a strategy for handling personal data in Application Insights.** To ensure ongoing compliance, regularly verify that the collection and handling of data, including IP addresses and personal data, complies with relevant regulations, such as GDPR.
- ✓ **Create intentional segmentation in your application monitoring design.** Determine the number of Application Insights resources that you need.
- ✓ **Use Azure Monitor customer-managed key.** By default, data in Azure Monitor is encrypted with Microsoft-managed keys. You can use your own encryption key to protect the data and saved queries in your workspaces. Customer-managed keys in Azure Monitor give you greater flexibility to manage access controls to stored data.
- ✓ **Control network traffic.** Consider private connectivity for accessing Azure services. Private connectivity effectively isolates your traffic from the public internet. Data flows for private networking include both data ingestion and query operations,

each targeting distinct endpoints. These endpoints can be managed independently. This approach allows you to configure private ingestion while maintaining public query access, or vice versa. By doing so, you can apply defense-in-depth principles by creating localized network controls at all available network boundaries.

- ✓ **Enhance data protection by securing storage systems and limiting access.** Visit the Log Analytics service guide to learn about how to secure the data that you collect.

Recommendations

Expand table

Recommendation	Benefit
If your business needs and hosting environment don't require manual instrumentation, consider using autoinstrumentation.	This approach eliminates the need for manual SDK updates, requires no code changes, and eliminates the overhead of maintaining instrumentation code. It can also enhance security by helping ensure consistent application monitoring without manual intervention.
Use managed identities and Microsoft Entra ID for your authentication and authorization needs.	Eliminates the need for credentials management because Azure manages, rotates, and protects these credentials.
Stop collecting personal data or obfuscate, anonymize, or adjust collected data. By default, Application Insights doesn't store IP addresses. We recommend that you keep this default setting.	Excludes your data from being considered *personal* and prevents breaking any compliance requirements or local regulations.
Use one Application Insights resource per workload per environment. For example, use one for development, one for staging, and one for production.	Using multiple Application Insights resources helps ensure data isolation and security, and makes it easier to apply environment-specific configurations and access controls.
Use Azure Private Link to access Azure services over a private endpoint. Follow the configuration and limitations. Avoid connecting your AMPLS to multiple virtual networks that share the same DNS because this causes *wrong private endpoint* errors.	A properly configured AMPLS topology enhances the confidentiality of your workload's logs by keeping logging traffic on fully private networks.

Cost Optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget while meeting business requirements.

The Cost Optimization design principles provide a high-level design strategy for achieving those goals and making tradeoffs as necessary in the technical design related to Application Insights and its environment.

For more information on how data charges are calculated for the underlying Log Analytics workspaces of your Application Insights resources, see Azure Monitor Logs cost calculations and options.

Design checklist for Cost Optimization

Start your design strategy based on the design review checklist for Cost Optimization for investments. Fine tune the design so that the workload aligns with the budget that's allocated for the workload. Your design should use the right Azure capabilities, monitor investments, and find opportunities to optimize over time.

- ✓ **Review Azure Monitor pricing to create a cost model.** Estimate the initial cost, run rates, and ongoing costs by using the pricing calculator. Notice that Application Insights is billed through the Log Analytics workspace into which its log data ingested.

- ✓ **Tune the amount of data collected.**
  - Use sampling on an Application Insights-level to reduce data traffic and storage costs while preserving a statistically correct analysis of application data.
  - Application Insights has several possible log sources. Use log levels to tune and reduce trace log telemetry.

- ✓ **Limit unplanned charges for your workspace.** You can set a daily cap in both Application Insights and Log Analytics.

- ✓ **Review costs like regional pricing and available pricing tiers regularly.** The most significant charges for most Azure Monitor implementations are typically ingestion and retention of data in your Log Analytics workspaces. For more information, see Azure Monitor Logs cost calculations and options or the Cost Optimization section in the Log Analytics service guide.

- ✓ **Remove or optimize legacy, unneeded, and underutilized components of your application monitoring solution regularly.** Edit ApplicationInsights.config to turn off collection modules that you don't need. For example, you might decide that performance counters or dependency data isn't required. Use telemetry filters or

processors in code to help optimize component costs by not logging or sampling irrelevant calls.

- ✓ **Optimize environment costs.** Determine the number of Application Insights resources you need and where to deploy them. Optimize log levels and metrics collected per environment to manage costs effectively. For example, production environments might require different logging levels compared to development environments.
- ✓ **Optimize flow costs.** Use telemetry filters or processors in code to help optimize component costs by not logging or sampling irrelevant calls.
- ✓ **Optimize code costs.** Make sure to use updated Application Insights SDKs. Earlier versions of the ASP.NET Core SDK and Worker Service SDK collect many counters by default, which were collected as custom metrics. Use later versions to specify only required counters.
- ✓ **Optimize personnel time.** Use Application Insights experiences such as the application map and failure and performance views, and customize saved queries, dashboards and workbooks for your specific workload needs. Keep track of your deployments and other events with Release Annotations.

Recommendations

Expand table

Recommendation	Benefit
Set a daily cap in both Application Insights and Log Analytics, but try to avoid reaching it.	Reaching the daily cap can cause data to not be written in your Log Analytics workspace. Setting a cap is a risky but effective solution for throttling spend.
For standard availability tests, the need for testing from different locations might vary between production and preproduction environments. Reduce the number of locations accordingly.	Tests incur costs, so only have enough tests to give you confidence in your workload's heath.
Deploy your Application Insights resource in the same region as the underlying Log Analytics workspace.	This approach minimizes latency and reduces costs associated with cross-region telemetry.
Decrease the sample rate for less critical flows and increase it for flows with high criticality. Use telemetry filters for nonessential telemetry.	Data volume is a cost driver when designing your Application Insights integration. Reducing data volume through sample rates and filters is a

Recommendation	Benefit
	potential solution to keeping your data collection under control.
Limit the number of Ajax calls that can be reported in every page view, or disable Ajax reporting. If you disable Ajax calls, you also disable JavaScript correlation.	Data volume is a cost driver when designing your Application Insights integration for your client application. Reducing data volume through reduced client page reporting is a potential solution to keeping your data collection costs under control.
Use preaggregated metrics to handle high-volume telemetry data more efficiently and limit the use of custom metrics. However, the Application Insights option to enable alerting on custom metric dimensions can also increase costs. Use metrics to ensure that cost savings are being obtained.	All custom metrics are stored in both logs and metric stores. Preaggregated metrics reduce storage costs related to custom metrics.
If your business needs and hosting environment don't require manual instrumentation, consider using autoinstrumentation.	This approach optimizes software engineering time by eliminating the need for manual SDK updates, code changes related to new versions, and the overhead of maintaining instrumentation code.
Limit unwanted trace logging: • Remove health checks. See *Filter out noisy traces* in Optimizing Cost using the Azure Monitor OpenTelemetry Distro. • For Azure Kubernetes Service (AKS), adjust control plane and data plane logs. • For Azure Functions, adapt log levels and scope to optimize log volume.	Limiting unwanted trace logging reduces the amount of stored data, which can lower storage costs.

Operational Excellence

Operational Excellence primarily focuses on procedures for **development practices, observability, and release management**.

The Operational Excellence design principles provide a high-level design strategy for achieving those goals for the operational requirements of the workload.

Design checklist for Operational Excellence

Start your design strategy based on the design review checklist for Operational Excellence for defining processes for observability, testing, and deployment related to Application Insights.

- ✓ **Integrate your application monitoring team members' specializations into a robust set of practices to instrument and monitor your workload.** Choose an instrumentation method (for example, autoinstrumentation or manual instrumentation) that's best for your situation based on your business needs and supported environments, languages, and resource providers.
- ✓ **Ensure optimal performance of your application monitoring solution by keeping Application Insights instrumentation up to date.** Follow our SDK update guidance and update Application Insights SDKs (Classic API) at least once a year. It's recommended to follow similar practices for the Azure Monitor OpenTelemetry Distro. Using the latest SDK or Distro versions ensures access to support services and provides the latest functionality and bug fixes.
- ✓ **Formalize ideation and planning processes.** Use work item ingetration to easily create work items in GitHub or Azure DevOps that have relevant Application Insights data embedded in them.
- ✓ **Configure Application Insights to monitor the availability and responsiveness of your web application.** Use built-in features like queries and dashboards based on your specific business needs. After you deploy your application, set up recurring tests to monitor availability and responsiveness.
- ✓ **Develop an effective emergency operations practice.** Use alerts and workbooks to identify and respond to incidents. Clearly define human responsiblities. For example, determine who reboots the application if your workload fails.
- ✓ **Clearly define your workload's safe deployment practices.** Use Release Annotations as part of your failure mitigation strategies to keep track of your deployments and other events.

Recommendations

Expand table

Recommendation	Benefit
If your business needs and hosting environment don't require manual	This approach optimizes software engineering time by eliminating the need for manual SDK updates,

Recommendation	Benefit
instrumentation, consider using autoinstrumentation.	code changes related to new versions, and the overhead of maintaining instrumentation code.
If your business needs require manual instrumentation, adopt the Azure Monitor OpenTelemetry Distro.	Avoid a future forced migration from Application Insights SDKs (Classic API) by adopting an OpenTelemetry-based instrumentation method.
Use connection strings.	Make telemetry ingestion more reliable and remove dependencies on global ingestion endpoints.

Performance Efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance.

The Performance Efficiency design principles provide a high-level design strategy for achieving those capacity goals against the expected usage.

Design checklist for Performance Efficiency

Start your design strategy based on the design review checklist for Performance Efficiency. Use key performance indicators for Application Insights to define a baseline.

- ✓ **Define performance targets related to your application monitoring requirements.** Determine the number of Application Insights resources you need.
- ✓ **Conduct capacity planning.** Understand usage patterns and how much data is coming in, and review ingestion and sample rates.
- ✓ **Select the right region for your application monitoring solution.** Deploy your Application Insights resource in the same region as the underlying Log Analytics workspace to prevent latency and reliability problems. See Create a resource.
- ✓ **Evaluate how many Application Insights resources you need.** Monitoring mulitple applications or application components with a single Application Insights resource provides a holistic view, but can also affect the performance of experiences like Application Map and Usage.
- ✓ **Optimize code and infrastructure.** Regularly evaluate custom Application Insights code to reduce complexity, improve performance, and ensure that the code is up to date.

- ✓ **Understand usage patterns and how much data is coming in by reviewing ingestion and sample rates.** To optimize data usage, adjust them accordingly and reduce the amount of custom metrics, for example ITelemetryProcessor.
- ✓ **Continuously optimize performance.** Use built-in features like Smart Detection, queries, and dashboards to find components that show deteriorating performence.

Recommendations

Expand table

Recommendation	Benefit
Use one Application Insights resource per workload per environment (such as one for development, one for staging, and one for production).	This approach prevents mixing telemetry from different application versions.
When applicable, ensure that profiling frequency and duration is set appropriately.	Avoid adding excessive overhead to the running process.

Azure policies

Azure provides built-in policies related to Application Insights and its dependencies. Some of the preceding recommendations can be audited through Azure Policy. For example, you can check whether:

- Application Insights components should prevent log ingestion public networks, or sources that Microsoft Entra ID doesn't authenticate.
- You should enforce linking of the Application Insights component to a Log Analytics workspace to encrypt logs.

For comprehensive governance, review the Azure Policy built-in definitions for Application Insights and other policies that might affect the security of the application performance monitoring solution.

Azure Advisor recommendations

Azure Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments. Currently, Azure provides no Azure Advisor recommendations that are specific to Application Insights.

The following are recommendations that can help you improve the cost effectiveness and operational excellence of Azure Monitor.

- Cost Optimization
- Operational Excellence

Related content

- Application Insights
- Monitoring and diagnostics guidance
- Azure Well-Architected Framework perspective on Log Analytics
- Enable application monitoring in Azure App Service for .NET, Node.js, Python, and Java applications
- Enable Azure Monitor OpenTelemetry for .NET, Node.js, Python, and Java applications

---

Feedback

Was this page helpful? Yes No

Azure Well-Architected Framework perspective on Azure Firewall

Article • 09/26/2024

Azure Firewall is a cloud-native and intelligent network firewall security service that provides best-of-breed threat protection for your cloud workloads that run in Azure. It's a fully stateful, managed firewall service that has built-in high availability and unrestricted cloud scalability. Azure Firewall provides both east-west and north-south traffic inspection.

This article assumes that as an architect, you've reviewed the virtual network security options and chosen Azure Firewall as the network security service for your workload. The guidance in this article provides architectural recommendations that are mapped to the principles of the Azure Well-Architected Framework pillars.

> **ⓘ Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern along with design strategies localized to the technology scope.
>
> Also included are *recommendations* on the technology capabilities that can help materialize those strategies. The recommendations don't represent an exhaustive list of all configurations available for Azure Firewall and its dependencies. Instead, they list the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept or optimize your existing environments.
>
> Foundational architecture that demonstrates the key recommendations: **Hub-spoke network topology in Azure**.

Technology scope

This review focuses on the interrelated decisions for the following Azure resources:

- Azure Firewall
- Azure Firewall Manager

Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover fast from failures**.

The **Reliability design principles** provide a high-level design strategy applied for individual components, system flows, and the system as a whole.

Design checklist

Start your design strategy based on the design review checklist for Reliability. Determine its relevance to your business requirements while keeping in mind the policies and the type of architecture that you use. Extend the strategy to include more approaches as needed.

- ✓ **Review the list of Azure Firewall *known issues*.** Azure Firewall products maintain an updated list of known issues. This list contains important information about by-design behavior, fixes under construction, platform limitations, and possible workarounds or mitigation strategies.

- ✓ **Ensure that your Azure Firewall policy adheres to Azure Firewall limits and recommendations.** The policy structure has limits, including the number of rules and rule collection groups, total policy size, source destinations, and target destinations. Be sure to compose your policy and stay below the documented thresholds.

- ✓ **Deploy Azure Firewall across multiple availability zones** for a higher service-level agreement (SLA). Azure Firewall provides different SLAs depending on whether you deploy the service in a single availability zone or multiple zones. For more information, see SLAs for online services.

- ✓ **Deploy an Azure Firewall instance in each region** in multi-region environments. For traditional hub-and-spoke architectures, see Multi-region considerations. For secured Azure Virtual WAN hubs, configure routing intent and policies to secure inter-hub and branch-to-branch communications. For failure-resistant and fault-tolerant workloads, consider instances of Azure Firewall and Azure Virtual Network as regional resources.

- ✓ **Monitor Azure Firewall metrics and the resource health state.** Azure Firewall integrates with Azure Resource Health. Use the Resource Health check to view the health status of Azure Firewall and address service problems that might affect your Azure Firewall resource.

- ✓ **Deploy Azure Firewall in hub virtual networks or as part of Virtual WAN hubs.**

> **Note**
>
> The availability of network services differs between the traditional hub-and-spoke model and the Virtual WAN-managed secured hubs model. For example, in a Virtual WAN hub, the Azure Firewall public IP can't come from a public IP prefix and can't have Azure DDoS Protection enabled. When you choose your model, consider your requirements across all five pillars of the Well-Architected Framework.

Recommendations

Expand table

Recommendation	Benefit
Deploy Azure Firewall across multiple availability zones.	Deploy Azure Firewall across multiple availability zones to maintain a specific level of resiliency. If one zone experiences an outage, another zone continues to serve traffic.
Monitor Azure Firewall metrics in a Log Analytics workspace. Closely monitor metrics that indicate the Azure Firewall health state, such as *throughput*, *Firewall health state*, *SNAT port utilization*, and *AZFW latency probe* metrics. Use Azure Service Health to monitor Azure Firewall health.	Monitor resource metrics and service health so you can detect when a service state degrades and take proactive measures to prevent failures.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload.

The **Security design principles** provide a high-level design strategy for achieving those goals by applying approaches to the technical design of Azure Firewall.

Design checklist

Start your design strategy based on the design review checklist for Security. Identify vulnerabilities and controls to improve the security posture. Extend the strategy to include more approaches as needed.

- ✓ **Send all internet traffic from your workload through a firewall or a network virtual appliance (NVA)** to detect and block threats. Configure user-defined routes (UDRs) to force traffic through Azure Firewall. For web traffic, consider using Azure Firewall as an explicit proxy.

  Configure supported partner software as a service (SaaS) security providers within Firewall Manager if you want to use these providers to protect outbound connections.

  Restrict the usage of public IP addresses that are directly tied to virtual machines so that traffic can't bypass the firewall. The Azure Cloud Adoption Framework model assigns a specific Azure policy to the CORP management group.

  Follow the Zero Trust configuration guide for Azure Firewall and Application Gateway if your security needs require that you implement a Zero Trust approach for web applications, such as adding inspection and encryption. Follow this guide to integrate Azure Firewall and Application Gateway for both traditional hub-and-spoke and Virtual WAN scenarios.

  For more information, see Apply firewalls at the edge.

- ✓ **Establish network perimeters** as part of your workload segmentation strategy to control the blast radius, obfuscate workload resources, and block unexpected, prohibited, and unsafe access. Create rules for Azure Firewall policies based on the least-privilege access criteria.

  Set the public IP address to *None* to deploy a fully private data plane when you configure Azure Firewall in forced tunneling mode. This approach doesn't apply to Virtual WAN.

  Use fully qualified domain names (FQDN) and service tags when you define network rules to simplify management.

- ✓ **Use detection mechanisms** to diligently monitor for threats and signs of abuse. Take advantage of platform-provided detection mechanisms and measures. Enable the intrusion detection and prevention system (IDPS). Associate an Azure DDoS Protection plan with your hub virtual network.

  For more information, see Detect abuse.

Recommendations

Expand table

Recommendation	Benefit
Configure Azure Firewall in forced tunneling mode if you need to route all internet-bound traffic to a designated next hop instead of directly to the internet. This recommendation doesn't apply to Virtual WAN. Azure Firewall must have direct internet connectivity. If your **AzureFirewallSubnet** learns a default route to your on-premises network via the Border Gateway Protocol, you must configure Azure Firewall in forced tunneling mode. You can use the forced tunneling feature to add another /26 address space for the Azure Firewall Management subnet. Name the subnet **AzureFirewallManagementSubnet**. If you have an existing Azure Firewall instance that you can't reconfigure in forced tunneling mode, create a UDR with a 0.0.0.0/0 route. Set the **NextHopType** value as **Internet**. To maintain internet connectivity, associate the UDR with **AzureFirewallSubnet**. Set the public IP address to **None** to deploy a fully private data plane when you configure Azure Firewall in forced tunneling mode. But the management plane still requires a public IP for management purposes only. The internal traffic from virtual and on-premises networks doesn't use that public IP.	Use forced tunneling so you don't expose your Azure resources directly to the internet. This approach reduces the attack surface and minimizes the risk of external threats. To enforce corporate policies and compliance requirements more effectively, route all internet-bound traffic through an on-premises firewall or an NVA.
Create rules for Firewall policies in an hierarchical structure to overlay a central base policy. For more information, see Use Azure Firewall policies to process rules. Create your rules based on the least-privilege access Zero Trust principle	Organize rules in a hierarchical structure so that granular policies can meet the requirements of specific regions. Each policy can contain different sets of Destination Network Address Translation (DNAT), network, and application rules that have specific priorities, actions, and processing orders.
Configure supported security partner providers within Firewall Manager to protect outbound connections. This scenario requires Virtual WAN with a S2S VPN gateway in the hub because it uses an IPsec tunnel to connect to the provider's infrastructure. Managed security service providers might charge extra license fees and limit throughput on IPsec connections. You can also use alternative solutions, such as Zscaler Cloud Connector.	Enable security partner providers in Azure Firewall to take advantage of best-in-breed cloud security offerings, which provide advanced protection for your internet traffic. These providers offer specialized, user-aware filtering and comprehensive threat-detection capabilities that enhance your overall security posture.

Recommendation	Benefit
Enable Azure Firewall DNS proxy configuration. Also configure Azure Firewall to use custom DNS for forwarding DNS queries.	Enable this feature to point clients in the virtual networks to Azure Firewall as a DNS server. This feature protects internal DNS infrastructure that's not directly accessed and exposed.
Configure UDRs to force traffic through Azure Firewall in a traditional hub-and-spoke architecture for *spoke-to-spoke*, *spoke-to-internet*, and *spoke-to-hybrid* connectivity. In Virtual WAN, configure routing intent and policies to redirect private traffic or internet traffic through the Azure Firewall instance that's integrated into the hub. If you can't apply a UDR, and you only require web traffic redirection, use Azure Firewall as an explicit proxy on the outbound path. You can configure a proxy setting on the sending application, such as a web browser, when you configure Azure Firewall as a proxy.	Send traffic through the firewall to inspect traffic and help identify and block malicious traffic. Use Azure Firewall as an explicit proxy for outbound traffic so that web traffic reaches the firewall's private IP address and therefore egresses directly from the firewall without using a UDR. This feature also facilitates the use of multiple firewalls without modifying existing network routes.
Use FQDN filtering in network rules. You must enable the Azure Firewall DNS proxy configuration to use FQDNs in your network rules.	Use FQDNs in Azure Firewall network rules so that administrators can manage domain names instead of multiple IP addresses, which simplifies management. This dynamic resolution ensures that firewall rules automatically update when domain IPs change.
Use Azure Firewall service tags in place of specific IP addresses to provide selective access to specific services in Azure, Microsoft Dynamics 365, and Microsoft 365.	Use service tags in network rules so you can define access controls based on service names rather than specific IP addresses, which simplifies security management. Microsoft manages and updates these tags automatically when IP addresses change. This method ensures that your firewall rules remain accurate

Recommendation	Benefit
	and effective without manual intervention.
Use FQDN tags in application rules to provide selective access to specific Microsoft services. You can use an FQDN tag in application rules to allow the required outbound network traffic through your firewall for specific Azure services, such as Microsoft 365, Windows 365, and Microsoft Intune.	Use FQDN tags in Azure Firewall application rules to represent a group of FQDNs that are associated with well-known Microsoft services. This method simplifies the management of network security rules.
Enable threat intelligence on Azure Firewall in *Alert and deny* mode.	Use threat intelligence to provide real-time protection against emerging threats, which reduces the risk of cyberattacks. This feature uses the Microsoft threat intelligence feed to automatically alert and block traffic from known malicious IP addresses, domains, and URLs.
Enable the IDPS in *Alert* or *Alert and deny* mode. Consider the performance impact of this feature.	Enable IDPS filtering in Azure Firewall provides real-time monitoring and analysis of network traffic to detect and prevent malicious activities. This feature uses signature-based detection to swiftly identify known threats and block them before they cause harm. For more information, see Detect abuse.
Use an internal enterprise certification authority (CA) to generate certificates when you use TLS inspection with Azure Firewall Premium. Use self-signed certificates only for testing and proof of concept (PoC) purposes.	Enable TLS inspection so that Azure Firewall Premium terminates and inspects TLS connections to detect, alert, and mitigate malicious activity in HTTPS.
Use Firewall Manager to create and associate an Azure DDoS Protection plan with your hub virtual network. This approach doesn't apply to Virtual WAN.	Configure an Azure DDoS Protection plan so that you can centrally manage DDoS protection alongside your firewall policies. This approach

Recommendation	Benefit
	streamlines how you manage your network security and simplifies how you deploy and monitor processes.

Cost Optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget while meeting business requirements.

The Cost Optimization design principles provide a high-level design strategy for achieving those goals and making tradeoffs as necessary in the technical design related to Azure Firewall and its environment.

Design checklist

Start your design strategy based on the design review checklist for Cost Optimization for investments. Fine-tune the design so that the workload is aligned with the budget that's allocated for the workload. Your design should use the right Azure capabilities, monitor investments, and find opportunities to optimize over time.

- ✓ **Select an Azure Firewall SKU to deploy.** Choose from three Azure Firewall SKUs: Basic, Standard, and Premium. Use Azure Firewall Premium to secure highly sensitive applications, such as payment processing. Use Azure Firewall Standard if your workload needs a Layer 3 to Layer 7 firewall and needs autoscaling to handle peak traffic periods of up to 30 Gbps. Use Azure Firewall Basic if you use SMB and require up to 250 Mbps of throughput. You can downgrade or upgrade between Standard and Premium SKUs. For more information, see Choose the right Azure Firewall SKU.

- ✓ **Remove unused firewall deployments and optimize underused deployments.** Stop Azure Firewall deployments that don't need to continuously run. Identify and delete unused Azure Firewall deployments. To reduce operational costs, monitor and optimize firewall instances usage, Azure Firewall Manager policies configuration, and the number of public IP addresses and policies that you use.

- ✓ **Share the same instance of Azure Firewall.** You can use a central instance of Azure Firewall in the hub virtual network or Virtual WAN secure hub and share the same Azure Firewall instance across spoke virtual networks that connect to the same hub

from the same region. Ensure that you don't have unexpected cross-region traffic in a hub-and-spoke topology.

- ✓ **Optimize traffic through the firewall.** Regularly review traffic that Azure Firewall processes. Find opportunities to reduce the amount of traffic that traverses the firewall.

- ✓ **Decrease the amount of log data that you store.** Azure Firewall can use Azure Event Hubs to comprehensively log the traffic's metadata and send it to Log Analytics workspaces, Azure Storage, or non-Microsoft solutions. All logging solutions incur costs to process data and provide storage. Large amounts of data can incur significant costs. Consider a cost-effective approach and alternative to Log Analytics, and estimate the cost. Consider whether you need to log traffic metadata for all logging categories.

Recommendations

Expand table

Recommendation	Benefit
Stop Azure Firewall deployments that don't need to continuously run. You might have development or testing environments that you only use during business hours. For more information, see Deallocate and allocate Azure Firewall.	Shut down these deployments during off-peak hours or when idle to reduce unnecessary expenses but maintain security and performance during critical times.
Regularly review traffic that Azure Firewall processes, and find originating workload optimizations. The top flows log, also known as the *fat flows log*, shows the top connections that contribute to the highest throughput through the firewall.	Optimize workloads that generate the most traffic through the firewall to reduce the volume of traffic, which decreases the load on the firewall and minimizes data-processing and bandwidth costs.
Identify and delete unused Azure Firewall deployments. Analyze monitoring metrics and UDRs that are associated with subnets that point to the firewall's private IP. Also consider other validations and internal documentation about your environment and deployments. For example, analyze any classic NAT, network, and application rules for Azure Firewall. And consider your settings. For example, you might configure the DNS proxy setting to **Disabled**.	Use this approach to detect cost-effective deployments over time and eliminate unused resources, which prevents unnecessary costs.

Recommendation	Benefit
For more information, see Monitor Azure Firewall.	
Review your Firewall Manager policies, associations, and inheritance carefully to optimize cost. Policies are billed based on firewall associations. A policy with zero or one firewall association is free. A policy with multiple firewall associations is billed at a fixed rate. For more information, see Firewall Manager pricing.	Properly use Firewall Manager and its policies to reduce operational costs, increase efficiency, and reduce management overhead.
Review all the public IP addresses in your configuration, and disassociate and delete the ones that you don't use. Evaluate source network address translation (SNAT) port usage before you remove any IP addresses. For more information, see Monitor Azure Firewall logs and metrics and SNAT port usage.	Delete unused IP addresses to reduce costs.

Operational Excellence

Operational Excellence primarily focuses on procedures for **development practices, observability, and release management**.

The Operational Excellence design principles provide a high-level design strategy for achieving those goals for the operational requirements of the workload.

Design checklist

Start your design strategy based on the design review checklist for Operational Excellence for defining processes for observability, testing, and deployment related to Azure Firewall.

- ✓ **Use Firewall Manager with traditional hub-and-spoke topologies or Virtual WAN network topologies** to deploy and manage instances of Azure Firewall. Use native security services for traffic governance and protection to create hub-and-spoke and transitive architectures. For more information, see Network topology and connectivity.

  Migrate Azure Firewall classic rules to Firewall Manager policies for existing deployments. Use Firewall Manager to centrally manage your firewalls and policies. For more information, see Migrate to Azure Firewall Premium.

- ✓ **Maintain regular backups of Azure Policy artifacts.** If you use an infrastructure-as-code approach to maintain Azure Firewall and all dependencies, you should have backup and versioning of Azure Firewall policies in place. If you don't, you can deploy a companion mechanism that's based on an external logic app to provide an effective automated solution.
- ✓ **Monitor Azure Firewall logs and metrics.** Take advantage of diagnostic logs for firewall monitoring and troubleshooting and activity logs for auditing operations.
- ✓ **Analyze monitoring data to assess the overall health of the system.** Use the built-in Azure Firewall monitoring workbook, familiarize yourself with Kusto Query Language (KQL) queries, and use the policy analytics dashboard to identify potential problems.
- ✓ **Define alerts for key events** so that operators can quickly respond to them.
- ✓ **Take advantage of platform-provided detection mechanisms in Azure to detect abuse.** Integrate Azure Firewall with Microsoft Defender for Cloud and Microsoft Sentinel if possible. Integrate with Defender for Cloud so you can visualize the status of network infrastructure and network security in one place, including Azure network security across all virtual networks and virtual hubs in different regions in Azure. Integrate with Microsoft Sentinel to provide threat-detection and prevention capabilities.

Recommendations

Expand table

Recommendation	Benefit
Enable diagnostic logs for Azure Firewall. Use firewall logs or workbooks to monitor Azure Firewall. You can also use activity logs to audit operations on Azure Firewall resources. Use the structured firewall logs format. Only use the previous diagnostic logs format if you have an existing tool that requires it. Don't enable both logging formats at the same time.	Enable diagnostic logs to optimize your monitoring tools and strategies for Azure Firewall. Use structured firewall logs to structure log data so that it's easy to search, filter, and analyze. The latest monitoring tools are based on this type of log, so it's often a prerequisite.
Use the built-in Azure Firewall workbook.	Use the Azure Firewall workbook to extract valuable insights from Azure Firewall events, analyze your application and network rules, and examine statistics

Recommendation	Benefit
	about firewall activities across URLs, ports, and addresses.
Monitor Azure Firewall logs and metrics, and create alerts for Azure Firewall capacity. Create alerts to monitor *Throughput*, *Firewall health state*, *SNAT port utilization*, and *AZFW latency probe* metrics.	Set up alerts for key events to notify operators before potential problems arise, help prevent disruptions, and initiate quick capacity adjustments.
Regularly review the policy analytics dashboard to identify potential problems.	Use policy analytics to analyze the impact of your Azure Firewall policies. Identify potential problems in your policies, such as meeting policy limits, improper rules, and improper IP groups usage. Get recommendations to improve your security posture and rule-processing performance.
Understand KQL queries so you can use Azure Firewall logs to quickly analyze and troubleshoot problems. Azure Firewall provides sample queries.	Use KQL queries to quickly identify events inside your firewall and check to see which rule is triggered or which rule allows or blocks a request.

Performance Efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance.

The Performance Efficiency design principles provide a high-level design strategy for achieving those capacity goals against the expected usage.

Design checklist

Start your design strategy based on the design review checklist for Performance Efficiency. Define a baseline that's based on key performance indicators for Azure Firewall.

- ✓ **Optimize your Azure Firewall configuration** in accordance with the Well-Architected Framework recommendations to optimize code and infrastructure and ensure peak operation. To maintain an efficient and secure network, regularly review and optimize firewall rules. This practice helps ensure that your firewall configurations remain effective and up to date with the latest security threats.

Assess policy requirements, and find opportunities to summarize IP ranges and URL lists. Use web categories to allow or deny outbound access in bulk to streamline management and enhance security. Evaluate the performance impact of IDPS in *Alert and deny* mode because this configuration can affect network latency and throughput. Configure public IP addresses to support your SNAT port requirements. Follow these practices to create a robust and scalable network security infrastructure.

- ✓ **Don't use Azure Firewall for intra-virtual network traffic control.** Use Azure Firewall to control the following types of traffic:
  - Traffic across virtual networks
  - Traffic between virtual networks and on-premises networks
  - Outbound traffic to the internet
  - Incoming non-HTTP or non-HTTPS traffic

  For intra-virtual network traffic control, use network security groups.

- ✓ **Warm up Azure Firewall properly before performance tests.** Create initial traffic that isn't part of your load tests 20 minutes before your tests. Use diagnostics settings to capture scale-up and scale-down events. You can use the Azure Load Testing service to generate the initial traffic so you can scale up Azure Firewall to the maximum number of instances.

- ✓ **Configure an Azure Firewall subnet with a /26 address space.** You need a dedicated subnet for Azure Firewall. Azure Firewall provisions more capacity as it scales. A /26 address space ensures that the firewall has enough IP addresses available to accommodate the scaling. Azure Firewall doesn't require a subnet that's larger than /26. Name the Azure Firewall subnet **AzureFirewallSubnet**.

- ✓ **Don't enable advanced logging if you don't need it.** Azure Firewall provides some advanced logging capabilities that can incur significant costs to keep active. Instead, you can use these capabilities for troubleshooting purposes only and for limited amounts of time. Disable capabilities when you don't need them. For example, top flows and flow trace logs are expensive and can cause excessive CPU and storage usage on the Azure Firewall infrastructure.

Recommendations

Expand table

Recommendation	Benefit
Use the policy analytics dashboard to identify ways to optimize Azure Firewall policies.	Use policy analytics to identify potential problems in your policies, such as meeting policy limits, improper rules, and improper IP groups usage. Get recommendations to improve your security posture and rule-processing performance.
Place frequently used rules early in a group to optimize latency for Azure Firewall policies that have large rule sets. For more information, see Use Azure Firewall policies to process rules.	Place frequently used rules high in a rule set to optimize processing latency. Azure Firewall processes rules based on the rule type, inheritance, rule collection group priority, and rule collection priority. Azure Firewall processes high-priority rule collection groups first. Inside a rule collection group, Azure Firewall processes rule collections that have the highest priority first.
Use IP groups to summarize IP address ranges and avoid exceeding the limit of unique source or unique destination network rules. Azure Firewall treats the IP group as a single address when you create network rules.	This approach effectively increases the number of IP addresses that you can cover without exceeding the limit. For each rule, Azure multiplies ports by IP addresses. So, if one rule has four IP address ranges and five ports, you consume 20 network rules.
Use Azure Firewall web categories to allow or deny outbound access in bulk, instead of explicitly building and maintaining a long list of public internet sites.	This feature dynamically categorizes web content and permits the creation of compact application rules, which reduces operational overhead.
Evaluate the performance impact of IDPS in *Alert and deny* mode. For more information, see Azure Firewall performance.	Enable IDPS in *Alert and deny* mode to detect and prevent malicious network activity. This feature might introduce a performance penalty. Understand the effect on your workload so you can plan accordingly.
Configure Azure Firewall deployments with a minimum of five public IP addresses for deployments that are susceptible to SNAT port exhaustion.	Azure Firewall supports 2,496 ports for each public IP address that each back-end Azure Virtual Machine Scale Sets instance uses. This configuration increases the available SNAT ports by five times. By default, Azure Firewall deploys two Virtual Machine Scale Sets instances that support 4,992 ports for each flow destination IP, destination port, and TCP or UDP protocol. The firewall scales up to a maximum of 20 instances.

Azure policies

Azure provides an extensive set of built-in policies related to Azure Firewall and its dependencies. Some of the preceding recommendations can be audited through Azure Policy. For example, you can check whether:

- Network interfaces shouldn't have public IPs. This policy denies network interfaces that are configured with a public IP. Public IP addresses allow internet resources to communicate inbound to Azure resources, and Azure resources can communicate outbound to the internet.
- All internet traffic should be routed via your deployed Azure Firewall instance. Azure Security Center identifies that some of your subnets aren't protected with a next generation firewall. Protect your subnets from potential threats. Use Azure Firewall or a supported next generation firewall to restrict access to your subnets.

For comprehensive governance, review the Azure Policy built-in definitions for Azure Firewall and other policies that might affect the security of the network.

Azure Advisor recommendations

Azure Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments. Here are some recommendations that can help you improve the reliability, security, cost effectiveness, performance, and operational excellence of Azure Firewall.

- Reliability
- Security
- Cost Optimization
- Performance
- Operational Excellence

Next steps

See the following resources that demonstrate the recommendations in this article.

- Use the following reference architectures as examples of how to apply this article's guidance to a workload:
  - Hub-spoke network topology in Azure
  - Azure Firewall architecture overview
  - Use Azure Firewall to help protect an Azure Kubernetes Service (AKS) cluster
- Use the following resources to improve your implementation expertise:
  - Azure Firewall documentation

    - What is Firewall Manager?

- See other resources:
    - Azure Firewall service limits, quotas, and constraints
    - Azure security baseline for Azure Firewall
    - Use the Azure portal to deploy and configure Azure Firewall and policy
    - Configure Azure Firewall in a Virtual WAN hub

Feedback

Was this page helpful? Yes No

Architecture best practices for Azure App Service (Web Apps)

Article • 02/12/2025

Azure App Service is a platform as a service (PaaS) that provides a fully managed hosting environment for building, deploying, and scaling web applications. As a PaaS solution, App Service abstracts the underlying infrastructure, which enables you to focus on your application development. The Web Apps feature of App Service runs your apps in the context of an App Service plan. This plan determines the resources, operating system, region, and pricing model (SKU) that's used to host your app.

This article assumes that as an architect, you've reviewed the compute decision tree and chose App Service as the compute for your workload. The guidance in this article provides architectural recommendations that are mapped to the principles of the Well-Architected Framework pillars.

> **ⓘ Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern along with design strategies localized to the technology scope.
>
> Also included are recommendations for the technology capabilities that can help materialize those strategies. The recommendations don't represent an exhaustive list of all configurations that are available for the Web Apps feature of App Service and their dependencies. Instead, they list the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept or to optimize your existing environments.
>
> Foundational architecture that demonstrates the key recommendations: App Service baseline architecture.

Technology scope

This review focuses on the interrelated decisions for the following Azure resources:

- App Service plans
- Web Apps

Other Azure offerings are associated with App Service, such as Azure Functions, Azure Logic Apps, and App Service Environment. Those offerings are out of scope for this article. App Service Environment is referenced occasionally to help clarify features or options of the core App Service offerings.

Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover fast from failures**.

The Reliability design principles provide a high-level design strategy applied for individual components, system flows, and the system as a whole.

Design checklist

Start your design strategy based on the design review checklist for Reliability. Determine its relevance to your business requirements while keeping in mind the tiers and features of App Service and its dependencies. Extend the strategy to include more approaches as needed.

- ✓ **Prioritize user flows:** Not all flows are equally important. Identify the critical paths in your application and assign priorities to each flow to guide your design decisions. User flow design can influence which service tiers and the number of instances that you choose for an App Service plan and configuration.

  For example, your application might include front-end and back-end tiers that communicate through a message broker. You might choose to segment the tiers in multiple web apps to allow for independent scaling, lifecycle management, and maintenance. If you place a large application in a single plan, it can result in memory or CPU problems that affect reliability.

  You might need more instances on the front end for optimal performance on the UI side. However, the back end might not require the same number of instances.

- ✓ **Anticipate potential failures:** Plan mitigation strategies for potential failures. The following table shows examples of failure mode analysis.

Expand table

Failure	Mitigation
Failure of underlying or abstracted App Service	Have component redundancy in instances and dependencies. Monitor the health of instances, network

Failure	Mitigation
components	performance, and storage performance.
Failure of external dependencies	Use design patterns such as the Retry pattern and the Circuit Breaker pattern. Monitor the external dependencies and set appropriate time-outs.
Failure because of traffic getting routed to unhealthy instances	Monitor instance health. Consider responsiveness, and avoid sending requests to unhealthy instances.

For more information, see Failure mode analysis for Azure applications.

- ✓ **Build redundancy:** Build redundancy in the application and supporting infrastructure. Spread instances across availability zones to improve fault tolerance. Traffic is routed to other zones if one zone fails. Deploy your application across multiple regions to help ensure that your app remains available, even if an entire region experiences an outage.

  Build similar levels of redundancy in dependent services. For instance, application instances bind to blob storage. Consider configuring the associated storage account with zone-redundant storage if an application uses a zone-redundant deployment.

- ✓ **Use multiple instances:** An immediate single-point-of-failure occurs if you run your app on only one instance. Allocate multiple instances to your app to help ensure high availability. If one instance fails, other instances can still handle incoming requests. Your app code should be able to handle multiple instances without synchronization issues when reading from data sources or writing to data sources.

  Have redundancy in networking components. For example, use zone-redundant IP addresses and load balancers.

- ✓ **Have a reliable scaling strategy:** Unexpected load on an application can make it unreliable. Consider the right scaling approach based on your workload characteristics. Horizontal scaling, or *scaling out*, allows you to add more instances to distribute the load. Vertical scaling, or *scaling up*, increases the capacity of an existing instance, such as CPU or memory. Be cautious of over-provisioning because adding unnecessary instances increases costs without tangible performance benefits.

  You can sometimes scale up to handle the load. However, if the load continues to increase, scale out to new instances. Choose the automatic, or *autoscaling*,

approach over the manual approach. Always maintain a buffer of extra capacity during scaling operations to prevent performance degradation.

The App Service plan tier that you choose affects scaling in terms of the number of instances and the compute units.

- ✓ **Ensure proper app initialization so that new instances warm up quickly and can receive requests.** Strive for stateless applications when possible. Reliably scaling state with new instances can increase complexity. Consider an external data store that you can scale independently if you need to store application state. Storing session state in memory can result in losing session state when there's a problem with the application or App Service. It also limits the possibility of spreading the load across other instances.

  Regularly test your autoscaling rules. Simulate load scenarios to verify that your app scales as expected. You should log scaling events so that you can troubleshoot problems that might arise and optimize your scaling strategy over time.

  App Service has a limitation on the number of instances within a plan, which can affect scaling reliability. To address this, one strategy is to deploy identical stamps, with each stamp running its own App Service plan instance and endpoint. It's essential that you front all stamps with an external load balancer to distribute traffic across them. Use Azure Application Gateway for single-zone deployments and Azure Front Door for multiple-region deployments. This approach is ideal for mission-critical applications when reliability is crucial. For more information, see Mission-critical baseline with App Service.

- ✓ **Plan your recoverability:** Redundancy is crucial for business continuity. Fail over to another instance if one instance is unreachable. Explore automatic healing capabilities in App Service, such as automatic repair of instances.

  Implement design patterns to handle graceful degradation for transient failures, such as network connectivity problems and large-scale events like regional outages. Consider the following design patterns:

  - *The Bulkhead pattern* segments your application into isolated groups to prevent a failure from affecting the entire system.
  - *The Queue-Based Load Leveling pattern* queues work items that serve as a buffer to smooth out traffic spikes.
  - *The Retry pattern* handles transient failures because of network glitches, dropped database connections, or busy services.

- *The Circuit Breaker pattern* prevents an application from repeatedly trying to perform an operation that's likely to fail.

You can use WebJobs to run background tasks in your web app. To run those tasks reliably, ensure that the app that hosts your job runs continuously on a schedule or based on event-driven triggers.

For more information, see Cloud design patterns that support reliability.

- ✓ **Conduct reliability testing:** Conduct load testing to evaluate your application's reliability and performance under load. Test plans should include scenarios that validate your automated recovery operations.

  Use fault injection to intentionally introduce failures and validate your self-healing and self-preservation mechanisms. For more information, see Azure Chaos Studio fault and action library.

  App Service imposes resource limits on hosted apps. The App Service plan determines these limits. Make sure that your tests confirm that the app runs within those resource limits. For more information, see App Service limits.

- ✓ **Use the health check feature to identify unresponsive workers:** App Service has built-in capabilities that periodically ping a specific path of your web application. The platform pings this path to determine whether your application is healthy and responds to requests.

  When your site is scaled out to multiple instances, App Service excludes any unhealthy instances from serving requests. This process improves your overall availability.

  Your app's health check path should poll the essential components of your application, such as your database, cache, or messaging service. This step helps ensure that the status returned by the health check path is an accurate picture of the overall health of your application.

- ✓ **Use the auto-heal feature:** Sometimes your application might experience unexpected behaviors that a simple restart can solve. Use the auto-heal feature to define a *condition* that triggers auto-heal and the *action* that auto-heal initiates when that condition is met. For more information, see App Service diagnostics overview.

- ✓ **App Service resiliency score report:** To review tailored best practice recommendations, see the App Service diagnostics overview.

Recommendations

Expand table

Recommendation	Benefit
(App Service) Choose the Premium v3 tier of an App Service plan for production workloads. Set the maximum and minimum number of workers according to your capacity planning. For more information, see App Service plan overview.	A Premium v3 App Service plan provides advanced scaling features and ensures redundancy if failures occur.
(App Service) Enable zone redundancy. Consider provisioning more than three instances to enhance fault tolerance. Check regional support for zone redundancy because not all regions have this feature.	Your application can withstand failures in a single zone when multiple instances are spread across zones. Traffic automatically shifts to healthy instances in other zones and maintains application reliability if one zone is unavailable.
(Web Apps) Consider disabling the application request routing (ARR) affinity feature. ARR affinity creates sticky sessions that redirect users to the node that handled their previous requests.	Incoming requests are evenly distributed across all available nodes when you disable ARR affinity. Evenly distributed requests prevent traffic from overwhelming any single node. Requests can be seamlessly redirected to other healthy nodes if a node is unavailable. Avoid session affinity to ensure that your App Service instance remains stateless. A stateless App Service instance reduces complexity and ensures consistent behavior across nodes. Remove sticky sessions so that App Service can add or remove instances to scale horizontally.
(Web Apps) Define automatic healing rules based on request count, slow requests, memory limits, and other indicators that are part of your performance baseline. Consider this configuration as part of your scaling strategy.	Automatic healing rules help your application recover automatically from unexpected problems. The configured rules trigger healing actions when thresholds are breached. Automatic healing enables automatic proactive maintenance.
(Web Apps) Enable the health check feature and provide a path that responds to the health check requests.	Health checks can detect problems early. Then the system can automatically take corrective actions when a health check request fails. The load balancer routes traffic away from

Recommendation	Benefit
	unhealthy instances, which directs users to healthy nodes.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload.

The Security design principles provide a high-level design strategy for achieving those goals by applying approaches to the technical design around hosting on App Service.

Design checklist

Start your design strategy based on the design review checklist for Security and identify vulnerabilities and controls to improve the security posture. Extend the strategy to include more approaches as needed.

- ✓ **Review security baselines:** To enhance the security posture of your application that's hosted on an App Service plan, review the security baseline for App Service.
- ✓ **Use the latest runtime and libraries:** Thoroughly test your application builds before you do updates to catch problems early and help ensure a smooth transition to the new version. App Service supports the language runtime support policy for updating existing stacks and retiring end-of-support stacks.
- ✓ **Create segmentation through isolation boundaries to contain breaches:** Apply identity segmentation. For example, implement role-based access control (RBAC) to assign specific permissions based on roles. Follow the principle of least privilege to limit access rights to only what's necessary. Also create segmentation at the network level. Integrate App Service apps with an Azure virtual network for isolation and define network security groups (NSGs) to filter traffic.

  App Service plans provide the App Service Environment tier that provides a high degree of isolation. With App Service Environment, you get dedicated compute and networking.
- ✓ **Apply access controls on identities:** Restrict inward access to the web app and outward access from the web app to other resources. This configuration applies access controls on identities and helps maintain the workload's overall security posture.

Use Microsoft Entra ID for all authentication and authorization needs. Use built-in roles, such as a Web Plan Contributor, Website Contributor, and a generic Contributor, Reader, and Owner.

- ✓ **Apply network security controls:** Integrate your App Service with a virtual network to control the outbound traffic. Use private endpoints to control inbound traffic, limit access to your App Service instance from within your virtual network, and disable public internet access. For more information, see Network routing.

  Deploy a web application firewall (WAF) to help protect against common vulnerabilities. Both Application Gateway and Azure Front Door have integrated WAF capabilities.

  Configure the reverse proxy rules and network settings appropriately to achieve the desired level of security and control. For example, add NSG rules on the private endpoint subnet to only accept traffic from the reverse proxy.

  Egress traffic from the application to other PaaS services should be over private endpoints. Consider placing a firewall component to restrict egress traffic to the public internet. Both approaches help prevent data exfiltration.

  For more information, see App Service networking features.

- ✓ **Encrypt data:** Help protect data in transit by using end-to-end Transport Layer Security (TLS). Use your customer-managed keys for full encryption of data at rest.

  Don't use legacy protocols such as TLS 1.0 and 1.1. Ensure that all web apps use HTTPS only and enforce TLS 1.2 or higher. The default minimum TLS version is 1.2. For more information, see App Service TLS overview.

  All of your App Service instances have a default domain name. Use a custom domain and secure that domain by using certificates.

- ✓ **End-to-end TLS encryption:** End-to-end TLS encryption is available in premium App Service plans. This feature encrypts your traffic throughout the entire transaction, which minimizes the risk of traffic interception.

- ✓ **Reduce the attack surface:** Remove default configurations that you don't need. For example, disable remote debugging, local authentication for Source Control Manager (SCM) sites, and basic authentication. Disable unsecured protocols like HTTP and File Transfer Protocol (FTP). Enforce configurations by using Azure policies. For more information, see Azure policies.

- ✓ **Implement restrictive cross-origin resource sharing (CORS) policies:** Use restrictive CORS policies in your web app to only accept requests from the allowed domains,

headers, and other criteria. Enforce CORS policies by using built-in Azure policy definitions.

- ✓ **Use managed identities:** Enable managed identities for your App Service instance to more securely access other Azure services without needing to manage credentials.
- ✓ **Protect application secrets:** You need to handle sensitive information, like API keys or authentication tokens. Instead of hardcoding these secrets directly into your application code or configuration files, you can use Azure Key Vault references in app settings. When the application starts, App Service automatically retrieves the secret values from Key Vault by using the app's managed identity.
- ✓ **Enable resource logs for your application:** Enable resource logs for your application to create comprehensive activity trails that provide valuable data during investigations that follow security incidents. For more information, see Azure Monitor resource logs.

  Consider logging as part of your threat modeling process when you assess threats.

Recommendations

Expand table

Recommendation	Benefit
(Web Apps) Assign managed identities to the web app. To maintain isolation boundaries, don't share or reuse identities across applications. Make sure that you securely connect to your container registry if you use containers for your deployment.	The application retrieves secrets from Key Vault to authenticate outward communication from the application. Azure manages the identity and doesn't require you to provision or rotate any secrets. You have distinct identities for granularity of control. Distinct identities make revocation easy if an identity is compromised.
(Web Apps) Configure custom domains for applications. Disable HTTP and only accept HTTPS requests.	Custom domains enable secure communication through HTTPS by using TLS protocol, which helps ensure the protection of sensitive data and builds user trust.
(Web Apps) Evaluate whether App Service built-in authentication is the right mechanism to authenticate	When you use this feature, you don't have to use authentication libraries in

Recommendation	Benefit
users that access your application. App Service built-in authentication integrates with Microsoft Entra ID. This feature handles token validation and user identity management across multiple sign-in providers and supports OpenID Connect. With this feature, you don't have authorization at a granular level, and you don't have a mechanism to test authentication.	application code, which reduces complexity. The user is already authenticated when a request reaches the application.
(Web Apps) Configure the application for virtual network integration. Use private endpoints for App Service apps. Block all public traffic. Route the container image pull through the virtual network integration. All outgoing traffic from the application passes through the virtual network.	Get the security benefits of using an Azure virtual network. For example, the application can securely access resources within the network. Add a private endpoint to help protect your application. Private endpoints limit direct exposure to the public network and allow controlled access through the reverse proxy.
(Web Apps) To implement hardening: - Disable basic authentication that uses a username and password in favor of Microsoft Entra ID-based authentication. - Turn off remote debugging so that inbound ports aren't opened. - Enable CORS policies to tighten incoming requests. - Disable protocols, such as FTP.	We don't recommend basic authentication as a secure deployment method. Microsoft Entra ID employs OAuth 2.0 token-based authentication, which provides numerous advantages and enhancements that address the limitations that are associated with basic authentication. Policies restrict access to application resources, only allow requests from specific domains, and secure cross-region requests.
(Web Apps) Always use Key Vault references as app settings.	Secrets are kept separate from your app's configuration. App settings are encrypted at rest. App Service also manages secret rotations.
(App Service) Enable Microsoft Defender for Cloud for App Service.	Get real-time protection for resources that run in an App Service plan. Guard against threats and enhance your overall security posture.

Recommendation	Benefit
(App Service) Enable diagnostic logging and add instrumentation to your app. The logs are sent to Azure Storage accounts, Azure Event Hubs, and Log Analytics. For more information about audit log types, see Supported log types.	Logging captures access patterns. It records relevant events that provide valuable insights into how users interact with an application or platform. This information is crucial for accountability, compliance, and security purposes.

Cost Optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget while meeting business requirements.

The Cost Optimization design principles provide a high-level design strategy for achieving those goals and making tradeoffs as necessary in the technical design related to Web Apps and its environment.

Design checklist

Start your design strategy based on the design review checklist for Cost Optimization for investments. Fine-tune the design so that the workload is aligned with the budget that's allocated for the workload. Your design should use the right Azure capabilities, monitor investments, and find opportunities to optimize over time.

- ✓ **Estimate the initial cost:** As part of your cost modeling exercise, use the Azure pricing calculator to evaluate the approximate costs associated with various tiers based on the number of instances that you plan to run. Each App Service tier provides different compute options.

  Continuously monitor the cost model to track expenditures.

- ✓ **Evaluate the discounted options:** Higher tiers include dedicated compute instances. You can apply a reservation discount if your workload has a predictable and consistent usage pattern. Make sure that you analyze usage data to determine the type of reservation that suits your workload. For more information, see Save costs with App Service reserved instances.

- ✓ **Understand usage meters:** Azure charges an hourly rate, prorated to the second, based on your App Service plan's pricing tier. Charges apply to each scaled-out instance in your plan, based on the time that you allocate the VM instance. Pay

attention to underused compute resources that might increase your costs as a result of overallocation because of suboptimal SKU selection, or poorly configured scale-in configuration.

Extra App Service features, like custom domain registration and custom certificates, might add costs. Other resources, such as virtual networks that isolate your solution or key vaults that protect workload secrets, can also integrate with your App Service resources and add costs. For more information, see Understand the full billing model for App Service.

- ✓ **Consider the tradeoffs between density and isolation:** You can use App Service plans to host multiple applications on the same compute, which saves costs with shared environments. For more information, see Tradeoffs.

- ✓ **Evaluate the effects of your scaling strategy on cost:** You must properly design, test, and configure for scaling out and for scaling in when you implement autoscaling. Establish precise maximum and minimum limits on autoscaling.

  Proactively initialize the application for reliable scaling. For example, don't wait until the CPU reaches 95% usage. Instead, trigger scaling at around 65% to allow sufficient time for new instances to be allocated and initialized during the scaling process. However, this strategy might result in unused capacity.

  We recommend that you combine and balance mechanisms for scaling up and scaling out. For example, an app can scale up for some time and then scale out as necessary. Explore high tiers that provide large capacity and efficient resource usage. Based on usage patterns, higher Premium tiers are often more cost effective because they're more capable.

- ✓ **Optimize environment costs:** Consider using the basic tier or free tier to run pre-production environments. These tiers are low performance and low cost. If you use the basic tier or free tier, use governance to enforce the tier, constrain the number of instances and CPUs, restrict scaling, and limit log retention.

- ✓ **Implement design patterns:** This strategy reduces the volume of requests that your workload generates. Consider using patterns like the Backends for Frontends pattern and the Gateway Aggregation pattern to minimize the number of requests and reduce costs.

- ✓ **Regularly check data-related costs:** Extended data retention periods or expensive storage tiers can result in high storage costs. More expenses can accumulate because of bandwidth usage and prolonged retention of logging data.

Consider implementing caching to minimize data transfer costs. Start with local in-memory caching, and then explore distributed caching options to reduce the number of requests to the back-end database. Consider the bandwidth traffic costs that are associated with cross-region communication if your database is located in a different region.

- ✓ **Optimize deployment costs:** Take advantage of deployment slots to optimize costs. The slot runs in the same compute environment as the production instance. Use them strategically for scenarios like blue-green deployments that switch between slots. This approach minimizes downtime and ensures smooth transitions.

  Use deployment slots with caution. You can introduce problems, such as exceptions or memory leaks, that might affect the existing instances and new instances. Make sure that you thoroughly test changes. For more information about operational guidance, see Operational Excellence.

Recommendations

Expand table

Recommendation	Benefit
(App Service) Choose free tiers or basic tiers for lower environments. We recommend these tiers for experimental use. Remove the tiers when you no longer need them.	The free tiers and basic tiers are budget-friendly compared to higher tiers. They provide a cost-effective solution for nonproduction environments that don't need the full features and performance of premium plans.
(App Service) Take advantage of discounts and explore preferred pricing for: - Lower environments with dev/test plans. - Azure reservations and Azure savings plans for dedicated compute that you provision in the Premium v3 tier and App Service Environment. Use reserved instances for stable workloads that have predictable usage patterns.	Dev/test plans provide reduced rates for Azure services, which make them cost-effective for nonproduction environments. Use reserved instances to prepay for compute resources and get significant discounts.
(Web Apps) Monitor costs that App Service resources incur. Run the cost analysis tool in the Azure portal.	You can identify cost spikes, inefficiencies, or unexpected expenses early on. This proactive approach helps you to provide budgetary controls to prevent overspending.

Recommendation	Benefit
Create budgets and alerts to notify stakeholders.	
(App Service) Scale in when demand decreases. To scale in, define scale rules to reduce the number of instances in Azure Monitor.	Prevent wastage and reduce unnecessary expenses.

Operational Excellence

Operational Excellence primarily focuses on procedures for **development practices, observability, and release management**.

The Operational Excellence design principles provide a high-level design strategy for achieving those goals for the operational requirements of the workload.

Design checklist

Start your design strategy based on the design review checklist for Operational Excellence for defining processes for observability, testing, and deployment related to Web Apps.

- ✓ **Manage releases:** Use deployment slots to manage releases effectively. You can deploy your application to a slot, perform testing, and validate its functionality. After verification, you can seamlessly move the app to production. This process doesn't incur extra costs because the slot runs in the same virtual machine (VM) environment as the production instance.

  Use the swap with preview (multiple-phase swap) feature. Swap with preview allows you to test the app in your staging slots against your production settings and also warm up the app. After doing your tests and warming up all the necessary paths, you can then complete the swap. Then the app will receive production traffic without restarting.

- ✓ **Run automated tests:** Before you promote a release of your web app, thoroughly test its performance, functionality, and integration with other components. Use Azure Load Testing. It integrates with Apache JMeter, a popular tool for performance testing. Explore automated tools for other types of testing, such as Phantom for functional testing.

- ✓ **Automate deployments:** Use continuous integration and continuous deployment pipelines with Azure DevOps or GitHub Actions to automate deployments and

reduce manual effort. For more information, see Continuous deployment to App Service.

- ✓ **Deploy immutable units:** Implement the Deployment Stamps pattern to compartmentalize App Service into an immutable stamp. App Service supports the use of containers, which are inherently immutable. Consider custom containers for your App Service web app.

  Each stamp represents a self-contained unit that you can quickly scale out or scale in. Units that are based on this stamp are temporary and stateless. Stateless design simplifies operations and maintenance. This approach is ideal for mission-critical applications. For more information, see Mission-critical baseline with App Service.

  Use an infrastructure as code (IaC) technology, such as Bicep, to stamp out units with repeatability and consistency.

- ✓ **Keep production environments safe:** Create separate App Service plans to run production and pre-production environments. Don't make changes directly in the production environment to help ensure stability and reliability. Separate instances allow flexibility in development and testing before you promote changes to production.

  Use low environments to explore new features and configurations in an isolated manner. Keep development and test environments ephemeral.

- ✓ **Manage certificates:** For custom domains, you need to manage TLS certificates.

  Have processes in place to procure, renew, and validate certificates. Offload these processes to App Service if possible. If you use your own certificate, you must manage its renewal. Choose an approach that best aligns with your security requirements.

Expand table

Recommendation	Benefit
(Web Apps) Monitor the health of your instances and activate instance health probes. Set up a specific path for handling health probe requests.	You can detect problems promptly and take necessary actions to maintain availability and performance.

Recommendation	Benefit
(Web Apps) Enable diagnostics logs for the application and the instance. Frequent logging can slow down the performance of the system, add to storage costs, and introduce risk if you have unsecure access to logs. Follow these best practices: - Log the right level of information. - Set retention policies. - Keep an audit trail of authorized access and unauthorized attempts. - Treat logs as data and apply data-protection controls.	Diagnostic logs provide valuable insights into your app's behavior. Monitor traffic patterns and identify anomalies.
(Web Apps) Take advantage of App Service-managed certificates to offload certification management to Azure.	App Service automatically handles processes like certificate procurement, certificate verification, certificate renewal, and importing certificates from Key Vault. Alternatively, upload your certificate to Key Vault and authorize the App Service resource provider to access it.
(App Service) Validate app changes in the staging slot before you swap it with the production slot.	Avoid downtime and errors. Quickly revert to the last-known good state if you detect a problem after a swap.

Performance Efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance.

The Performance Efficiency design principles provide a high-level design strategy for achieving those capacity goals against the expected usage.

Design checklist

Start your design strategy based on the design review checklist for Performance Efficiency. Define a baseline that's based on key performance indicators for Web Apps.

- ✓ **Identify and monitor performance indicators:** Set targets for the key indicators for the application, such as the volume of incoming requests, time that the application takes to respond to requests, pending requests, and errors in HTTP responses. Consider key indicators as part of the performance baseline for the workload.

  Capture App Service metrics that form the basis of performance indicators. Collect logs to gain insights into resource usage and activities. Use application performance monitoring tools, such as Application Insights, to collect and analyze performance data from the application. For more information, see App Service monitoring data reference.

  Include code-level instrumentation, transaction tracing, and performance profiling.

- ✓ **Assess capacity:** Simulate various user scenarios to determine the optimal capacity that you need to handle expected traffic. Use Load Testing to understand how your application behaves under different levels of load.

- ✓ **Choose the right tier:** Use dedicated compute for production workloads. Premium v3 tiers provide larger SKUs with increased memory and CPU capacity, more instances, and more features, such as zone redundancy. For more information, see Premium v3 pricing tier.

- ✓ **Optimize your scaling strategy:** When possible, use autoscaling instead of manually adjusting the number of instances as application load changes. With autoscaling, App Service adjusts server capacity based on predefined rules or triggers. Make sure you do adequate performance testing and set the right rules for the right triggers.

  If you prioritize simplicity during the initial setup, use an autoscaling option that requires only setting limits, without needing to define rules.

  Have sufficient resources readily available to help ensure optimal performance. Allocate resources appropriately to maintain performance targets, such as response time or throughput. Consider overallocation of resources when necessary.

  When you define autoscale rules, account for the time that it takes for your application to initialize. Consider this overhead when you make all scaling decisions.

- ✓ **Use caching:** Retrieving information from a resource that doesn't change frequently and is expensive to access affects performance. Complex queries, including joins and multiple lookups, contribute to runtime. Perform caching to minimize the processing time and latency. Cache query results to avoid repeated round trips to the database or back end and reduce processing time for subsequent requests.

For more information about using local and distributed cache in the workload, see Caching.

- ✓ **Review the performance antipatterns:** To make sure that the web application performs and scales in accordance with your business requirements, avoid the typical antipatterns. The following table describes some antipatterns that App Service corrects.

Expand table

Antipattern	Description
Busy Front End	Resource-intensive tasks can increase the response times for user requests and cause high latency. Move processes that consume significant resources to a separate back end. Use a message broker to queue resource-intensive tasks that the back end picks up to asynchronously process.
No Caching	Serve requests from an intermediate cache in front of the back-end database to reduce latency.
Noisy Neighbor	Multitenant systems share resources between tenants. The activity of one tenant can have a negative effect on another tenant's use of the system. App Service Environment provides a fully isolated and dedicated environment to run App Service apps.

Recommendations

Expand table

Recommendation	Benefit
(App Service) Enable the *Always On* setting when applications share a single App Service plan. App Service apps automatically unload when idle to save resources. The next request triggers a cold start, which can cause request time-outs.	The application is never unloaded with Always On enabled.
(Web Apps) Consider using HTTP/2 for applications to improve protocol efficiency.	Choose HTTP/2 over HTTP/1.1 because HTTP/2 fully multiplexes connections, reuses connections to reduce overhead, and compresses headers to minimize data transfer.

Tradeoffs

You might have to make design tradeoffs if you use the approaches in the pillar checklists. Here are some examples of advantages and drawbacks.

Density and isolation

- **Higher density:** Colocate multiple apps within the same App Service plan to minimize resources. All apps share resources like CPU and memory, which can save money and reduce operational complexity. This approach also optimizes resource usage. Apps can use idle resources from another app if load patterns change over time.

  Also consider the disadvantages, like resource contention. For example, spikes in usage or instability of an app can affect the performance of other apps. Incidents in one app can also permeate other apps within the shared environment, which can affect security. For critical applications that require high availability and performance, isolated environments like **App Service Environment v3** provide dedicated resources but at a higher cost. Consider using shared environments for noncritical workloads and isolated environments for mission-critical applications.

- **Higher isolation:** Isolation helps prevent interference. This strategy applies to security, performance, and even segregation of development, testing, and production environments.

  App Service Environment provides better control over security and data protection because each app can have its own security settings. Your environment can contain breaches because isolation limits the blast radius. Resource contention is minimized from a performance perspective. Isolation allows for independent scaling based on specific demand and individual capacity planning.

  As a disadvantage, this approach is more expensive and requires operational rigor.

Reliable scaling strategy

A well-defined scaling strategy helps ensure that your application can handle various loads without compromising performance. However, there are tradeoffs on cost. Scaling operations take time. When new resources are allocated, the application must be properly initialized before it can effectively process requests. You can overprovision resources (or prewarm instances) to provide a safety net. Without that extra capacity, during the initialization phase, there might be a delay in serving requests, which affects

user experience. Autoscaling operations might trigger early enough to enable proper resource initialization by the time customers use the resources.

As a disadvantage, overprovisioned resources cost more. You're charged per second for every instance, including prewarmed instances. Higher tiers include prewarmed instances. Determine whether capabilities with more expensive tiers are worth the investment.



Building redundancy

Redundancy provides resiliency but also incurs costs. Service-level objectives (SLOs) for your workload determine acceptable performance thresholds. It becomes wasteful if redundancy exceeds SLO requirements. Evaluate whether excess redundancy improves SLOs or adds unnecessary complexity.

Also consider the disadvantages. For example, multiple-region redundancy provides high availability but adds complexity and cost because of data synchronization, failover mechanisms, and interregional communication. Determine whether zone redundancy can meet your SLOs.

Azure policies

Azure provides an extensive set of built-in policies related to App Service and its dependencies. A set of Azure policies can audit some of the preceding recommendations. For example, you can check whether:

- Proper network controls are in place. For example, you can incorporate network segmentation by placing App Service in Azure Virtual Network through virtual network injection to have greater control over network configuration. The application doesn't have public endpoints and connects to Azure services through private endpoints.
- Identity controls are in place. For example, the application uses managed identities to authenticate itself against other resources. App Service built-in authentication (or Easy Auth) verifies incoming requests.
- Features such as remote debugging and basic authentication are disabled to reduce the attack surface.

For comprehensive governance, review the Azure Policy built-in definitions and other policies that might affect the security of the compute layer.

Azure Advisor recommendations

Azure Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments.

For more information, see Azure Advisor.

Next steps

Consider the following articles as resources that demonstrate the recommendations highlighted in this article.

- Use the following reference architectures as examples of how to apply these recommendations to a workload.
  - If you haven't deployed a web app, see Basic web application.
  - For a foundational architecture as your starting point for a production-grade deployment, see Baseline highly available zone-redundant web application.
- Use the following product documentation to build your implementation expertise:
  - App Service
  - App Service plan

Feedback

Was this page helpful? Yes No

Architecture best practices for Azure Blob Storage

Article • 04/22/2024

Azure Blob Storage is a Microsoft object storage solution for the cloud. Blob Storage is optimized to store massive amounts of unstructured data. Unstructured data is data that doesn't adhere to a specific data model or definition, such as text or binary data.

This article assumes that as an architect, you reviewed your storage options and chose Blob Storage as the storage service on which to run your workloads. The guidance in this article provides architectural recommendations that are mapped to the principles of the Azure Well-Architected Framework pillars.

> **Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern along with design strategies.
>
> Also included are *recommendations* on the technology capabilities that can help implement those strategies. The recommendations don't represent an exhaustive list of all configurations available for Blob Storage and its dependencies. Instead, they list the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept or optimize your existing environments.

Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover fast from failures**.

The Reliability design principles provide a high-level design strategy applied for individual components, system flows, and the system as a whole.

Design checklist

Start your design strategy based on the design review checklist for Reliability.

- ✓ **Use failure mode analysis**: Minimize points of failure by considering internal dependencies such as the availability of virtual networks, Azure Key Vault, or Azure Content Delivery Network or Azure Front Door endpoints. Failures can occur if credentials required by workloads to access Blob Storage go missing from Key Vault, or if workloads use an endpoint based on a content delivery network that's removed. In these cases, workloads might need to use an alternative endpoint to connect. For general information about failure mode analysis, see Recommendations for performing failure mode analysis.

- ✓ **Define reliability and recovery targets**: Review the Azure service-level agreements (SLAs). Derive the service-level objective (SLO) for the storage account. For example, the SLO might be affected by the redundancy configuration that you chose. Consider the effect of a regional outage, the potential for data loss, and the time required to restore access after an outage. Also consider the availability of any internal dependencies that you identified as part of your failure mode analysis.

- ✓ **Configure data redundancy**: For maximum durability, choose a configuration that copies data across availability zones or global regions. For maximum availability, choose a configuration that allows clients to read data from the secondary region during an outage of the primary region.

- ✓ **Design applications**: Design applications to seamlessly shift to reading data from the secondary region if the primary region becomes unavailable for any reason. This only applies to geo-redundant storage (GRS) and geo-zone-redundant storage (GZRS) configurations. Designing applications to handle outages reduces downtime for end users.

- ✓ **Explore features to help you meet your recovery targets**: Make blobs restorable so that they can be recovered if they're corrupted, edited, or deleted by mistake.

- ✓ **Create a recovery plan**: Consider data protection features, backup and restore operations, or failover procedures. Prepare for potential data loss and data inconsistencies and the time and cost of failing over. For more information, see Recommendations for designing a disaster recovery strategy.

- ✓ **Monitor potential availability problems**: Subscribe to the Azure Service Health dashboard to monitor potential availability problems. Use storage metrics in Azure Monitor and diagnostic logs to investigate alerts.

Recommendations

Expand table

Recommendation	Benefit
Configure your account for redundancy. For maximum availability and durability, configure your account by using zone-redundant storage (ZRS) or GZRS.	Redundancy protects your data against unexpected failures. The ZRS and GZRS configuration options replicate across different availability zones and enable applications to continue reading data during an outage. For more information, see Durability and availability by outage scenario and Durability and availability parameters.
Before initiating a failover or failback, evaluate the potential for data loss by checking the value of the last synchronization time property. This recommendation applies only to GRS and GZRS configurations.	This property helps you estimate how much data you might lose by initiating an account failover. All data and metadata written before the last synchronization time is available on the secondary region, but data and metadata written after the last synchronization time might be lost because it's not written to the secondary region.
As a part of your backup and recovery strategy, enable the container soft delete, blob soft delete, versioning, and point-in-time restore options.	The soft delete option enables a storage account to recover deleted containers and blobs. The versioning option automatically tracks changes made to blobs. This option lets you restore a blob to a previous state. The point-in-time restore option protects against accidental blob deletion or corruption and lets you restore block blob data to an earlier state. For more information, see Data protection overview.
Configure vaulted backup for Azure Blob as a part of your backup strategy.	Vaulted backup enables you to protect the block blob data from ransomware, other malicious attacks, or source data loss. The data is copied and stored in the Backup vault (an offsite copy of data) that can be retained for up to 10 years. If any data loss happens on the source account, you can trigger a restore to an alternate account and get access to your data. Learn more about the supportability for vaulted backup using Azure Backup.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload.

The Security design principles provide a high-level design strategy for achieving those goals by applying approaches to the technical design of your Blob Storage

configuration.

Design checklist

Start your design strategy based on the design review checklist for Security. Identify vulnerabilities and controls to improve the security posture. Extend the strategy to include more approaches as needed.

- ✓ **Review the security baseline for Azure Storage**: To get started, first review the security baseline for Storage.
- ✓ **Use network controls to restrict ingress and egress traffic**: Disable all public traffic to the storage account. Use account network controls to grant the minimal level of access required by users and applications. For more information, see How to approach network security for your storage account.
- ✓ **Reduce the attack surface**: Preventing anonymous access, account key access, or access over non-secure (HTTP) connections can reduce the attack surface. Require clients to send and receive data by using the latest version of the Transport Layer Security (TLS) protocol.
- ✓ **Authorize access without using passwords or keys**: Microsoft Entra ID provides superior security and ease of use compared to shared keys and shared access signatures. Grant security principals only those permissions that are necessary for them to do their tasks.
- ✓ **Protect sensitive information**: Protect sensitive information such as account keys and shared access signature tokens. While these forms of authorization are generally not recommended, you should make sure to rotate, expire, and store them securely.
- ✓ **Enable the secure transfer required option**: Enabling this setting for all your storage accounts ensures that all requests made against the storage account must take place over secure connections. Any requests made over HTTP fail.
- ✓ **Protect critical objects**: Apply immutability policies to protect critical objects. Policies protect blobs that are stored for legal, compliance, or other business purposes from being modified or deleted. Configure holds for set time periods or until restrictions are lifted by an administrator.
- ✓ **Detect threats**: Enable Microsoft Defender for Storage to detect threats. Security alerts are triggered when anomalies in activity occur. The alerts notify subscription

administrators via email with details of suspicious activity and recommendations on how to investigate and remediate threats.

Recommendations

Expand table

Recommendation	Benefit
Disable anonymous read access to containers and blob.	When anonymous access is allowed for a storage account, a user that has the appropriate permissions can modify a container's anonymous access setting to enable anonymous access to the data in that container.
Apply an Azure Resource Manager lock on the storage account.	Locking an account prevents it from being deleted and causing data loss.
Disable traffic to the public endpoints of your storage account. Create private endpoints for clients that run in Azure. Enable the public endpoint only if clients and services external to Azure require direct access to your storage account. Enable firewall rules that limit access to specific virtual networks.	Start with zero access and then incrementally authorize the lowest levels of access required for clients and services to minimize the risk of creating unnecessary openings for attackers.
Authorize access by using Azure role-based access control (RBAC).	With RBAC, there are no passwords or keys that can be compromised. The security principal (user, group, managed identity, or service principal) is authenticated by Microsoft Entra ID to return an OAuth 2.0 token. The token is used to authorize a request against the Blob Storage service.
Disallow shared key authorization. This disables not only account key access but also service and account shared access signature tokens because they're based on account keys.	Only secured requests that are authorized with Microsoft Entra ID are permitted.
We recommend that you don't use an account key. If you must use account keys, then store them in Key Vault, and make sure that you regenerate them periodically.	Key Vault lets you retrieve keys at runtime, instead of saving them by using your application. Key Vault also makes it easy to rotate your keys without interruption to your applications. Rotating the account keys periodically reduces the risk of exposing your data to malicious attacks.

Recommendation	Benefit
We recommend that you don't use shared access signature tokens. Evaluate whether you need shared access signature tokens to secure access to Blob Storage resources. If you must create one, then review this list of shared access signature best practices before you create and distribute it.	Best practices can help you prevent a shared access signature token from being leaked and quickly recover if a leak does occur.
Configure your storage account so clients can send and receive data by using the minimum version of TLS 1.2.	TLS 1.2 is more secure and faster than TLS 1.0 and 1.1, which don't support modern cryptographic algorithms and cipher suites.
Consider using your own encryption key to protect the data in your storage account. For more information, see Customer-managed keys for Azure Storage encryption.	Customer-managed keys provide greater flexibility and control. For example, you can store encryption keys in Key Vault and automatically rotate them.

Cost Optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget and business requirements.

The Cost Optimization design principles provide a high-level design strategy for achieving those goals and making tradeoffs as necessary in the technical design related to Blob Storage and its environment.

Design checklist

Start your design strategy based on the design review checklist for Cost Optimization for investments. Fine-tune the design so that the workload is aligned with the budget that's allocated for the workload. Your design should use the right Azure capabilities, monitor investments, and find opportunities to optimize over time.

- ✓ **Identify the meters that are used to calculate your bill**: Meters are used to track the amount of data stored in the account (data capacity) and the number and type of operations that are performed to write and read data. There are also meters associated with the use of optional features such as blob index tags, blob inventory, change feed support, encryption scopes, and SSH File Transfer Protocol (SFTP) support. For more information, see How you're charged for Blob Storage.
- ✓ **Understand the price of each meter**: Make sure to use the appropriate pricing page and apply the appropriate settings in that page. For more information, see

Finding the unit price for each meter. Consider the number of operations associated with each price. For example, the price associated with write and read operations applies to 10,000 operations. To determine the price of an individual operation, divide the listed price by 10,000.

- ✓ **Estimate the cost of capacity and operations**: You can model the costs associated with data storage, ingress, and egress by using the Azure pricing calculator. Use fields to compare the cost associated with various regions, account types, namespace types, and redundancy configurations. For certain scenarios, you can use sample calculations and worksheets available in Microsoft documentation. For example, you can estimate the cost of archiving data or estimate the cost of using the AzCopy command to transfer blobs.

- ✓ **Choose a billing model for capacity**: Evaluate whether using a commitment-based model is more cost-efficient than using a consumption-based model. If you're unsure about how much capacity you need, you can start with a consumption-based model, monitor the capacity metrics, and then evaluate later.

- ✓ **Choose an account type, a redundancy level, and a default access tier**: You must select a value for each of these settings when you create a storage account. All the values affect transaction charges and capacity charges. All these settings except for the account type can be changed after the account is created.

- ✓ **Choose the most cost-effective default access tier**: Unless a tier is specified with each blob upload, blobs infer their access tier from the default access tier setting. A change to the default access tier setting of a storage account applies to all blobs in the account for which an access tier hasn't been explicitly set. This cost could be significant if you've collected a large number of blobs. For more information about how a tier change affects each existing blob, see Changing a blob's access tier.

- ✓ **Upload data directly to the most cost-efficient access tier**: For example, if the default access tier setting of your account is hot, but you're uploading files for archiving purposes, specify a cooler tier as the archive or a cold tier as part of your upload operation. After uploading blobs, use lifecycle management policies to move blobs to the most cost-efficient tiers based on usage metrics such as the last accessed time. Choosing the most optimal tier up front can reduce costs. If you change the tier of a block blob that you already uploaded, then you pay the cost of writing to the initial tier when you first upload the blob, and then pay the cost of writing to the desired tier.

- ✓ **Have a plan for managing the data lifecycle**: Optimize transaction and capacity costs by taking advantage of access tiers and lifecycle management. Data used less

often should be placed in cooler access tiers while data that's accessed often should be placed in warmer access tiers.

- ✓ **Decide which features you need**: Some features such as versioning and blob soft delete incur additional transaction and capacity costs as well as other charges. Make sure to review the pricing and billing sections in articles that describe those capabilities when you choose which capabilities to add to your account.

  For example, if you enable the blob inventory feature, you're billed for the number of objects scanned. If you use blob index tags, you're billed for the number of index tags. If you enable SFTP support, you're billed an hourly charge, even if there are no SFTP transfers. If you decide against using a feature, confirm that the feature is disabled because some features are automatically enabled when you create the account.

- ✓ **Create guardrails**: Create budgets based on subscriptions and resource groups. Use governance policies to restrict resource types, configurations, and locations. Additionally, use RBAC to block actions that can lead to overspending.

- ✓ **Monitor costs**: Ensure costs stay within budgets, compare costs against forecasts, and see where overspending occurs. You can use the cost analysis pane in the Azure portal to monitor costs. You also can export cost data to a storage account and analyze that data by using Excel or Power BI.

- ✓ **Monitor usage**: Continuously monitor usage patterns and detect unused or underutilized accounts and containers. Use Storage insights to identity accounts with no or low use. Enable blob inventory reports, and use tools such as Azure Databricks or Azure Synapse Analytics and Power BI to analyze cost data. Watch out for unexpected increases in capacity, which might indicate that you're collecting numerous log files, blob versions, or soft-deleted blobs. Develop a strategy for expiring or transitioning objects to more cost-effective access tiers.Have a plan for expiring objects or moving objects to more affordable access tiers.

Recommendations

Expand table

Recommendation	Benefit
Pack small files into larger files before moving them to cooler tiers. You can use file formats such as TAR or ZIP.	Cooler tiers have higher data transfer costs. By having fewer large files, you can reduce the number of operations required to transfer data.

Recommendation	Benefit
Use standard-priority rehydration when rehydrating blobs from archive storage. Use high-priority rehydration only for emergency data restoration situations. For more information, see Rehydrate an archived blob to an online tier	High-priority rehydration from the archive tier can lead to higher-than-normal bills.
Reduce the cost of using resource logs by choosing the appropriate log storage location and by managing log-retention periods. If you only plan to query logs occasionally (for example, querying logs for compliance auditing), consider sending resource logs to a storage account instead of sending them to an Azure Monitor Logs workspace. You can use a serverless query solution such as Azure Synapse Analytics to analyze logs. For more information, see Optimize cost for infrequent queries. Use lifecycle management policies to delete or archive logs.	Storing resource logs in a storage account for later analysis can be a cheaper option. Using lifecycle management policies to manage log retention in a storage account prevents large numbers of logs files building up over time, which can lead to unnecessary capacity charges.
If you enable versioning, use a lifecycle management policy to automatically delete old blob versions.	Every write operation to a blob creates a new version. This increases capacity costs. You can keep costs in check by removing versions that you no longer need.
If you enable versioning, then place blobs that are frequently overwritten into an account that doesn't have versioning enabled.	Every time a blob is overwritten, a new version is added which leads to increased storage capacity charges. To reduce capacity charges, store frequently overwritten data in a separate storage account with versioning disabled.
If you enable soft delete, then place blobs that are frequently overwritten into an account that doesn't have soft delete enabled. Set retention periods. Consider starting with a short retention period to better understand how the feature affects your bill. The minimum recommended retention period is seven days.	Every time a blob is overwritten, a new snapshot is created. The cause of increased capacity charges might be difficult to access because the creation of these snapshots doesn't appear in logs. To reduce capacity charges, store frequently overwritten data in a separate storage account with soft delete disabled. A retention period keeps soft-deleted blobs from piling up and adding to the cost of capacity.
Enable SFTP support only when it's used to transfer data.	Enabling the SFTP endpoint incurs an hourly cost. By thoughtfully disabling SFTP support, and then enabling it as

Recommendation	Benefit
	needed, you can avoid passive charges from accruing in your account.
Disable any encryption scopes that aren't needed to avoid unnecessary charges.	Encryptions scopes incur a per month charge.

Operational Excellence

Operational Excellence primarily focuses on procedures for **development practices, observability, and release management**.

The Operational Excellence design principles provide a high-level design strategy for achieving those goals for the operational requirements of the workload.

Design checklist

Start your design strategy based on the design review checklist for Operational Excellence for defining processes for observability, testing, and deployment related to your Blob Storage configuration.

- ✓ **Create maintenance and emergency recovery plans**: Consider data protection features, backup and restore operations, and failover procedures. Prepare for potential data loss and data inconsistencies and the time and cost of failing over.
- ✓ **Monitor the health of your storage account**: Create Storage insights dashboards to monitor availability, performance, and resilience metrics. Set up alerts to identify and address problems in your system before your customers notice them. Use diagnostic settings to route resource logs to an Azure Monitor Logs workspace. Then you can query logs to investigate alerts more deeply.
- ✓ **Enable blob inventory reports**: Enable blob inventory reports to review the retention, legal hold, or encryption status of your storage account contents. You can also use blob inventory reports to understand the total data size, age, tier distribution, or other attributes of your data. Use tools such as Azure Databricks or Azure Synapse Analytics and Power BI to better visualize inventory data and to create reports for stakeholders.
- ✓ **Set up policies that delete blobs or move them to cost-efficient access tiers**: Create a lifecycle management policy with an initial set of conditions. Policy runs automatically delete or set the access tier of blobs based on the conditions you

define. Periodically analyze container use by using Monitor metrics and blob inventory reports so that you can refine conditions to optimize cost efficiency.

Recommendations

Expand table

Recommendation	Benefit
Use infrastructure as code (IaC) to define the details of your storage accounts in Azure Resource Manager templates (ARM templates), Bicep, or Terraform.	You can use your existing DevOps processes to deploy new storage accounts, and use Azure Policy to enforce their configuration.
Use Storage insights to track the health and performance of your storage accounts. Storage insights provides a unified view of the failures, performance, availability, and capacity for all your storage accounts.	You can track the health and operation of each of your accounts. Easily create dashboards and reports that stakeholders can use to track the health of your storage accounts.

Performance Efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance.

The Performance Efficiency design principles provide a high-level design strategy for achieving those capacity goals against the expected usage.

Design checklist

Start your design strategy based on the design review checklist for Performance Efficiency. Define a baseline that's based on key performance indicators for your Blob Storage configuration.

- ✓ **Plan for scale**: Understand the scale targets for storage accounts.
- ✓ **Choose the optimal storage account type**: If your workload requires high transaction rates, smaller objects, and a consistently low transaction latency, then consider using premium block blob storage accounts. A standard general-purpose v2 account is most appropriate in most cases.
- ✓ **Reduce travel distance between the client and server**: Place data in regions nearest to connecting clients (ideally in the same region). Optimize for clients in

regions far away by using object replication or a content delivery network. Default network configurations provide the best performance. Modify network settings only to improve security. In general, network settings don't decrease travel distance and don't improve performance.

- ✓ **Choose an efficient naming scheme**: Decrease the latency of listing, list, query, and read operations by using hash tag prefixes nearest the beginning of the blob partition key (account, container, virtual directory, or blob name). This scheme benefits mostly accounts that have a flat namespace.

- ✓ **Optimize the performance of data clients**: Choose a data transfer tool that's most appropriate for the data size, transfer frequency, and bandwidth of your workloads. Some tools such as AzCopy are optimized for performance and require little intervention. Consider the factors that influence latency, and fine-tune performance by reviewing the performance optimization guidance that's published with each tool.

- ✓ **Optimize the performance of custom code**: Consider using Storage SDKs instead of creating your own wrappers for blob REST operations. Azure SDKs are optimized for performance and provide mechanisms to fine-tune performance. Before creating an application, review the performance and scalability checklist for Blob Storage. Consider using query acceleration to filter out unwanted data during the storage request and keep clients from needlessly transferring data across the network.

- ✓ **Collect performance data**: Monitor your storage account to identify performance bottlenecks that occur from throttling. For more information, see Monitoring your storage service with Monitor Storage insights. Use both metrics and logs. Metrics provide numbers such as throttling errors. Logs describe activity. If you see throttling metrics, you can use logs to identity which clients are receiving throttling errors. For more information, see Auditing data plane operations.

Recommendations

Expand table

Recommendation	Benefit
Provision storage accounts in the same region where dependent resources are placed. For applications that aren't hosted on Azure, such as mobile device apps or on-premises enterprise services, locate the storage account in a region	Reducing the physical distance between the storage account and VMs, services, and on-premises clients can improve performance and reduce network latency. Reducing the physical distance also

Recommendation	Benefit
nearer to those clients. For more information, see Azure geographies. If clients from a different region don't require the same data, then create a separate account in each region. If clients from a different region require only some data, consider using an object-replication policy to asynchronously copy relevant objects to a storage account in the other region.	reduces cost for applications hosted in Azure because bandwidth usage within a single region is free.
For broad consumption by web clients (streaming video, audio, or static website content), consider using a content delivery network through Azure Front Door.	Content is delivered to clients faster because it uses the Microsoft global edge network with hundreds of global and local points of presence around the world.
Add a hash character sequence (such as three digits) as early as possible in the partition key of a blob. The partition key is the account name, container name, virtual directory name, and blob name. If you plan to use timestamps in names, then consider adding a seconds value to the beginning of that stamp. For more information, see Partitioning.	Using a hash code or seconds value nearest the beginning of a partition key reduces the time required to list query and read blobs.
When uploading blobs or blocks, use a blob or block size that's greater than 256 KiB.	Blob or block sizes above 256 KiB takes advantage of performance enhancements in the platform made specifically for larger blobs and block sizes.

Azure policies

Azure provides an extensive set of built-in policies related to Blob Storage and its dependencies. Some of the preceding recommendations can be audited through Azure policies. For example, you can check if:

- Anonymous public read access to containers and blobs isn't enabled.
- Diagnostic settings for Blob Storage are set to stream resource logs to an Azure Monitor Logs workspace.
- Only requests from secure connections (HTTPS) are accepted.
- A shared access signature expiration policy is enabled.
- Cross-tenant object replication is disabled.
- Shared key authorization is disabled.
- Network firewall rules are applied to the account.

For comprehensive governance, review the Azure Policy built-in definitions for Storage and other policies that might affect the security of the compute layer.

Azure Advisor recommendations

Azure Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments. Here are some recommendations that can help you improve the reliability, security, cost effectiveness, performance, and operational excellence of Blob Storage.

- Reliability
- Security
- Cost Optimization
- Performance
- Operational Excellence

Next step

For more information about Blob Storage, see Blob Storage documentation.

Feedback

Was this page helpful? Yes No

Azure Cache for Redis and reliability

Article • 11/14/2023

Azure Cache for Redis provides an in-memory data store based on the Redis (Remote Dictionary Server) software. It's a secure data cache and messaging broker that provides high throughput and low-latency access to data for applications.

Key concepts and best practices that support reliability include:

- High availability
- Failover and patching
- Connection resilience

The following sections include design considerations, a configuration checklist, and recommended configuration options specific to Azure Cache for Redis.

Design considerations

The Azure Cache for Redis Service Level Agreements (SLA) covers only Standard and Premium tier caches. Basic tier isn't covered.

Redis is an in-memory cache for key value pairs and has High Availability (HA), by default, except for Basic tier. There are three tiers for Azure Cache for Redis:

- *Basic*: *Not recommended for production workloads*. Basic tier is ideal for:
  - Single node
  - Multiple sizes
  - Development
  - Test
  - Non-critical workloads

- *Standard*: A replicated cache in a two-node primary and secondary configuration managed by Microsoft, with a high availability SLA.

- *Premium*: Includes all standard-tier features and includes the following other features:
  - Faster hardware and performance compared to Basic or Standard tier.
  - Larger cache size, up to `120GB`.
  - Data persistence, which includes Redis Database File (RDB) and Append Only File (AOF).
  - VNET support.
  - Clustering

- Geo-Replication: A secondary cache is in another region and replicates data from the primary for disaster recovery. To failover to the secondary, the caches need to be unlinked manually and then the secondary is available for writes. The application writing to Redis needs to be updated with the secondary's cache connection string.
- Availability Zones: Deploy the cache and replicas across availability zones.

  > **Note**
  >
  > By default, each deployment will have one replica per shard. Persistence, clustering, and geo-replication are all disabled at this time with deployments that have more than one replica. Your nodes will be distributed evenly across all zones. You should have a replica count `>=` number of zones.

- Import and export.

Microsoft guarantees at least `99.9%` of the time that customers will have connectivity between the Cache Endpoints and Microsoft's Internet gateway.

Checklist

Have you configured Azure Cache for Redis with resiliency in mind?

- ✓ Schedule updates.
- ✓ Monitor the cache and set alerts.
- ✓ Deploy the cache within a VNET.
- ✓ Evaluate a partitioning strategy within Redis cache.
- ✓ Configure **Data Persistence** to save a copy of the cache to Azure Storage or use Geo-Replication, depending on the business requirement.
- ✓ Implement retry policies in the context of your Azure Redis Cache.
- ✓ Use one static or singleton implementation of the connection multiplexer to Redis and follow the best practices guide.
- ✓ Review How to administer Azure Cache for Redis.

Configuration recommendations

Explore the following table of recommendations to optimize your Azure Cache for Redis configuration for service reliability:

Recommendation	Description
Schedule updates.	Schedule the days and times that Redis Server updates will be applied to the cache, which doesn't include Azure updates, or updates to the VM operating system.
Monitor the cache and set alerts.	Set alerts for exceptions, high CPU, high memory usage, server load, and evicted keys for insights about when to scale the cache. If the cache needs to be scaled, understanding when to scale is important because it will increase CPU during the scaling event to migrate data.
Deploy the cache within a VNET.	Gives the customer more control over the traffic that can connect to the cache. Make sure that the subnet has sufficient address space available to deploy the cache nodes and shards (cluster).
Evaluate a partitioning strategy within Redis cache.	Partitioning a Redis data store involves splitting the data across instances of the Redis server. Each instance makes up a single partition. Azure Redis Cache abstracts the Redis services behind a facade and doesn't expose them directly. The simplest way to implement partitioning is to create multiple Azure Redis Cache instances and spread the data across them. You can associate each data item with an identifier (a partition key) that specifies which cache stores the data item. The client application logic can then use this identifier to route requests to the appropriate partition. This scheme is simple, but if the partitioning scheme changes (for example, if extra Azure Redis Cache instances are created), client applications may need to be reconfigured.
Configure **Data Persistence** to save a copy of the cache to Azure Storage or use Geo-Replication, depending on the business requirement.	*Data Persistence*: if the master and replica reboot, the data will be loaded automatically from the storage account. *Geo-Replication*: The secondary cache needs to be unlinked from the primary. The secondary will now become the primary and can receive *writes*.
Implement retry policies in the context of your Azure Redis Cache.	Most Azure services and client SDKs include a retry mechanism. These mechanisms differ because each service has different characteristics and requirements. Each retry mechanism is tuned to a specific service.
Review How to administer Azure Cache for Redis.	Understand how data loss can occur with cache reboots and how to test the application for resiliency.

Source artifacts

To identify Redis instances that aren't on the Premium tier, use the following query:

```
SQL
```

```
Resources
| where type == 'microsoft.cache/redis'
| where properties.sku.name != 'Premium'
```

Next step

Azure Cache for Redis and operational excellence

Azure Cache for Redis and operational excellence

Article • 11/14/2023

Azure Cache for Redis provides an in-memory data store based on the Redis (Remote Dictionary Server) software. It's a secure data cache and messaging broker that provides high throughput and low-latency access to data for applications.

Best practices that support operational excellence include:

- Server load management
- Memory management
- Performance testing

The following sections include design considerations, a configuration checklist, and recommended configuration options specific to Azure Cache for Redis.

Design considerations

The Azure Cache for Redis Service Level Agreements (SLA) covers only Standard and Premium tier caches. Basic tier isn't covered.

Redis is an in-memory cache for key value pairs and has High Availability (HA), by default, except for Basic tier. There are three tiers for Azure Cache for Redis:

- *Basic*: *Not recommended for production workloads*. Basic tier is ideal for:
  - Single node
  - Multiple sizes
  - Development
  - Test
  - Non-critical workloads

- *Standard*: A replicated cache in a two-node primary and secondary configuration managed by Microsoft, with a high availability SLA.

- *Premium*: Includes all standard-tier features and includes the following other features:
  - Faster hardware and performance compared to Basic or Standard tier.
  - Larger cache size, up to `120GB`.
  - Data persistence, which includes Redis Database File (RDB) and Append Only File (AOF).

- VNET support.
- Clustering
- Geo-Replication: A secondary cache is in another region and replicates data from the primary for disaster recovery. To failover to the secondary, the caches need to be unlinked manually and then the secondary is available for writes. The application writing to Redis will need to be updated with the secondary's cache connection string.
- Availability Zones: Deploy the cache and replicas across availability zones.

  > **Note**
  >
  > By default, each deployment will have one replica per shard. Persistence, clustering, and geo-replication are all disabled at this time with deployments that have more than one replica. Your nodes will be distributed evenly across all zones. You should have a replica count `>=` number of zones.

- Import and export.

Microsoft guarantees at least `99.9%` of the time that customers will have connectivity between the Cache Endpoints and Microsoft's Internet gateway.

Checklist

Have you configured Azure Cache for Redis with operational excellence in mind?

- ✓ Schedule updates.
- ✓ Monitor the cache and set alerts.
- ✓ Deploy the cache within a VNET.
- ✓ Use the correct caching type (local, in role, managed, redis) within your solution.
- ✓ Configure **Data Persistence** to save a copy of the cache to Azure Storage or use Geo-Replication, depending on the business requirement.
- ✓ Use one static or singleton implementation of the connection multiplexer to Redis and follow the best practices guide.
- ✓ Review How to administer Azure Cache for Redis.

Configuration recommendations

Explore the following table of recommendations to optimize your Azure Cache for Redis configuration for operational excellence:

Recommendation	Description
Schedule updates.	Schedule the days and times that Redis Server updates will be applied to the cache, which doesn't include Azure updates, or updates to the VM operating system.
Monitor the cache and set alerts.	Set alerts for exceptions, high CPU, high memory usage, server load, and evicted keys for insights about when to scale the cache. If the cache needs to be scaled, understanding when to scale is important because it will increase CPU during the scaling event to migrate data.
Deploy the cache within a VNET.	Gives the customer more control over the traffic that can connect to the cache. Make sure that the subnet has sufficient address space available to deploy the cache nodes and shards (cluster).
Use the correct caching type (local, in role, managed, redis) within your solution.	Distributed applications typically implement either or both of the following strategies when caching data: - Using a private cache, where data is held locally on the machine that's running an instance of an application or service. - Using a shared cache, serving as a common source that can be accessed by multiple processes and machines. In both cases, caching can be performed client-side and server-side. Client-side caching is done by the process that provides the user interface for a system, such as a web browser or desktop application. Server-side caching is done by the process that provides the business services that are running remotely.
Configure **Data Persistence** to save a copy of the cache to Azure Storage or use Geo-Replication, depending on the business requirement.	*Data Persistence*: If the master and replica reboot, the data will be loaded automatically from the storage account. *Geo-Replication*: The secondary cache needs to be unlinked from the primary. The secondary will now become the primary and can receive *writes*.
Review How to administer Azure Cache for Redis.	Understand how data loss can occur with cache reboots and how to test the application for resiliency.

Source artifacts

To identify Redis instances that aren't on the Premium tier, use the following query:

SQL

```
Resources
| where type == 'microsoft.cache/redis'
```

```
| where properties.sku.name != 'Premium'
```

Next step

Azure Databricks and security

Architecture best practices for Azure Container Apps

Article • 04/07/2025

Azure Container Apps is a fully managed serverless container service that runs microservices and containerized applications on Azure. It provides built-in autoscaling, including scale to zero, and supports multiple programming languages and frameworks. Container Apps integrates with Azure Kubernetes Service (AKS) for advanced networking and monitoring. It also provides seamless deployment and management of containerized applications without the need to manage underlying infrastructure. It supports both HTTP-based and event-driven applications, so it's ideal for building modern, scalable, and resilient cloud-native applications.

This article assumes that as an architect, you've reviewed the compute decision tree and chose Container Apps as the compute platform for your workload. The guidance in this article provides architectural recommendations that are mapped to the principles of the Well-Architected Framework pillars.

> **Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern along with design strategies localized to the technology scope.
>
> Also included are recommendations for the technology capabilities that can help materialize those strategies. The recommendations don't represent an exhaustive list of all configurations that are available for Container Apps and its dependencies. Instead, they list the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept or to optimize your existing environments.
>
> Foundational architecture that demonstrates the key recommendations: Microservices with Azure Container Apps.

Technology scope

This review focuses on the interrelated decisions for the following Azure resources:

- Container Apps

Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover fast from failures**.

Reliability design principles provide a high-level design strategy applied for individual components, system flows, and the system as a whole.

Design checklist

Start your design strategy based on the design review checklist for Reliability. Determine its relevance to your business requirements while keeping in mind the performance and reliability of your applications. Extend the strategy to include more approaches as needed.

- ✓ **Choose appropriate SKU configurations.** Select an environment SKU that aligns with the resource and performance requirements of your container apps.
- ✓ **Build redundancy to improve resiliency.** For ingress-exposed (HTTP or Transmission Control Protocol (TCP)) apps, use at least three replicas to help ensure availability. To minimize cold starts, configure a minimum number of always-ready replicas.

  Use availability zones as part of your resiliency strategy to increase availability when you deploy to a single region. Many Azure regions provide availability zones. The zones are positioned close enough to ensure low-latency connections among them, but far enough apart to minimize the risk of local outages affecting multiple zones.

  For critical workloads, deploy Container Apps environments across multiple regions and use Azure Front Door or Azure Traffic Manager for traffic management. These services help ensure high availability and business continuity. If a regional outage occurs, you can automatically redirect traffic to a secondary region to minimize downtime and data loss.
- ✓ **Implement horizontal autoscaling.** Configure autoscaling by using scale rules that are based on HTTP requests, TCP connections, or custom metrics, such as CPU and memory thresholds. You can define custom metrics in Azure Service Bus, Azure Event Hubs, Apache Kafka, and Azure Cache for Redis. Use autoscaling to dynamically manage loads and maintain high availability during peak usage times.

  Ensure that your container apps can still serve requests within your service-level objective (SLO) under high load.
- ✓ **Monitor reliability and overall health indicators of the container app.** Collect logs and metrics to monitor health, identify performance and reliability trends, and troubleshoot problems. For more information about designing the reliability and health monitoring solution for your workload, see Health modeling for workloads.

Implement monitoring tooling and alerts. Activate monitoring tools, like Azure Monitor and OpenTelemetry. Set up alerts to promptly detect and respond to events that affect reliability.

Configure health probes. Set up startup, readiness, and liveness probes for all services to monitor and maintain application health.

- ✓ **Configure self-healing mechanisms to automatically restart unhealthy container instances.** Automatic restarts increase the reliability and availability of your applications. They help ensure quick recovery from failures without manual intervention. Use health probes to detect failing containers and configure resiliency policies to automatically handle retries and circuit breakers.

Recommendations

Expand table

Recommendation	Benefit
Enable Container Apps availability zone support to automatically distribute replicas across the zones in a region. Traffic is load balanced among the replicas.	If a zone failure occurs, traffic is automatically routed to the replicas in the remaining zones.
Define resource quotas and limits.	Prevent resource contention to help ensure fair allocation and avoid performance degradation. Use monitoring over time to observe actual resource usage and adjust quotas and limits accordingly.
Use volume mounts to store data outside the container in stateful applications. For enhanced data resilience, use Azure zone-redundant storage (ZRS) to help ensure high availability and durability of your data.	Ensure data persistence and integrity across container restarts and failures. Use ZRS to protect against data loss from zonal failures, which provides a robust solution for critical stateful applications.
Implement liveness, readiness, and startup health probes for your container apps. Liveness probes detect and restart containers that are in a failed state. Recommended settings: `failureThreshold: 3`, `periodSeconds: 10`, `timeoutSeconds: 5`, `successThreshold: 1`, and `initialDelaySeconds: 10` Readiness probes ensure that only healthy containers receive traffic.	Proper probe configuration helps ensure that your container apps run smoothly and can handle traffic. Improper probe configurations can lead to unintended restarts or downtime.

Recommendation	Benefit
Recommended settings: `failureThreshold: 60`, `periodSeconds: 1`, `timeoutSeconds: 1`, `successThreshold: 1`, and `initialDelaySeconds: 5` Startup probes prevent premature restarts by allowing slow-starting apps to initialize properly. Recommended settings: `failureThreshold: 60`, `periodSeconds: 1`, `timeoutSeconds: 1`, `successThreshold: 1`, and `initialDelaySeconds: 0`	
Use the Container Apps built-in observability features, such as log streaming, container console, and Azure Monitor metrics and alerts, to ensure proactive monitoring and efficient debugging.	Container Apps provides deep observability support, including integration with the .NET Aspire dashboard and Java metrics. These tools enable enhanced insights for key ecosystems. You can also use the OpenTelemetry collector for comprehensive distributed tracing and metrics collection. These features improve application reliability because you can quickly identify and resolve problems.
Implement service discovery resiliency policies, such as retries, time-outs, and circuit breakers, to proactively prevent, detect, and recover from service request failures.	Enhance the reliability of your container apps by ensuring smoother and more resilient inter-service communication.
Implement horizontal autoscaling by using scale rules that are based on HTTP requests, TCP connections, or custom metrics, such as CPU and memory thresholds. You can define custom metrics in Service Bus, Event Hubs, Apache Kafka, and Azure Cache for Redis.	The workload can dynamically manage loads and maintain high availability during peak usage times.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload.

The Security design principles provide a high-level design strategy for achieving those goals by applying approaches to the technical design of Container Apps.

Design checklist

Start your design strategy based on the design review checklist for Security and identify vulnerabilities and controls to improve the security posture. Extend the strategy to include more approaches as needed.

- ✓ **Review security baselines.** To enhance the security posture of your workload, review the security baseline for Container Apps.

- ✓ **Integrate with Microsoft Entra ID for identity and access mangement.** Use role-based access control (RBAC) with Microsoft Entra ID for least privilege access.

  Use managed identities with Microsoft Entra ID for secure, credential-free access to Azure resources.

- ✓ **Implement segmentation and network controls.** Deploy private container apps environments and use internal ingress mode for isolation from the public internet.

  Control egress traffic. To prevent data exfiltration, integrate your container apps environment into a custom virtual network that has user-defined routes to help secure outbound traffic.

- ✓ **Maintain a hardened workload's software supply chain.** Implement container-aware scanning in your secure continuous integration and continuous delivery (CI/CD) pipelines. This feature detects vulnerabilities and helps ensure the integrity of your container images. For more information, see Containers Secure Supply Chain.

  Reduce the attack surface. Harden container images and remove unused components. Use lean and minimal base images like Alpine and Chiselled Ubuntu images.

  Integrate with Microsoft Defender. Use Microsoft Defender for Containers to scan images in Azure Container Registry.

- ✓ **Encrypt data at rest and data in transit.** Use modern industry-standard methods to guard confidentiality and integrity.

  Use Azure Key Vault. Store sensitive configuration values and secrets in Key Vault to protect them from unauthorized access.

  Enable mutual Transport Layer Security (mTLS). Use mTLS to authenticate and encrypt traffic between services. This feature verifies both parties, which enhances security.

  Enforce HTTPS. Configure Envoy proxy to redirect all HTTP traffic to HTTPS. The default configuration for Envoy is `allowInsecure: false`.

- ✓ **Implement a security monitoring strategy.** Capture detailed logs for monitoring and auditing. Send system and console logs to a Log Analytics workspace, Event Hubs, or a

non-Microsoft solution for monitoring and auditing purposes. Scrub sensitive data from logs. Console logs originate from the `stderr` and `stdout` streams in the app.

Recommendations

Expand table

Recommendation	Benefit
Use managed identities to access Microsoft Entra ID protected resources. Assign a managed identity to your container app to access Microsoft Entra ID protected Azure resources, such as a storage account or key vault, without having to use keys or passwords.	Centralize identity management and eliminate the need for manual credential management. Simplify secure access to Azure resources.
Deploy container apps in a private network to integrate them into an existing virtual network. Use features such as private application connectivity, network security group (NSG) attachment, and resource communication via a private IP address.	Provide isolation from the public internet and enable secure communication between container apps and other network resources.
Use Key Vault to manage certificates and app secrets with enhanced security. Key Vault keeps certificates and other app secrets separate from your app's configuration. It also provides an access audit log that details when certificates and secrets are updated, retrieved, and rotated.	Help protect sensitive information, ensure compliance, and support secure secret management by using logging and certificate rotation capabilities.
Use Azure Application Gateway with Web Application Firewall enabled to help secure HTTP and HTTPS traffic when you publish your container app via a reverse proxy. Web Application Firewall scans incoming HTTP traffic for potential Open Web Application Security Project (OWASP) attacks.	Enhance security by protecting against common web vulnerabilities and providing centralized traffic management.
Authenticate with Microsoft Entra ID to Container Registry to avoid the use of administrative credentials. You can control access via RBAC.	Enable granular access control via RBAC for container image management, which helps ensure secure and credential-free authentication.
Use NSG rules to help secure traffic that accesses your internal container apps endpoint. NSG rules provide more granular control of which virtual networks can communicate with your container apps.	Enhance network security by restricting access to only trusted networks, which minimizes attack surfaces.
Control outbound traffic by using user-defined network routes. Control how your containers communicate with resources outside of the container environment. You can route traffic to Azure Firewall, Azure NAT Gateway, or a non-Microsoft appliance.	Ensure controlled and secure outbound traffic flow, which supports advanced routing and inspection policies.

Recommendation	Benefit
Configure logging options to send logs to a Log Analytics workspace, Event Hubs, or a non-Microsoft solution.	Support centralized monitoring, diagnostics, and auditing while ensuring compliance with sensitive data handling policies.

Cost Optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget while meeting business requirements.

The Cost Optimization design principles provide a high-level design strategy for achieving those goals and making tradeoffs as necessary in the technical design related to Container Apps and its environment.

Design checklist

Start your design strategy based on the design review checklist for Cost Optimization for investments. Fine-tune the design so that the workload is aligned with the budget that's allocated for the workload. Your design should use the right Azure capabilities, monitor investments, and find opportunities to optimize over time.

- ✓ **Select appropriate pricing plans.** Choose the most cost-effective pricing plans for your container apps based on workload requirements and expected usage patterns.

  Take advantage of the Azure savings plan for compute by committing to a fixed hourly rate for one or three years. Use a savings plan to save up to 17% compared to pay-as-you-go pricing. Optimize your budget and reduce overall expenses for long-term, predictable workloads.

- ✓ **Optimize workload component costs.** Regularly review and adjust CPU and memory allocations to match the needs of your applications. This practice prevents over-provisioning and minimizes costs.

  Use managed disk tiers. When you use volume mounts, select appropriate managed disk tiers and sizes to optimize storage costs for persistent data. Only pay for what you need.

  Use streamlined and optimized container images to improve startup times and resource efficiency, which reduces storage and network costs.

- ✓ **Optimize scaling costs.** Configure autoscaling policies to automatically scale down resources during low-demand periods and scale up during peak times. This practice ensures efficient resource usage.
- ✓ **Optimize networking costs.** Optimize network paths to minimize data transfer costs, especially for applications that have high bandwidth volumes.
- ✓ **Use cost management tools.** Use Microsoft Cost Management tools to track and analyze spending, set budgets, create cost alerts, and implement consistent tagging across resources.

  These tools provide detailed visibility into cloud spending, help identify cost-saving opportunities, ensure adherence to budgetary constraints, and allow for granular tracking and reporting of costs associated with specific workloads, applications, and environments.

Recommendations

Expand table

Recommendation	Benefit
Regularly review and adjust CPU, memory allocations, and other metrics to align with the actual needs of your applications.	Prevent over-provisioning and reduce unnecessary costs by ensuring that resources are rightsized for workloads.
Implement scale-to-zero autoscaling rules for apps that don't need to continuously run.	Eliminate costs during periods of inactivity, which ensures that you only pay for resources when you need them. This approach significantly reduces expenses for apps that have variable or infrequent usage patterns.
Choose the appropriate managed disk tiers for stateful applications. Base your selection on storage performance and capacity needs, and consider using reserved disks for predictable workloads.	Ensure that you only pay for the necessary storage performance, which prevents excess costs from over-provisioned storage. Reserved disks can provide significant cost savings for long-term storage requirements by offering discounts compared to pay-as-you-go pricing.

Operational Excellence

Operational Excellence primarily focuses on procedures for **development practices, observability, and release management**.

The Operational Excellence design principles provide a high-level design strategy for achieving those goals for the operational requirements of the workload.

Design checklist

Start your design strategy based on the design review checklist for Operational Excellence for defining processes for observability, testing, and deployment related to Container Apps.

- ✓ **Implement an infrastructure as code (IaC) deployment approach.** Implement template-based deployments by using tools like Bicep or Terraform. Ensure that all deployments are repeatable, traceable, and stored in a source code repository.

- ✓ **Automate infrastructure and workload deployments.** Use standard software solutions to manage, integrate, and automate the deployment of your workloads.

  Set up deployment pipelines to redeploy the environment to another region if a regional outage occurs. This approach helps you quickly restore and redeploy critical data and configurations in another region, which enhances disaster recovery capabilities and minimizes downtime during regional failures.

  Build automated processes to set up your environments with necessary configurations and deployments by using CI/CD pipelines.

- ✓ **Implement a comprehensive monitoring strategy.** Configure diagnostics settings to capture logs, metrics, and diagnostics data. Use tools like Azure Monitor and Application Insights to track application health and performance, identify trends in performance and reliability, and troubleshoot problems.

  Emit telemetry for workloads. Design your workloads to emit telemetry data, including liveliness and readiness statuses, to facilitate monitoring and troubleshooting.

  Monitor performance metrics. Continuously monitor key performance metrics such as CPU, memory, and network usage to identify opportunities for cost optimization and operational efficiencies.

- ✓ **Implement chaos engineering.** Apply chaos engineering practices by using tools such as Azure Chaos Studio to identify potential reliability problems within your Container Apps environment. Conduct experiments to ensure that your applications can withstand unexpected failures. Conduct performance testing by using tools such as Azure Load Testing to make sure scaling rules work as expected without disruptions to clients.

- ✓ **Implement consistent resource tagging** across all container apps and other workload resources. Consistent tagging facilitates efficient resource management, cost tracking, and automation.

- ✓ **Enforce workload governance.** Azure Policy helps ensure consistent compliance with organizational standards, automates policy enforcement, and provides centralized visibility and control over your workload's resources.

Recommendations

Expand table

Recommendation	Benefit
Store your Container Apps environment configurations as IaC, and set up deployment pipelines to redeploy the environment to another region if a regional outage occurs.	Ensure that you can quickly restore and redeploy critical data and configurations in another region, which enhances disaster recovery capabilities and minimizes downtime during regional failures.
Use revisions to implement blue-green deployments or canary deployments. Revisions require that you tag and version container images appropriately. You can use labels on revisions to facilitate sharing, for example for user acceptance testing or limited previews.	Minimize downtime and reduce risk during releases by enabling safe rollouts and quick rollbacks.
Configure Azure Monitor and Application Insights.	Track the performance and health of your container apps and provide detailed insights into application performance and reliability. Use these insights to proactively detect and resolve problems.

Performance Efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance.

The Performance Efficiency design principles provide a high-level design strategy for achieving those capacity goals against the expected usage.

Design checklist

Start your design strategy based on the design review checklist for Performance Efficiency for defining a baseline based on key performance indicators for Container Apps.

- ✓ **Develop a detailed capacity plan** to ensure that your container apps have sufficient resources to handle varying loads while avoiding over-provisioning. A capacity plan helps optimize cost and performance.

  Regularly update your plan to document the correct resource allocations, autoscale settings, and failover strategies for your container apps. This approach ensures that you

can quickly restore and redeploy critical data and configurations in another region, which enhances disaster recovery capabilities and minimizes downtime during regional failures.

- ✓ **Enable autoscaling.** Configure autoscaling policies to automatically adjust the number of container instances based on real-time demand, which ensures optimal performance during peak and off-peak times.
- ✓ **Optimize resource allocation.** Continuously monitor and adjust CPU and memory allocations based on performance metrics to ensure efficient resource usage and prevent over-provisioning.
- ✓ **Conduct load testing.** Perform regular load testing to evaluate the performance and scalability of your container apps under different conditions. Testing helps ensure that container apps can handle expected traffic levels.
- ✓ **Separate workloads.** Deploy critical and sensitive workloads in separate Container Apps environments to avoid noisy neighbor problems. Distribute workloads across multiple environments to ensure that critical applications have dedicated resources. This method also ensures that the performance demands of less-critical applications don't affect critical applications.

Recommendations

Expand table

Recommendation	Benefit
Configure autoscaling policies to automatically adjust the number of container instances in response to resource demands.	Help maintain application performance and cost-efficiency. Ensure that resources are available when needed and conserved when not needed. Use Azure Load Testing to do load testing experiments, and adjust autoscaling policies as needed.
Use the workload profiles dedicated tier for applications that require predictable performance and guaranteed resource allocation.	Provide dedicated resources for critical applications, which ensures consistent performance and reduces the risk of resource contention.
Use custom scaling metrics, such as metrics based on application-specific data, to drive autoscaling decisions.	Ensure that scaling actions are based on relevant workload demands, which improves the efficiency and responsiveness of your container apps.

Azure policies

Azure provides an extensive set of built-in policies related to Container Apps and its dependencies. Some of the preceding recommendations can be audited through Azure Policy. For example, you can check whether:

- **Diagnostics settings should be enabled:** Enable logging by category group for Container Apps environments (`microsoft.app/managedenvironments`) to send information to Storage. This setting ensures that Storage consistently collects logs and metrics for monitoring, troubleshooting, and compliance.
- **Authentication for Container Apps should be enabled:** Enable authentication to prevent anonymous HTTP requests, and authenticate requests via tokens before they reach a Container Apps environment.
- **Container Apps environments should use network injection:** Configure Container Apps environments with virtual network injection to isolate them from the public internet, enable network integration with resources on-premises or other Azure virtual networks, and achieve granular control over network traffic.
- **Public network access should be disabled:** Disable public network access to improve security by exposing the Container Apps environment through an internal load balancer. This approach blocks internet access to all container apps within the environment.
- **External network access should be disabled:** Enforce internal-only ingress to ensure that inbound communication for container apps is limited to callers within the Container Apps environment.
- **HTTPS should be enforced:** Ensure that container apps are only accessible over HTTPS to protect data in transit from network layer eavesdropping attacks.
- **Managed identities should be enabled:** Require managed identities for Container Apps environments to securely authenticate to any resource that supports Microsoft Entra ID authentication.

For comprehensive governance, review the Azure Policy built-in definitions for Container Apps and other policies that might affect the security of the networks.

Azure Advisor recommendations

Azure Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments.

For more information, see Azure Advisor.

Next steps

- Deploy your first container app by using the Azure CLI
- Cloud design patterns that support reliability
- Mission-critical workloads
- Site reliability engineering resources
- Container Apps overview
- Container Apps landing zone accelerator

Architecture best practices for Azure Cosmos DB for NoSQL

Article • 05/07/2025

Azure Cosmos DB is a fully managed database solution that can host multiple database types like NoSQL, relational, vector, and table databases. It can accommodate most use cases for your workload. The recommendations in this guide focus on the Azure Cosmos DB for NoSQL variant. You can also apply some foundational recommendations to other variants.

This article assumes that as an architect, you've reviewed the Azure data options and chose Azure Cosmos DB for NoSQL as the data store for your workload. The guidance in this article provides architectural recommendations that are mapped to the principles of the Well-Architected Framework pillars.

> **Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern along with design strategies localized to the technology scope.
>
> Also included are recommendations for the technology capabilities that can help materialize those strategies. The recommendations don't represent an exhaustive list of all configurations that are available for Azure Cosmos DB for NoSQL and its dependencies. Instead, they list the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept or to optimize your existing environments.

Technology scope

This review focuses on the interrelated decisions for the following Azure resources:

- Azure Cosmos DB for NoSQL

Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover fast from failures**.

Reliability design principles provide a high-level design strategy applied for individual components, system flows, and the system as a whole.

Design checklist

Start your design strategy based on the design review checklist for Reliability. Determine its relevance to your business requirements while keeping in mind the consistency levels, cross-region replication, and provisioned throughput. Extend the strategy to include more approaches as needed.

- ✓ **Choose a simple design and avoid unnecessary features.** Don't adopt functionality that unnecessarily adds complexity to your workload, like multiple-primary writes and custom indexing. Azure Cosmos DB supports these features, but they can add significant complexity and operational burden. Carefully evaluate whether your workload requires any advanced features.

  Offload workload responsibilities and cross-cutting concerns to other services, such as Microsoft Entra ID for authentication and authorization or Azure Monitor and Application Insights for monitoring.

- ✓ **Identify and prioritize workload flows.** Not all flows in the application are equally important. Identify the critical paths and assign priorities to each flow to guide your design decisions. User flow design can influence your service tiers, features, and the capacity that you allocate to the Azure Cosmos DB database.

  To help identify critical flows that might need special treatment, capture the processing requirements that set them apart in terms of data processing capabilities. For example, some flows might require higher throughput or be more sensitive to latency problems. And some solution flows might require different transaction consistency levels.

- ✓ **Use failure mode analysis to identify potential failures in your workload.** Plan mitigation strategies for potential failures. The following table shows examples of failure mode analysis.

Expand table

Failure	Mitigation
High latency and timeouts after rollout to users in a new geographical region	Enable multiple-region replication to an Azure region that's closer to the new users.
Throttling because of exceeding request units (RUs)	Implement retry logic. Monitor RU consumption and adjust throughput settings as needed.
Azure Cosmos DB service failure in a single zone in a single Azure region	Configure availability zone support when you create a database.

For more information, see Failure mode analysis for Azure applications.

- ✓ **Build redundancy to improve resiliency and help meet reliability targets.** Design your database account deployment so it spans at least two regions in Azure. Distribute your account across multiple availability zones if your Azure region supports it.

  Evaluate the multiple-region and single-region write strategies for your workload. For single-region write, design your workload to have at least a second read region for failover. Enable automatic failover for single-region write and multiple-region read scenarios. For multiple-region write, compare the tradeoffs in complexity and consistency against the advantages of writing to multiple regions. For more information, see Expectations during a regional outage for single-region and multiple-region write accounts.

  Consider how your consistency level and replication mode affects the recovery point objective (RPO) in a region-wide outage.

  Design an end-to-end test of high availability for your application that includes testing database failover and fail back.

- ✓ **Use native disaster recovery and backup features.** Implement database and container backups and restores to meet your requirements. Restore scenarios include point-in-time restore, recovering from accidental destructive operations, restoring deleted resources, and restoring to another region at a point in time. Configure your account with continuous backup. Choose the appropriate retention period based on your business requirements.

- ✓ **Align your Azure Cosmos DB design with industry-standard application design guidance and patterns.** Explore the guide to design resilient applications, review the default retry policy for the SDKs, and plan for custom handling for specific transient errors. These guides provide best practices to make application code resilient to transient errors.

Recommendations

Expand table

Recommendation	Benefit
Distribute your Azure Cosmos DB account across availability zones when available.	Availability zones provide segregated power, networking, and cooling, which helps isolate hardware failures to a subset of your replicas. When you don't use the availability zones feature, Azure Cosmos DB spans multiple replicas across a single, randomly selected availability zone.
Configure your Azure Cosmos DB account to span at least two regions.	Spanning multiple regions helps ensure that your workload is resilient to regional outages.

Recommendation	Benefit
Enable service-managed failover for your account. Understand the tradeoffs with service-managed failover, and plan for forced failover if necessary. Temporarily disable service-managed failover to validate the end-to-end high availability of your application. You can start a manual failover by using a script or the Azure portal.	Service-managed failover allows Azure Cosmos DB to automatically change the write region of a multiple-region account to preserve availability. This change occurs without user interaction.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload.

The Security design principles provide a high-level design strategy for achieving those goals by applying approaches to the technical design of Azure Cosmos DB for NoSQL.

Design checklist

Start your design strategy based on the design review checklist for Security and identify vulnerabilities and controls to improve the security posture. Extend the strategy to include more approaches as needed.

- ✓ **Review security baselines.** To enhance the security posture of your workload, review the security checklist for Azure databases and the security baseline for Azure Cosmos DB. At a minimum, implement the data protection and identity management security baselines to help secure your Azure Cosmos DB account.

  Assess service-level compliance and certifications in the context of current global personal data requirements.

- ✓ **Implement strict, conditional, and auditable identity and access management.** Use Microsoft Entra ID for your workload's authentication and authorization needs. Microsoft Entra ID provides centralized authorization and access management.

  For control plane and data plane access to your account, create roles, groups, and assignments based on the principle of least-privilege access. Consider disabling key-based authentication.

- ✓ **Encrypt data.** Encrypt data at rest or data in motion by using service-managed keys or customer-managed keys.

- ✓ **Apply network segmentation and security controls.** Create intentional segmentation and perimeters in your network design and apply defense-in-depth principles by using localized network controls at all network boundaries.

  Reduce the surface attack area by using private endpoints in accordance with the security baseline for Azure Cosmos DB.

- ✓ **Implement a holistic security monitoring strategy.** Audit user access, security breaches, and resource operations by using control plane logs.

  Monitor data egress, data changes, usage, and latency by using data plane metrics.

- ✓ **Use secure development and testing practices.** Follow best software development practices for secure access to data. Follow secure coding practices and perform secure code reviews when you develop applications that interact with Azure Cosmos DB.

Recommendations

Expand table

Recommendation	Benefit
Disable public endpoints and use private endpoints whenever possible.	Using private networking through private endpoints reduces the surface area susceptible to attacks on your account.
Use role-based access control (RBAC) to limit control-plane access to specific identities and groups within well-defined assignments.	Using RBAC helps prevent unauthorized access to your account. Assign appropriate roles and permissions to users or applications that access Azure Cosmos DB based on the principle of least privilege.
Create virtual network endpoints and rules to limit access to the account. Use network security groups (NSGs) to control traffic to and from Azure Cosmos DB resources.	Virtual network service endpoints, NSGs, and firewall rules help restrict access to your Azure Cosmos DB account and help protect your data from unauthorized access.
Protect against common security vulnerabilities like injection attacks, cross-site scripting, or insecure direct object references. Implement input validation, parameterized queries, and appropriate error handling for common HTTP status codes to prevent security risks.	Input validation helps prevent malicious data from being processed by your application. This approach blocks potentially harmful data that could lead to security breaches or data corruption. Proper error handling enables your application to respond to errors in a controlled manner and prevents the exposure of sensitive information.

Recommendation	Benefit
Monitor control plane logs to detect unauthorized modifications to your Azure Cosmos DB account.	Monitoring helps you track access patterns and audit logs to ensure that your database remains secure and compliant with relevant data protection regulations. Monitoring data-plane metrics can also help identify unfamiliar patterns that might reveal a security breach.
Enable Microsoft Defender for Azure Cosmos DB to trigger security alerts when anomalous activities occur.	Microsoft Defender detects attempts to exploit databases in your account. Defender detects potential SQL injections, suspicious access patterns, and other potential exploitation activities.

Cost Optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget while meeting business requirements.

The Cost Optimization design principles provide a high-level design strategy for achieving those goals and making tradeoffs as necessary in the technical design related to Cosmos DB for No SQL and its environment.

Design checklist

Start your design strategy based on the design review checklist for Cost Optimization for investments. Fine-tune the design so that the workload is aligned with the budget that's allocated for the workload. Your design should use the right Azure capabilities, monitor investments, and find opportunities to optimize over time.

- ✓ **Optimize your Azure Cosmos DB scaling strategy.** Understand how Azure Cosmos DB handles storage and throughput scaling. Determine the right balance of performance and cost to meet your requirements.

  Select a throughput allocation schema that suits your workload. Review the benefits of standard and autoscale throughput distributed at the database or container level. Also, consider serverless options when appropriate. Analyze your workload's traffic patterns to select the most suitable throughput allocation scheme.

  Determine a partition key or set of partition keys that have high cardinality and remain unchanged. Use existing guidance and best practices to help select an appropriate partition key. Also, consider your indexing policy when you determine a partition key.

- ✓ **Optimize your Azure Cosmos DB data costs.** Take an inventory of data and calculate the expected overall data storage for your workload. The size of both items and indexes influence your data storage cost. Calculate the impact of replication and backup on storage costs.

  Create a strategy to automatically remove older items that are no longer used or necessary. If required, export these items to a lower-cost storage solution before you remove them.

- ✓ **Optimize your queries for cost.** Evaluate your most common queries that minimize cross-partition lookups. Use this information to inform the process of selecting a partition key or customizing an indexing policy.

  Use projection to reduce throughput costs of large query result sets. Author queries to project only the minimal number of fields required from a result set. If calculations on fields are necessary, evaluate the throughput cost of performing those calculations server side versus client side.

  Avoid using unbounded cross-partition queries. Evaluate and author queries to ensure that they search within a single logical partition whenever possible. Use query filters to control which logical partitions the query targets. If a query must search across logical partitions, bound the query to only search a subset of logical partitions instead of a full scan.

  Design an indexing policy that considers the common operations and queries in your workload.

Recommendations

Expand table

Recommendation	Benefit
Monitor the usage and patterns of RUs per second. Use queries and other data research techniques to find antipatterns in your application code.	Using metrics to monitor RU consumption starting early in your Azure Cosmos DB deployment helps you identify inefficient queries and operations that consume excessive RUs. You can optimize queries to lower costs.
Customize your indexing policy to align with your workload. Use an indexing policy based on only the paths that you need to index for your common queries. For write-heavy workloads, disable automatic indexing of columns not used in queries.	The default indexing policy indexes all paths in an item and can significantly affect RU consumption and costs. Customizing your indexing policy allows you to tailor the indexing strategy to your specific requirements. This approach helps ensure optimal performance and resource usage.

Recommendation	Benefit
Select partition keys for your workload that evenly distribute throughput consumption and data storage across logical partitions. Avoid hot partitions that receive a disproportionate amount of traffic. Unbalanced partitions can increase throughput costs and transient errors. The selection should also minimize the number of unbounded cross-partition queries. Use the most common search queries to determine potential partition keys that likely run only single-partition or bounded cross-partition queries.	Choosing the right partition key evenly distributes data, which reduces the likelihood of hot partitions and optimizes throughput consumption. This approach can significantly improve efficiency and lower costs.
Select the right provisioned throughput option for your Azure Cosmos DB instance. Choose between serverless or provisioned throughput and manual provisioning or autoscale. Apply these options at the database or container level, depending on the specific needs of your workload. Generally, small workloads and dev/test workloads benefit from serverless throughput or manual shared throughput at the database level. Larger, mission-critical workloads might benefit from provisioned throughput at the container level.	Selecting the right provisioned throughput option helps optimize cost by avoiding over-provisioning or under-provisioning. It also reduces operational costs while ensuring that your application can effectively handle varying traffic patterns.
Configure the default consistency level for your application. When appropriate, downgrade the default consistency level in client sessions. Consider the higher costs associated with reads at stronger consistency levels. You might not always need to change the standard default consistency level or override it in client sessions.	Choosing the right consistency level for your application ensures that you achieve the desired balance between data consistency, availability, performance, and cost.
For dev/test workloads, use the Azure Cosmos DB emulator, which is also available as a Docker container image.	The emulator reduces costs for dev/test and continuous integration use cases.
Use transactional batch operations. Design partitions to take advantage of transactional batch operations within a logical partition key for inserting data. Use batch operations in client-side SDKs to insert, update, or delete multiple documents in a single transaction request.	Using transactional batch operations in Azure Cosmos DB can help reduce the number of individual requests. This approach helps reduce latency and improve throughput efficiency.

Recommendation	Benefit
Implement time-to-live (TTL) to automatically delete unnecessary data. Export the expired data to a lower-cost storage solution if needed.	Using TTL ensures that your database remains clutter free, which optimizes storage costs. Exporting expired data to a lower-cost storage solution can further reduce expenses while preserving historical data for compliance purposes.
Consider an analytical store for heavy aggregations.	Azure Cosmos DB analytical store automatically syncs your data to a separate column store to optimize for large aggregations, reporting, and analytical queries.

Operational Excellence

Operational Excellence primarily focuses on procedures for **development practices, observability, and release management**.

The Operational Excellence design principles provide a high-level design strategy for achieving those goals for the operational requirements of the workload.

Design checklist

Start your design strategy based on the design review checklist for Operational Excellence for defining processes for observability, testing, and deployment related to Azure Cosmos DB for NoSQL.

- ✓ **Monitor your Azure Cosmos DB instances.** Include Azure Cosmos DB instances across regions in your workload monitoring solution.

  Create identifiers in the client application to differentiate workloads. Consider flags, such as the user-agent suffix, to identify what workload each log entry or metric should be associated with.

  Draft a log and metrics monitoring strategy to differentiate between different workloads, flag exceptional scenarios, track patterns in exceptions and errors, and track host machine performance.

  Define multiple alerts to monitor throttling, analyze throughput allocation, and track the size of your data.

  Plan expected metric thresholds based on partition and index design. Create a plan to monitor those metrics to determine how close they are to the planned thresholds.

- ✓ **Automate deployments through infrastructure as code (IaC).** Incorporate all Azure Cosmos DB changes, including infrastructure deployments and configuration changes, in your code deployment practices.

  Create and enforce best practices to automate the deployment of your Azure Cosmos DB for NoSQL account and resources.

  Ensure that your team uses the same templates to deploy to other nonproduction environments.

Recommendations

Expand table

Recommendation	Benefit
Ensure that application developers use the latest version of the developer SDK. For more information, see .NET SDK and Java SDK. Each Azure Cosmos DB for NoSQL SDK has a minimum recommended version.	Using the latest versions of the developer SDKs helps maintain the performance, security, and compatibility of your applications.
Capture supplemental diagnostics by using the diagnostics injection techniques for each SDK. These techniques add supplemental information about the workload, alongside default metrics and logs. For more information, see .NET SDK and Java SDK.	By injecting diagnostics, you can capture custom metrics, traces, and logs to improve troubleshooting and optimize your applications. You can monitor specific aspects of your workload that default metrics and logs don't cover.
Create alerts for host machine resources.	Connectivity and availability problems might occur because of client-side host machine problems. Monitor resources such as CPU, memory, and storage on host machines by using client applications that use the Azure Cosmos DB for NoSQL SDKs.
Create alerts for throughput throttling by using metrics like the normalized RU consumption. This metric measures the percentage usage of provisioned throughput on a database or container. Use alerts to track when this metric exceeds expected thresholds. Over time, review and adjust alerts as you learn more about your workload.	If throughput throttling is consistently at 100%, requests likely return a transient error. By creating alerts based on this metric, you can receive notifications when the usage exceeds your defined thresholds. You can take corrective actions before it affects your application's performance.
Use query performance metrics to track the performance of your top queries over time. If	Performance metrics allow you to better understand how your queries run and where potential

Recommendation	Benefit
query performance is poor, troubleshoot performance and apply query best practices.	bottlenecks exist. This information helps you make informed decisions about indexing policies and query optimizations.
Use templates to automatically deploy account resources. Consider Azure Resource Manager, Bicep, or Terraform templates to automate the deployment of your account and subsequent resources.	IaC templates help define your infrastructure in a descriptive model. They facilitate repeatability and consistency and help maintain uniform and predictable workload environments.
Track key metrics to identify common problems in your workload like RU usage by partition, throttling, and request volumes by type.	Monitoring these key metrics helps you understand how the database performs under various conditions. Monitoring also helps identify common problems so that you can take corrective action.

Performance Efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance.

The Performance Efficiency design principles provide a high-level design strategy for achieving those capacity goals against the expected usage.

Design checklist

Start your design strategy based on the design review checklist for Performance Efficiency for defining a baseline based on key performance indicators for Azure Cosmos DB for NoSQL.

- ✓ **Plan and proactively manage your Azure Cosmos DB capacity.** Define a performance baseline for your application. Measure how many concurrent users and transactions you might need to handle. Consider workload characteristics such as your average user flow, common operations, and spikes in usage.

  Research your target users. Determine which Azure regions are closest to them.

- ✓ **Optimize your data design.** Design items so their corresponding JSON documents are as small as possible. Consider splitting data across multiple items if necessary.

  Keep items less than **100 KB** in size. Larger items consume more throughput for common read and write operations. Queries on larger items that project all fields can also add significant throughput cost.

- ✓ **Select the right tier, features, and billing model.** Determine if your workload requires an analytical store. Consider analytical stores and services like Azure Synapse Link for complex queries.
- ✓ **Optimize the deployment model.** Deploy Azure Cosmos DB for NoSQL to regions closest to your end users. | Reduce latency by deploying Azure Cosmos DB for NoSQL to the regions closest to your end users as much as possible.
- ✓ **Optimize queries for performance.** Review the query performance tips and apply the strategies that apply to your use cases.

  Identify queries that use one or more ordering fields. Also, identify operations that affect multiple fields. Include these fields explicitly in the indexing policy design.

  Design large workloads to use bulk operations whenever possible.

  Identify queries on child arrays and determine if they're candidates for more efficient subqueries.

  Research your most common and complex queries. Identify queries that use multiple lookups, joins, or aggregates. Consider these queries in any design considerations for the partition key or indexing policy.

  For the most common queries, determine the number of results that you expect for each page. This number helps you formalize a buffered item count for prefetched results.
- ✓ **Optimize code logic.** Use the singleton pattern for the `CosmosClient` class in most SDKs. Use the client class in most SDKs as a singleton. The client class manages its own lifecycle and is designed to not be disposed. Constantly creating and disposing instances can result in reduced performance.

Recommendations

Expand table

Recommendation	Benefit
Use tools like the capacity calculator to determine the amount of throughput required for your performance baseline. Use features like autoscale to scale your actual throughput to more closely match your actual workload. Monitor your actual throughput consumption afterwards and make adjustments.	Using the capacity calculator helps you make informed decisions about provisioning resources initially. You can ensure that your database can handle the expected load without unnecessary costs. Autoscale throughput automatically adjusts the provisioned throughput to match the actual workload needs.

Recommendation	Benefit
Use optimization techniques on the client and server sides when appropriate. Take advantage of the built-in integrated cache. Configure the SDK to manage how many items are prefetched, or *buffered*, and returned for each page.	Code optimization techniques can improve the user experience and make your client and server side application more efficient by helping reduce the RU charges for repeated reads and queries. These benefits can reduce costs.
Deploy Azure Cosmos DB instances in the regions closest to your end users. Configure the .NET or Java SDK to prefer regions closer to your end users. Take advantage of read replication to provide optimized read performance regardless of how you configure write operations in single or multiple regions.	Deploying your database to the regions closest to your end users helps reduce latency and improve user experience. Read replication enables performant read operations regardless of how the write operations are configured.
Configure the SDK for Direct mode for the best performance. This mode allows your client to open Transmission Control Protocol (TCP) connections directly to partitions in the service and send requests directly with no intermediary gateway.	Direct mode provides better performance because there are fewer network hops.
Use the bulk feature of client SDKs in scenarios that require high throughput. The bulk feature automatically manages and batches operations to maximize throughput.	Using the bulk feature reduces the number of back-end requests and efficiently distributes tasks across partitions. This approach improves performance and resource usage.
Disable indexing for bulk operations. If there are several insert, replace, or upsert operations, disable indexing to improve the speed of the operation while using the bulk support of the corresponding SDK. You can immediately reenable indexing later.	Disabling indexing during bulk operations reduces the overhead associated with maintaining the index. You can better utilize resources for bulk operations, which accelerates data insertion and updates.
Create composite indexes for fields in complex operations. Consider using composite indexes for `ORDER BY` statements that have multiple fields.	Composite indexes can increase the efficiency of operations on multiple fields, which makes data retrieval fast and efficient.
Optimize host client machines for the SDKs. For most common cases, use at least four cores and eight GB of memory on 64-bit host machines that use the SDKs for .NET or Java. Enable accelerated networking on host machines.	Using machines that have higher specifications ensures that the SDKs can handle more operations simultaneously. Accelerated networking can help reduce latency, which increases response time.
Use subqueries strategically to optimize queries that join large data sets. Optimize self-join expressions by using subqueries to filter the arrays before joining arrays within the item. For cross-partition queries, optimize your query to	Queries that join child arrays can increase in complexity if multiple arrays are involved and not filtered. By employing subqueries, arrays can be filtered before joining them, which increases the efficiency of self-join expressions.

Recommendation	Benefit
include a filter on the partition key to optimize the routing of your query to the least amount of partitions possible.	
Use analytical workloads for the most complex queries. If you run frequent aggregations or join queries over large containers, consider enabling the analytical store and doing queries in Azure Synapse Analytics.	Analytical workloads are isolated from transactional workloads and optimized for handling complex queries, which prevents performance degradation and ensures faster results.

Azure policies

Azure provides an extensive set of built-in policies related to Azure Cosmos DB for NoSQL and its dependencies. Some of the preceding recommendations can be audited through Azure Policy. For example, you can check whether:

- The Azure Cosmos DB account is deployed to at least two regions.
- Azure Cosmos DB database accounts are appropriately configured for zone redundancy.
- Azure Cosmos DB database accounts should have local authentication methods disabled so that Microsoft Entra identities are required for authentication.
- Public network access is disabled so that your Azure Cosmos DB account isn't exposed on the public internet.
- The maximum throughput should be restricted when you create Azure Cosmos DB databases and containers through the resource provider.

For comprehensive governance, review the Azure Policy built-in definitions for Azure Cosmos DB for NoSQL and other policies that might affect the security of the data storage.

Azure Advisor recommendations

Azure Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments.

For more information, see Azure Advisor.

Related resources

- Multitenancy and Azure Cosmos DB
- Common Azure Cosmos DB use cases
- Serverless database computing by using Azure Cosmos DB

Azure Databricks and security

Article • 11/14/2023

Azure Databricks is a data analytics platform optimized for Azure cloud services. It offers three environments for developing data intensive applications:

- Databricks SQL
- Databricks Data Science and Engineering
- Databricks Machine Learning

To learn more about how Azure Databricks improves the security of big data analytics, reference Azure Databricks concepts.

The following sections include design considerations, a configuration checklist, and recommended configuration options specific to Azure Databricks.

Design considerations

All users' notebooks and notebook results are encrypted at rest, by default. If other requirements are in place, consider using customer-managed keys for notebooks.

Checklist

Have you configured Azure Databricks with security in mind?

- ✓ Use Microsoft Entra ID credential passthrough to avoid the need for service principals when communicating with Azure Data Lake Storage.
- ✓ Isolate your workspaces, compute, and data from public access. Make sure that only the right people have access and only through secure channels.
- ✓ Ensure that the cloud workspaces for your analytics are only accessible by properly managed users.
- ✓ Implement Azure Private Link.
- ✓ Restrict and monitor your virtual machines.
- ✓ Use Dynamic IP access lists to allow admins to access workspaces only from their corporate networks.
- ✓ Use the VNet injection functionality to enable more secure scenarios.
- ✓ Use diagnostic logs to audit workspace access and permissions.
- ✓ Consider using the Secure cluster connectivity feature and hub/spoke architecture to prevent opening ports, and assigning public IP addresses on cluster nodes.

Configuration recommendations

Explore the following table of recommendations to optimize your Azure Databricks configuration for security:

Recommendation	Description
Ensure that the cloud workspaces for your analytics are only accessible by properly managed users.	Microsoft Entra ID can handle single sign-on for remote access. For extra security, reference Conditional Access.
Implement Azure Private Link.	Ensure all traffic between users of your platform, the notebooks, and the compute clusters that process queries are encrypted and transmitted over the cloud provider's network backbone, inaccessible to the outside world.
Restrict and monitor your virtual machines.	Clusters, which execute queries, should have SSH and network access restricted to prevent installation of arbitrary packages. Clusters should use only images that are periodically scanned for vulnerabilities.
Use the VNet injection functionality to enable more secure scenarios.	Such as: - Connecting to other Azure services using service endpoints. - Connecting to on-premises data sources, taking advantage of user-defined routes. - Connecting to a network virtual appliance to inspect all outbound traffic and take actions according to allow and deny rules. - Using custom DNS. - Deploying Azure Databricks clusters in existing virtual networks.
Use diagnostic logs to audit workspace access and permissions.	Use audit logs to see privileged activity in a workspace, cluster resizing, files, and folders shared on the cluster.

Source artifacts

Azure Databricks source artifacts include the Databricks blog: Best practices to secure an enterprise-scale data platform.

Next step

Azure Database for MySQL and cost optimization

Azure Database for MySQL and cost optimization

Article • 11/14/2023

Azure Database for MySQL is a relational database service in the Microsoft cloud based on the MySQL Community Edition. You can use either Single Server or Flexible Server to host a MySQL database in Azure. It's a fully managed database as a service offering that can handle mission-critical workloads with predictable performance and dynamic scalability.

For more information about how Azure Database for MySQL supports cost optimization for your workload, reference Server concepts, specifically, Stop/Start an Azure Database for MySQL.

The following sections include design considerations, a configuration checklist, and recommended configuration options specific to Azure Database for MySQL.

Design considerations

Azure Database for MySQL includes the following design considerations:

- Take advantage of the scaling capabilities of Azure Database for MySQL to lower consumption cost whenever possible. To scale your database up and down, as needed, reference the following Microsoft Support article, which covers the automation process using runbooks: How to autoscale an Azure Database for MySQL/PostgreSQL instance with Azure run books and Python.
- Plan your Recovery Point Objective (RPO) according to your operation level requirement. There's no extra charge for backup storage for up to `100%` of your total provisioned server storage. Extra consumption of backup storage will be charged in `GB/month`.
- The cloud native design of the Single-Server service allows it to support `99.99%` of availability, eliminating the cost of passive *hot* standby.
- Consider using Flexible Server SKU for non-production workloads. Flexible servers provide better cost optimization controls with ability to stop and start your server. They provide a burstable compute tier that is ideal for workloads that don't need continuous full compute capacity.

Checklist

Have you configured Azure Database for MySQL with cost optimization in mind?

- ✓ Choose the appropriate server size for your workload.
- ✓ Consider Reserved Capacity for Azure Database for MySQL Single Server.

Configuration recommendations

Explore the following table of recommendations to optimize your Azure Database for MySQL configuration for cost optimization:

Recommendation	Description
Choose the appropriate server size for your workload.	Configuration options: Single Server and Flexible Server.
Consider Reserved Capacity for Azure Database for MySQL Single Server.	Compute costs associated with Azure Database For MySQL Single Server Reservation Discount. Once you've determined the total compute capacity and performance tier for Azure Database for MySQL in a region, this information can be used to reserve the capacity. The reservation can span one or three years. You can realize significant cost optimization with this commitment.

Azure Database for PostgreSQL and cost optimization

Architecture best practices for Azure Database for PostgreSQL

05/16/2025

Azure Database for PostgreSQL is a relational database service in Azure that's based on the PostgreSQL open-source relational database. It's a fully managed, cloud-based database solution that supports mission-critical workloads with predictable performance, security, high availability, and dynamic scalability. Azure Database for PostgreSQL is built on the community edition of the PostgreSQL database engine. It's compatible with the PostgreSQL server community edition and supports PostgreSQL extension features, such as PostGIS and TimescaleDB.

This article assumes that as an architect, you've reviewed the Azure data options and chose Azure Database for PostgreSQL as the data store for your workload. The guidance in this article provides architectural recommendations that are mapped to the principles of the Well-Architected Framework pillars.

> **ⓘ Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern along with design strategies localized to the technology scope.
>
> Also included are recommendations for the technology capabilities that can help materialize those strategies. The recommendations don't represent an exhaustive list of all configurations that are available for Azure Database for PostgreSQL and its dependencies. Instead, they list the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept or to optimize your existing environments.
>
> Foundational architecture that demonstrates the key recommendations: Reliable Web App pattern for Java.

Technology scope

This review focuses on the interrelated decisions for the following Azure resources:

- Azure Database for PostgreSQL

Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover fast from failures**.

Reliability design principles provide a high-level design strategy applied for individual components, system flows, and the system as a whole.

Design checklist

- ✓ **Familiarize yourself with Azure Database for PostgreSQL product reliability guidance.** For more information, see the following resources:
  - Business continuity overview
  - Reliability and high availability
  - Backup and restore
  - Geo-disaster recovery
- ✓ **Align your reliability and recovery targets for Azure Database for PostgreSQL with your workload targets.** Choose an appropriate Azure Database for PostgreSQL SKU that can support your reliability and recovery targets.
- ✓ **Determine the appropriate high availability and redundancy configurations for your Azure Database for PostgreSQL instances.** Determine whether you need zone-redundant or zonal configurations to meet your reliability requirements.
- ✓ **Incorporate the recovery of your Azure Database for PostgreSQL instances into your disaster recovery planning and exercises.** Ensure that your entire workload is recoverable according to your recovery targets.
- ✓ **Incorporate your Azure Database for PostgreSQL instances into your observability platform.** Enable enhanced metrics to monitor the health of your instance. Include high availability health monitoring in your monitoring solution.

Recommendations

Expand table

Recommendation	Benefit
Select the appropriate high availability configuration.	When high availability is configured, the Azure Database for PostgreSQL server automatically provisions and manages a standby replica. This setup ensures that the service remains available and doesn't lose data during a zone outage.

Recommendation	Benefit
Configure geo-redundant backups.	Cross-region read replicas can be deployed to protect your databases from region-level failures. Geo-redundant backups are enabled in selected regions and help with disaster recovery if the primary server region is unavailable. Geo-redundancy can also be achieved by using an Azure Backup vault for long-term storage of the recovery points. If a regional outage or disaster occurs, you can use Backup to restore the database server to an Azure-paired region, which minimizes downtime. Azure Backup also provides geo-redundancy for Azure Database for PostgreSQL. This feature increases efficiency and reduces downtime during disasters or regional outages.
Test your backup and restore strategy reguarly.	Regularly testing your backup and restore strategy ensures that you can recover your databases and maintain operations if a failure occurs.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload.

The Security design principles provide a high-level design strategy for achieving those goals by applying approaches to the technical design of Azure Database for PostgreSQL.

Design checklist

- ✓ **Review the security baseline.** To enhance the security posture of your workload, review the Azure security baseline for Azure Database for PostgreSQL.
- ✓ **Implement strict, conditional, and auditable identity and access management.** Use Microsoft Entra ID for authentication and authorization to enhance identity management.
- ✓ **Apply network segmentation and security controls.** Use built-in firewalling at the server level alongside virtual network firewalling mechanisms to isolate your instance from other workload components and public networks.
- ✓ **Use native security features to add extra layers of protection.** Help prevent malicious activities by configuring row-level security and connection throttling.

Recommendations

Expand table

Recommendation	Benefit
Implement network security groups and firewalls to control access to your database.	As part of the Zero Trust model for security, use network segmentation to restrict communication paths between components, such as application and database servers, to only what's needed. This feature can be implemented by using network security groups and application security groups.
Connect to your databases over Azure Private Link.	Private Link allows you to create private endpoints for Azure Database for PostgreSQL to extend the service inside your virtual network.
Use Microsoft Entra ID for authentication and authorization to enhance identity management.	You can use Microsoft Entra authentication to connect to Azure Database for PostgreSQL by using managed identities in Microsoft Entra.
Configure row-level security.	Row-level security is a PostgreSQL security feature that allows database administrators to define policies to control how specific rows of data display and operate for one or more roles. Row-level security is an extra filter that you can apply to a PostgreSQL database table.
If needed for compliance, use customer managed keys (CMKs) for data encryption, and store your keys in Azure Key Vault.	CMKs give you full control of your encryption key's life cycle, including key rotation, to align with corporate policies. Key Vault enables centralized management and organization of your encryption keys within your own dedicated Key Vault instances.
Enable connection throttling for IP addresses that have excessive failed sign-in attempts.	Setting the `connection_throttling` server parameter to `enabled` helps protect your databases from malicious sign-in attempts and distributed denial of service (DDoS) attacks by limiting repeated connection attempts from the same IP address.
Conduct security audits regularly.	Regularly conducting security audits helps identify and remediate potential vulnerabilities.

Cost Optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget while meeting business requirements.

The Cost Optimization design principles provide a high-level design strategy for achieving those goals and making tradeoffs as necessary in the technical design related to Azure Database for PostgreSQL and its environment.

Design checklist

- ✓ **Estimate the initial cost.** As part of your cost modeling exercise, use the Azure pricing calculator to evaluate the approximate costs associated with Azure Database for PostgreSQL in your workload.
- ✓ **Choose the right service tier for your workload.** Determine whether the Burstable, General Purpose, or Memory Optimized pricing tier for compute resources meets your use case needs.
- ✓ **Apply a well-informed scaling strategy.** To determine when and how to scale your Azure Database for PostgreSQL resources, continuously observe and analyze your workload capacity and demand.
- ✓ **Take advantage of available discounts.** Consider one or three year compute reservations. Reserved instances can save you significant costs for compute resources.
- ✓ **Use your provisioned storage.** There's no extra charge for backup storage up to 100% of your total provisioned server storage.
- ✓ **Understand redundancy costs.** Using zone-redundant or zonal configurations doubles your instance costs, so carefully consider your redundancy design.
- ✓ **Understand protected instance and backup storage costs.** When you back up Azure Databse for PostgreSQL by using Azure Backup, you incur protected instance fees for every 250 GB and backup storage fees based on total data stored and redundancy type.
- ✓ **Deploy to the same region as an app.** Deploy to the same region as your applications to minimize transfer costs.
- ✓ **Consolidate databases and servers.** When practical, consolidating multiple databases and servers into a single server can help reduce costs.

Recommendations

Expand table

Recommendations	Benefits
Pick the right tier and SKU.	Choosing the right tier and SKU helps you avoid wasting money on over-provisioned resources. Azure Advisor provides recommendations, include server rightsizing suggestions, to optimize and reduce your overall Azure spending.

Recommendations	Benefits
Scale compute and storage resources when your workload demand changes.	You can scale compute resources vertically (up or down) and horizontally (out or in) as necessary. You can't scale storage down after scaling up.
Use the start/stop feature.	Use the start/stop feature to stop the server from running when it isn't needed.

Operational Excellence

Operational Excellence primarily focuses on procedures for **development practices, observability, and release management**.

The Operational Excellence design principles provide a high-level design strategy for achieving those goals for the operational requirements of the workload.

Design checklist

- ✓ **Optimize the recoverability of your databases.** Define backup and retention policies to meet your compliance requirements.
- ✓ **Automate operational tasks.** Use automation tasks to automatically perform tasks like starting and stopping a server, scaling resources, and other tasks.
- ✓ **Monitor database health and performance.** Collect and analyze logs and metrics from your instance to proactively detect potential problems.

Recommendations

Expand table

Recommendation	Benefits
Use an Azure Backup policy individually or together with the native automated backups to back up databases.	Azure Database for PostgreSQL provides automated backups and point-in-time restore for your database. You can configure the retention period for backups up to 35 days. Use Azure Backup to define how and when backups are created, the retention period for recovery points, and the rules for data protection and recovery. You can retain recovery points in an Azure Backup vault for up to 10 years. You can also manage the backup and restore operations by using Azure Business Continuity Center.

Recommendation	Benefits
Schedule custom maintenance windows to apply service updates.	You can define a custom schedule for each Azure Database for PostgreSQL in your Azure subscription. With a custom schedule, you can specify your maintenance window for the server by choosing the day of the week and the start time of the one-hour maintenance window.
Monitor your server to ensure that it's healthy and performing as expected.	Azure Database for PostgreSQL has various metrics that provide insight into the behavior of the resources that support the Azure Database for PostgreSQL instance. You can also enable enhanced metrics to get fine-grained monitoring and alerting on databases.

Performance Efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance.

The Performance Efficiency design principles provide a high-level design strategy for achieving those capacity goals against the expected usage.

Design checklist

- ✓ **Optimize queries.** Use native features to find opportunities to optimize queries.
- ✓ **Optimize indexes.** Use features such as index tuning to automatically analyze query patterns and receive actionable recommendations for creating or dropping indexes to improve performance.
- ✓ **Offload read-only operations.** If your application supports read-only connection strings, you can offload read-only operations to read replicas.

Recommendations

Expand table

Recommendation	Benefits
Use query store to track query performance over time.	Query store simplifies troubleshooting performance problems by helping you quickly find the longest running and most resource-intensive queries.
Use Query Performance Insight to identify the top resource consuming	Query Performance Insight helps you identify long-running queries and their changes over time, determine the wait types that

Recommendation	Benefits
and long-running queries in your workload.	affect them, analyze top database queries by frequency, and other benefits.
Use index tuning with query store to analyze tracked queries and provide recommendations.	Index tuning analyzes the workload that query store tracks. It produces index recommendations to improve the performance of the analyzed workload or to drop duplicate or unused indexes. Index tuning helps you identify beneficial indexes, duplicate indexes, and unused indexes.
Use intelligent tuning to enhance performance automatically and help prevent problems.	Intelligent tuning continuously monitors the Azure Database for PostgreSQL flexible server database's status and dynamically adapts the database to your workload.

Azure policies

Azure provides an extensive set of built-in policies related to Azure Database for PostgreSQL and its dependencies. Some of the preceding recommendations can be audited through Azure Policy. For example, you can check whether:

- Geo-redundant backup is enabled.
- Connection throttling is enabled.
- Secure Sockets Layer (SSL) connections are enforced.
- Data at rest is encrypted with CMKs.
- Private endpoints are enabled.

For comprehensive governance, review the Azure Policy built-in definitions for Azure Database for PostgreSQL and other policies that might affect the security of the data stores.

Azure Advisor recommendations

Azure Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments.

For more information, see Azure Advisor.

Related content

- Multitenancy and Azure Database for PostgreSQL

Architecture best practices for Azure Disk Storage

Article • 12/03/2024

Azure managed disks are a type of Azure Disk Storage that simplify the management of storage for Azure Virtual Machines. Managed disks are block-level storage volumes that Azure manages. They're similar to physical disks in an on-premises server, but they operate in a virtual environment. When you use a managed disk, you must specify the disk size type and configure the disk. After you configure the disk, Azure manages subsequent operations and maintenance tasks.

This article assumes that as an architect, you've reviewed the storage options and chose Azure Disk Storage as the storage service for your workload. The guidance in this article provides architectural recommendations that are mapped to the principles of the Well-Architected Framework pillars.

This guide focuses on how to make decisions about Azure managed disks. But managed disks are a critical dependency of Azure Virtual Machines. As a prerequisite, read and implement the recommendations in Azure Well-Architected Framework perspective on Virtual Machines and scale sets.

> **ⓘ Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern along with design strategies localized to the technology scope.
>
> Also included are recommendations for the technology capabilities that can help materialize those strategies. The recommendations don't represent an exhaustive list of all configurations that are available for Azure Disk Storage and its dependencies. Instead, they list the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept or to optimize your existing environments.
>
> Foundational architecture that demonstrates the key recommendations: Azure Virtual Machines baseline architecture.

Technology scope

This review focuses on the interrelated decisions for the following Azure resources:

- Azure Disk Storage

Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover fast from failures**.

Reliability design principles provide a high-level design strategy applied for individual components, system flows, and the system as a whole.

Design checklist

Start your design strategy based on the design review checklist for Reliability. Determine its relevance to your business requirements while keeping in mind the features and capabilities of Azure Disk Storage. Extend the strategy to include more approaches as needed.

- ✓ **Review best practices to achieve high availability with managed disks.** Optimize your application for high availability by considering these recommendations and how they relate to the configuration of your managed disks and virtual machines (VMs).
- ✓ **Define reliability and recovery targets.** Review the Azure service-level agreements (SLAs). The disk types that you attach to your VM affect the VM SLA. For the highest SLA, only use Azure Ultra Disk Storage, Azure Premium SSD v2, or Premium SSD disks for OS and data disks. For guidance about calculating your reliability targets, see Recommendations for defining reliability targets.
- ✓ **Create a recovery plan.** Evaluate data-protection features, backup and restore operations, and failover procedures. Decide whether to use Azure Backup, Azure Site Recovery, or create your own backup solution by using incremental disk snapshots or restore points. A custom backup solution increases your costs.
- ✓ **Monitor potential availability problems.** Subscribe to the Azure Service Health dashboard. Use disk storage metrics in Azure Monitor to help prevent disk throttling. Manually check VMs to ensure that attached disks don't reach their storage capacity. For guidance about how to integrate these metrics into your overall workload health monitoring strategy, see Health modeling for workloads.
- ✓ **Use failure mode analysis.** Consider internal dependencies, such as the availability of virtual networks or Azure Key Vault, to help minimize points of failure.

Recommendations

Expand table

Recommendation	Benefit
Distribute VMs and disks across multiple availability zones. Use a zone-redundant virtual machine scale set in flexible orchestration mode, or deploy VMs and disks across three availability zones. Use zone balancing to equally spread the instances across zones.	You provision VM and disk instances in physically separate locations within each Azure region. Each location is tolerant to local failures. Depending on resource availability, you might have an uneven number of instances across zones. Zone balancing supports availability by making sure that, if one zone is down, the other zones have sufficient instances. Two instances in each zone provide a buffer during upgrades.
Use Ultra Disk Storage, Premium SSD v2, and Premium SSD disks.	Single-instance VMs that use Premium SSD OS disks and Ultra Disk Storage, Premium SSD v2, or Premium SSD data disks have the highest uptime SLA.
For maximum availability and durability, use a zone-redundant storage (ZRS) disk, especially when you share disks between VMs.	ZRS disks minimize the effect of a failure in an availability zone and increase recoverability from such zonal failures. If a zone fails and your VM remains active, workloads on ZRS disks continue to run. But if an outage does affect your VM, and you want to recover disks before the outage resolves, you can force-detach your ZRS disks from the failed VM. Then the ZRS disks can attach to a different VM. When you share a disk among multiple VMs, use a ZRS disk to prevent the shared disk from becoming a single point of failure.
Implement one of the backup options. For managed solutions, use Backup or Site Recovery. If you need to curate your own backup solution, use restore points or snapshots.	Identify the ideal backup option for your needs to help maximize your environment's recoverability.
If you manage your own snapshots, copy them across regions by using scripts.	Use scripts to simplify transferring data from one region to another. Use this option if you can't use Site Recovery. You can still create disaster recovery backups in other regions when you use this option.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload.

The Security design principles provide a high-level design strategy for achieving those goals by applying approaches to the technical design of Azure Disk Storage.

Design checklist

Start your design strategy based on the design review checklist for Security and identify vulnerabilities and controls to improve your security posture. Extend the strategy to include more approaches as needed.

- ✓ **Limit the ability to export or import managed disks.** Use this approach to increase the security of your data. To limit export or import capabilities, you can use one of these methods:
  - Create a custom role-based access control (RBAC) role that has the permissions necessary to import and export.
  - Use Microsoft Entra ID authentication.
  - Set up private links.
  - Configure an Azure policy.
  - Configure a network access policy.

  For more information, see Restrict managed disks from being imported or exported.

- ✓ **Take advantage of encryption options.** By default, managed disks are encrypted with server-side encryption (SSE), which helps you protect your data and meet organization and compliance commitments. You might need other configurations and options. You can:
  - Use SSE with encryption keys that you manage.
  - Enable encryption at host.
  - Enable double encryption at rest.

  For more information, see Server-side encryption of Azure Disk Storage.

- ✓ **Secure your shared access signature (SAS) with Microsoft Entra ID.** Microsoft Entra ID provides extra security compared to a shared key and SAS, and it's easier to use. Grant security principals only necessary permissions to perform their tasks.

- ✓ **Protect secrets.** Protect secrets, such as customer-managed keys and SAS tokens. We generally don't recommend these forms of authorization. But if you use them,

make sure to rotate your keys, set key expirations as early as practical, and securely store these secrets.

- ✓ **Detect threats.** Enable Microsoft Defender for Cloud so that you can trigger security alerts when anomalies in activity occur. Defender for Cloud notifies subscription administrators by email. The email includes details about the suspicious activity and recommendations to investigate and remediate threats.
- ✓ **Use tags and labels.** Apply tags and labels to important disks to help ensure that you apply the appropriate levels of protection to the disks.
- ✓ **Harden all workload components.** Reduce extraneous surface area and tighten configurations to help reduce the likelihood of attacks. Properly secure any related resources that you use with your managed disks, such as backup recovery vaults or Azure key vaults.

Recommendations

Expand table

Recommendation	Benefit
Use encryption at host for your managed disks whenever possible.	Encryption at host provides end-to-end encryption for environments that have it enabled. Encryption begins on your VM and flows through to its attached disks.
Apply an Azure Resource Manager lock on the disk.	Lock a disk to help prevent it from being deleted so that you don't lose data.
Disable traffic to the public endpoints of your disk. Create private endpoints for clients that run in Azure.	Disable traffic to the public endpoints so that traffic travels over the Microsoft backbone network, which helps eliminate exposure to the public internet.
If possible, use Azure RBAC to limit access to resources and functions.	Use RBAC to avoid tokens or keys that could be compromised. Microsoft Entra ID authenticates the security principal, such as a user, group, managed identity, or service principal. Microsoft Entra ID returns an OAuth 2.0 token. The token is used to authorize a request against the disk service.
Microsoft discourages the use of SAS tokens. If you must create one, review this list of SAS best practices before you create and distribute it.	Best practices can help you prevent a SAS token from being leaked and quickly recover from a leak if one occurs.

Recommendation	Benefit
Set the expiration of SAS tokens to 60 days or less.	
Consider using your own encryption key, or a customer-managed key, to help protect the data in your managed disk.	A customer-managed key provides greater flexibility and control if you need it. For example, you can store encryption keys in Key Vault and automatically rotate them.

Cost Optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget while meeting business requirements.

The Cost Optimization design principles provide a high-level design strategy for achieving those goals and making tradeoffs as necessary in the technical design related to Azure Disk Storage.

Design checklist

Start your design strategy based on the design review checklist for Cost Optimization for investments. Fine-tune the design so that the workload is aligned with the budget that's allocated for the workload. Your design should use the right Azure capabilities, monitor investments, and find opportunities to optimize over time.

- ✓ **Understand how Azure Disk Storage is billed.** Different disk types are billed in different ways and have different features that can affect billing. To design the most cost-optimized environment, see Understand Azure Disk Storage billing. For exact billing, find the specific pricing details and apply the appropriate settings. For more information, see Managed disks pricing.
- ✓ **Estimate the cost of capacity and operations.** Use the pricing calculator to model the costs that are associated with disk types, transactions, and capabilities. Compare the costs that are associated with various regions, account types, namespace types, and redundancy configurations.
- ✓ **Choose a billing model.** Evaluate whether a commitment-based model is more cost efficient than a consumption-based model. If you don't know how much capacity you need, start with a consumption-based model, monitor capacity metrics, and evaluate your choice later.

- ✓ **Decide which features you need.** Some features, such as snapshots or on-demand bursting, incur extra transaction costs, capacity costs, and other charges. For example, if you enable snapshots, you're billed for the amount of storage that each snapshot uses. When you decide which capabilities your disks need, review the pricing and billing details for those capabilities.
- ✓ **Create guardrails.** Create budgets based on subscriptions and resource groups. Use governance policies to restrict resource types, configurations, and locations. You can also use RBAC to block actions that can lead to overspending.
- ✓ **Monitor costs.** to ensure that you stay within budgets, compare costs against forecasts, and see where overspending might have occurred. Use the cost analysis feature in the Azure portal. You also can export cost data to a storage account and use Excel or Power BI to analyze that data.
- ✓ **Monitor disk resources.** Use sample scripts to search for unattached disks.

Recommendations

Expand table

Recommendation	Benefit
Carefully select the appropriate disk types for your workloads. Understand the available disk types and their features before you deploy an environment. Then use the pricing calculator to estimate costs.	One of the best ways to reduce costs is to plan for your requirements, and use the pricing calculator to model the environment.
Use reserved capacity for Premium SSD disks.	Reserved capacity for Premium SSDs reduces the total cost of your environment because you prepay for your capacity at a discount.
Assess whether the features that existing disks offer can improve performance without switching to another disk size or type. Features like disk bursting or changing performance tiers could improve performance to levels that meet your needs.	Depending on your environment and needs, enabling features to improve your disk performance can be more cost effective than switching to a different disk type. These features incur costs but might incur less costs than different disk types.
Directly adjust the performance of your Ultra Disk Storage and Premium SSD v2 disks to fit your performance needs.	These two disk types support a set number of adjustments to the disk's performance within a 24 hour period. This setting allows your workloads to be cost efficient while meeting your performance needs. You can increase performance (increase cost)

Recommendation	Benefit
	to meet higher demand and then lower performance (decrease cost) when the increase is no longer needed. For example, a transaction-intensive database might need a large amount of input/output operations per second (IOPS) at a small size. Or a gaming application might need a large amount of IOPS but only during peak hours.

Operational Excellence

Operational Excellence primarily focuses on procedures for **development practices, observability, and release management**.

The Operational Excellence design principles provide a high-level design strategy for achieving those goals towards the operational requirements of the workload.

Design checklist

Start your design strategy based on the design review checklist for Operational Excellence for defining processes for observability, testing, and deployment related to Azure Disk Storage.

- ✓ **Create maintenance and emergency recovery plans.** Evaluate data-protection features, backup operations, and restore operations. Select backup solutions that you can use to recover from regional disasters.
- ✓ **Create internal documentation.** Document your organization's standard practices. Incorporate existing Azure documentation to streamline your processes. Include documentation about attaching a disk to Windows or Linux VMs or expanding a disk on Windows or Linux VMs.
- ✓ **Detect threats.** Enable Defender for Cloud so that you can trigger security alerts when anomalies in activity occur. Defender for Cloud notifies subscription administrators by email. The email includes details about the suspicious activity and recommendations to investigate and remediate threats.

Recommendations

Expand table

Recommendation	Benefit
Use Azure Monitor to analyze metrics and create alerts.	Azure Monitor provides insight about how your disks and VMs perform. Use these metrics to ensure that your performance remains optimal.
Review the available backup options for managed disks.	Understand the available options so that you can select the configuration that best suits your needs.

Performance Efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance.

The Performance Efficiency design principles provide a high-level design strategy for achieving those capacity goals against the expected usage.

Design checklist

Start your design strategy based on the design review checklist for Performance Efficiency. Define a baseline that's based on key performance indicators for Azure Disk Storage.

- ✓ **Choose optimal disk types.** Identify the disk types that you need before you deploy your resources. This approach helps you maximize performance and cost efficiency. The five disk types include Ultra Disk Storage, Premium SSD v2, Premium SSD, Azure Standard SSD, and Azure Standard HDD. For the highest performance, use Premium SSD for your VM's OS disk, and use Ultra Disk Storage or Premium SSD v2 for your data disks.
- ✓ **Reduce the travel distance between the client and server.** Place data in regions that are closest to connecting clients, ideally in the same region. Default network configurations provide the best performance. Modify network settings only to improve security. In general, network settings don't decrease travel distance and don't improve performance.
- ✓ **Collect performance data.** Monitor your disks and VMs to identify performance bottlenecks that occur from throttling. For more information, see Storage IO metrics.
- ✓ **Benchmark your disks.** Create a test environment and determine whether it meets your needs and expectations. For more information, see Benchmark a disk.

Recommendations

Expand table

Recommendation	Benefit
Create disks in the same region as the VM that you attach them to. If clients from a different region don't require the same data, create a separate disk in each region.	Reduce the physical distance between VMs and their disks, services, and on-premises clients to help improve performance and reduce network latency. This approach also reduces cost for applications that you host in Azure because bandwidth usage within a single region is free.
For workloads and solutions that require the lowest latency, such as e-commerce workloads or databases, use a Premium SSD OS disk and Ultra Disk Storage or Premium SSD v2 data disks.	This configuration offers the best reliability and highest SLA and performance.
Use Azure metrics to monitor your environment and help prevent disk throttling.	Use Azure metrics to identify disks that are being throttled and address them. Throttling leads to suboptimal performance and problems like increased latency.
For disks that are being throttled, evaluate whether changing to a larger disk size or changing to a more performant disk is better for your needs. For Premium SSD disks that are being throttled, if you have short-term bursts of demand, enable on-demand bursting. For longer-term extended demand, change the tier of the disk or evaluate whether Premium SSD v2 or Ultra Disk Storage disks better fit your needs.	Place applications on disks that aren't being throttled to help ensure optimal performance without increased latency.
When you upload a virtual hard disk (VHD), use the Add-AzVHD Azure PowerShell command.	The *Add-AzVHD* Azure PowerShell command automates most of the upload process to help streamline the process.
For existing deployments that are on-premises or in another public cloud provider, use Azure Migrate and Modernize.	Azure Migrate and Modernize can evaluate your deployment and provide curated suggestions for the best sizing of disks and VMs in a prospective Azure deployment.

Azure policies

Azure provides an extensive set of built-in policies related to Azure Disk Storage and its dependencies. Some of the preceding recommendations can be audited through Azure Policy. For example, you can check whether:

- Public network access to your managed disks is disabled.
- Backup is enabled.
- Double encryption is enabled.
- Specific disk encryption sets are used with your disks.
- Customer-managed keys are used.
- Managed disks are zone resilient.
- Notification policies for key expiration are configured.
- Autorotate for customer-managed keys is enabled.

For comprehensive governance, review the Azure Policy built-in definitions for Azure compute and other policies that might impact the security of the storage infrastructure.

Azure Advisor recommendations

Azure Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments. Here are some recommendations that can help you improve the reliability, security, cost effectiveness, performance, and operational excellence of Azure Disk Storage.

- Reliability
- Security
- Cost Optimization
- Performance
- Operational Excellence

Next step

Azure Virtual Machines baseline architecture

Feedback

Was this page helpful? Yes No

Azure Well-Architected Framework perspective on Azure Files

Article • 05/23/2024

Azure Files is a Microsoft file storage solution for the cloud. Azure Files provides server message block (SMB) and network file system (NFS) file shares that you can mount to clients in the cloud, on-premises, or to both. You can also use Azure File Sync to cache SMB file shares on a local Windows server and tier infrequently used files to the cloud.

This article assumes that as an architect, you've reviewed the storage options and chosen Azure Files as the storage service on which to run your workloads. The guidance in this article provides architectural recommendations that are mapped to the principles of the Azure Well-Architected Framework pillars.

> **Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern along with design strategies localized to the technology scope.
>
> Also included are *recommendations* on the technology capabilities that can help implement those strategies. The recommendations don't represent an exhaustive list of all configurations available for Azure Files and its dependencies. Instead, they list the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept or optimize your existing environments.

Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover fast from failures**.

The Reliability design principles provide a high-level design strategy applied for individual components, workloads, system flows, and the system as a whole.

Design checklist

Start your design strategy based on the design review checklist for Reliability.

- ✓ **Use failure mode analysis**: Minimize points of failure by considering internal dependencies such as the availability of virtual networks, Azure Key Vault, or Azure Content Delivery Network or Azure Front Door endpoints. Failures can occur if you need credentials to access Azure Files, and the credentials go missing from Key Vault. Or you might have a failure if your workloads use an endpoint that's based on a missing content delivery network. In these cases, you might need to configure your workloads to connect to an alternative endpoint. For general information about failure mode analysis, see Recommendations for performing failure mode analysis.

- ✓ **Define reliability and recovery targets**: Review the Azure service-level agreements (SLAs). Derive the service-level objective (SLO) for the storage account. For example, the redundancy configuration that you chose might affect the SLO. Consider the effect of a regional outage, the potential for data loss, and the time required to restore access after an outage. Also consider the availability of internal dependencies that you identified as part of your failure mode analysis.

- ✓ **Configure data redundancy**: For maximum durability, choose a configuration that copies data across availability zones or global regions. For maximum availability, choose a configuration that allows clients to read data from the secondary region during an outage of the primary region.

- ✓ **Design applications**: Design your applications to seamlessly shift so that they read data from a secondary region if the primary region is unavailable. This design consideration only applies to geo-redundant storage (GRS) and geo-zone-redundant storage (GZRS) configurations. Design your applications to properly handle outages, which reduces downtime for customers.

- ✓ **Explore features to help you meet your recovery targets**: Make files restorable so that you can recover corrupted, edited, or deleted files.

- ✓ **Create a recovery plan**: Consider data protection features, backup and restore operations, or failover procedures. Prepare for potential data loss and data inconsistencies and the time and cost of failing over. For more information, see Recommendations for designing a disaster recovery strategy.

- ✓ **Monitor potential availability problems**: Subscribe to the Azure Service Health dashboard to monitor potential availability problems. Use storage metrics and diagnostic logs in Azure Monitor to investigate alerts.

Recommendations

Expand table

Recommendation	Benefit
Configure your storage account for redundancy. For maximum availability and durability, configure your account with zone-redundant storage (ZRS), GRS, or GZRS. Limited Azure regions support ZRS for standard and premium file shares. Only standard SMB accounts support GRS and GZRS. Premium SMB shares and NFS shares don't support GRS and GZRS. Azure Files doesn't support read-access geo-redundant storage (RA-GRS) or read-access geo-zone-redundant storage (RA-GZRS). If you configure a storage account to use RA-GRS or RA-GZRS, the file shares are configured and billed as GRS or GZRS.	Redundancy protects your data against unexpected failures. The ZRS and GZRS configuration options replicate across various availability zones and enable applications to continue reading data during an outage. For more information, see Durability and availability by outage scenario and Durability and availability parameters.
Before you initiate a failover or failback, check the value of the last synchronization time property to evaluate the potential for data loss. This recommendation applies only to GRS and GZRS configurations.	This property helps you estimate how much data you might lose if you initiate an account failover. All data and metadata that's written before the last synchronization time is available on the secondary region, but you might lose data and metadata that's written after the last synchronization time because it's not written to the secondary region.
As a part of your backup and recovery strategy, enable soft delete and use snapshots for point-in-time restore. You can use Azure Backup to back up your SMB file shares. You can also use Azure File Sync to back up on-premises SMB file shares to an Azure file share.	Soft delete works on a file share level to protect Azure file shares against accidental deletion. Point-in-time restore protects against accidental deletion or corruption because you can restore file shares to an earlier state. For more information, see Data protection overview.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload.

The Security design principles provide a high-level design strategy for achieving those goals by applying approaches to the technical design of your file storage configuration.

Security requirements and recommendations vary depending on whether your workload uses the SMB or NFS protocol to access your file shares. So the following sections have separate design checklists and recommendations for SMB and NFS file shares.

As a best practice, you should keep SMB and NFS file shares in separate storage accounts because they have different security requirements. Use this approach to provide your workload with strong security and high flexibility.

Design checklist for SMB file shares

Start your design strategy based on the design review checklist for Security. Identify vulnerabilities and controls to improve the security posture. Extend the strategy to include more approaches as needed.

- ✓ **Review the security baseline for Azure Storage**: To get started, review the security baseline for Storage.
- ✓ **Consider using network controls to restrict ingress and egress traffic**: You might be comfortable exposing your storage account to the public internet under certain conditions, like if you use identity-based authentication to grant access to file shares. But we recommend that you use network controls to grant the minimum required level of access to users and applications. For more information, see How to approach network security for your storage account.
- ✓ **Reduce the attack surface**: Use encryption in transit and prevent access over non-secure (HTTP) connections to reduce the attack surface. Require clients to send and receive data by using the latest version of the Transport Layer Security (TLS) protocol.
- ✓ **Minimize the use of storage account keys**: Identity-based authentication provides superior security compared to using a storage account key. But you must use a storage account key to get full administrative control of a file share, including the ability to take ownership of a file. Grant security principals only the necessary permissions that they need to perform their tasks.
- ✓ **Protect sensitive information**: Protect sensitive information, such as storage account keys and passwords. We don't recommend that you use these forms of authorization, but if you do, you should make sure to rotate, expire, and store them securely.

- ✓ **Detect threats**: Enable Microsoft Defender for Storage to detect potentially harmful attempts to access or exploit your Azure file shares over SMB or FileREST protocols. Subscription administrators get email alerts with details of suspicious activity and recommendations about how to investigate and remediate threats. Defender for Storage doesn't support antivirus capabilities for Azure file shares. If you use Defender for Storage, transaction-heavy file shares incur significant costs, so consider opting out of Defender for Storage for specific storage accounts.

Recommendations for SMB file shares

Expand table

Recommendation	Benefit
Apply an Azure Resource Manager lock on the storage account.	Lock the account to prevent accidental or malicious deletion of the storage account, which can cause data loss.
Open TCP port 445 outbound or set up a VPN gateway or Azure ExpressRoute connection for clients outside of Azure to access the file share.	SMB 3.x is an internet-safe protocol, but you might not have the ability to change organizational or ISP policies. You can use a VPN gateway or an ExpressRoute connection as an alternative option.
If you open port 445, be sure to disable SMBv1 on Windows and Linux clients. Azure Files doesn't support SMB 1, but you should still disable it on your clients.	SMB 1 is an outdated, inefficient, and insecure protocol. Disable it on clients to improve your security posture.
Consider disabling public network access to your storage account. Enable public network access only if SMB clients and services that are external to Azure require access to your storage account. If you disable public network access,create a private endpoint for your storage account. Standard data processing rates for private endpoints apply. A private endpoint doesn't block connections to the public endpoint. You should still disable public network access as previously described. If you don't require a static IP address for your file share and want to avoid the cost of private endpoints, you can instead restrict public endpoint access to specific virtual networks and IP addresses.	Network traffic travels over the Microsoft backbone network instead of the public internet, which eliminates risk exposure from the public internet.

Recommendation	Benefit
Enable firewall rules that limit access to specific virtual networks. Start with zero access, and then methodically and incrementally provide the least amount of access required for clients and services.	Minimize the risk of creating openings for attackers.
When possible, use identity-based authentication with AES-256 Kerberos ticket encryption to authorize access to SMB Azure file shares.	Use identity-based authentication to decrease the possibility of an attacker using a storage account key to access file shares.
If you use storage account keys, store them in Key Vault, and make sure to regenerate them periodically. You can completely disallow storage account key access to the file share by removing NTLMv2 from the share's SMB security settings. But you generally shouldn't remove NTLMv2 from the share's SMB security settings because administrators still need to use the account key for some tasks.	Use Key Vault to retrieve keys at runtime instead of saving them with your application. Key Vault also makes it easy to rotate your keys without interruption to your applications. Periodically rotate the account keys to reduce the risk of exposing your data to malicious attacks.
In most cases, you should enable the *Secure transfer required* option on all your storage accounts to enable encryption in transit for SMB file shares. Don't enable this option if you need to allow very old clients to access the share. If you disable secure transfer, be sure to use network controls to restrict traffic.	This setting ensures that all requests that are made against the storage account take place over secure connections (HTTPS). Any requests made over HTTP will fail.
Configure your storage account so that TLS 1.2 is the minimum version for clients to send and receive data.	TLS 1.2 is more secure and faster than TLS 1.0 and 1.1, which don't support modern cryptographic algorithms and cipher suites.
Use only the most recent supported SMB protocol version (currently 3.1.1.), and use only AES-256-GCM for SMB channel encryption. Azure Files exposes settings that you can use to toggle the SMB protocol and make it more compatible or more secure, depending on your organization's requirements. By default, all SMB versions are allowed. However, SMB 2.1 is disallowed if you enable *Require secure transfer* because SMB 2.1 doesn't support encryption of data in transit. If you restrict these settings to a high level of	SMB 3.1.1, released with Windows 10, contains important security and performance updates. AES-256-GCM offers more secure channel encryption.

Recommendation	Benefit
security, some clients might not be able to connect to the file share.	

Design checklist for NFS file shares

- ✓ **Review the security baseline for Storage**: To get started, review the security baseline for Storage.
- ✓ **Understand your organization's security requirements**: NFS Azure file shares only support Linux clients that use the NFSv4.1 protocol, with support for most features from the 4.1 protocol specification. Some security features, such as Kerberos authentication, access control lists (ACLs), and encryption in transit, aren't supported.
- ✓ **Use network-level security and controls to restrict ingress and egress traffic**: Identity-based authentication isn't available for NFS Azure file shares, so you must use network-level security and controls to grant the minimum required level of access to users and applications. For more information, see How to approach network security for your storage account.

Recommendations for NFS file shares

Expand table

Recommendation	Benefit
Apply a Resource Manager lock on the storage account.	Lock the account to prevent accidental or malicious deletion of the storage account, which might cause data loss.
You must open port 2049 on the clients that you want to mount your NFS share to.	Open port 2049 to let clients communicate with the NFS Azure file share.
NFS Azure file shares are only accessible through restricted networks. So you must create a private endpoint for your storage account or restrict public endpoint access to selected virtual networks and IP addresses. We recommend that you create a private endpoint. You must configure network-level security for NFS shares because Azure Files doesn't support encryption in transit with the NFS protocol. You need to disable the *Require*	Network traffic travels over the Microsoft backbone network instead of the public internet, which eliminates risk exposure from the public internet.

Recommendation	Benefit
secure transfer setting on the storage account to use NFS Azure file shares. Standard data processing rates apply for private endpoints. If you don't require a static IP address for your file share and want to avoid the cost of private endpoints, you can restrict public endpoint access instead.	
Consider disallowing storage account key access at the storage account level. You don't need this access to mount NFS file shares. But keep in mind that full administrative control of a file share, including the ability to take ownership of a file, requires use of a storage account key.	Disallow the use of storage account keys to make your storage account more secure.

Cost Optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget while meeting business requirements.

The Cost Optimization design principles provide a high-level design strategy for achieving those goals and making tradeoffs as necessary in the technical design related to file storage and its environment.

Design checklist

Start your design strategy based on the design review checklist for Cost Optimization for investments. Fine tune the design so that the workload is aligned with the budget that's allocated for the workload. Your design should use the right Azure capabilities, monitor investments, and find opportunities to optimize over time.

- ✓ **Decide whether your workload requires the performance of premium file shares (Azure Premium SSD) or if Azure Standard HDD storage is sufficient**: Determine your storage account type and billing model based on the type of storage that you need. If you require large amounts of input/output operations per second (IOPS), extremely fast data transfer speeds, or very low latency, then you should choose premium Azure file shares. NFS Azure file shares are only available on the premium tier. NFS and SMB file shares are the same price on the premium tier.
- ✓ **Create a storage account for your file share, and choose a redundancy level**: Choose either a standard (GPv2) or premium (FileStorage) account. The redundancy level that you choose affects cost. The more redundancy, the higher the cost.

Locally redundant storage (LRS) is the most affordable. GRS is only available for standard SMB file shares. Standard file shares only show transaction information at the storage account level, so we recommend that you deploy only one file share in each storage account to ensure full billing visibility.

- ✓ **Understand how your bill is calculated**: Standard Azure file shares provide a pay-as-you-go model. Premium shares use a provisioned model in which you specify and pay for a certain amount of capacity, IOPS, and throughput up front. In the pay-as-you-go model, meters track the amount of data that's stored in the account, or the capacity, and the number and type of transactions based on your usage of that data. The pay-as-you-go model can be cost efficient because you pay only for what you use. With the pay-as-you-go model, you don't need to overprovision or deprovision storage based on performance requirements or demand fluctuations.

  But you might find it difficult to plan storage as part of a budgeting process because end-user consumption drives cost. With the provisioned model, transactions don't affect billing, so costs are easy to predict. But you pay for the provisioned storage capacity whether you use it or not. For a detailed breakdown of how costs are calculated, see Understand Azure Files billing.

- ✓ **Estimate the cost of capacity and operations**: You can use the Azure pricing calculator to model the costs associated with data storage, ingress, and egress. Compare the cost associated with various regions, account types, and redundancy configurations. For more information, see Azure Files pricing.

- ✓ **Choose the most cost-effective access tier**: Standard SMB Azure file shares offer three access tiers: *transaction optimized*, *hot*, and *cool*. All three tiers are stored on the same standard storage hardware. The main difference for these three tiers is their data at rest storage prices, which are lower in cooler tiers, and the transaction prices, which are higher in cooler tiers. For more information, see Differences in standard tiers.

- ✓ **Decide which value-added services you need**: Azure Files supports integrations with value-added services such as Backup, Azure File Sync, and Defender for Storage. These solutions have their own licensing and product costs but are often considered part of the total cost of ownership for file storage. Consider other cost aspects if you use Azure File Sync.

- ✓ **Create guardrails**: Create budgets based on subscriptions and resource groups. Use governance policies to restrict resource types, configurations, and locations. Additionally, use role-based access control (RBAC) to block actions that can lead to overspending.

- ✓ **Monitor costs**: Ensure costs stay within budgets, compare costs against forecasts, and see where overspending occurs. You can use the cost analysis pane in the Azure portal to monitor costs. You can also export cost data to a storage account, and use Excel or Power BI to analyze that data.
- ✓ **Monitor usage**: Continuously monitor usage patterns to detect unused or underused storage accounts and file shares. Check for unexpected increases in capacity, which might indicate that you're collecting numerous log files or soft-deleted files. Develop a strategy for deleting files or moving files to more cost-effective access tiers.

Recommendations

Expand table

Recommendation	Benefit
When you migrate to standard Azure file shares, we recommend that you start in the *transaction-optimized* tier during the initial migration. Transaction usage during migration isn't typically indicative of normal transaction usage. This consideration doesn't apply for premium file shares because the provisioned billing model doesn't charge for transactions.	Migrating to Azure Files is a temporary, transaction-heavy workload. Optimize the price for high-transaction workloads to help reduce migration costs.
After you migrate your workload, if you use standard file shares, carefully choose the most cost effective access tier for your file share: *hot*, *cool*, or *transaction optimized*. After you operate for a few days or weeks with regular usage, you can insert your transaction counts in the pricing calculator to figure out which tier best suits your workload. Most customers should choose *cool* even if they actively use the share. But you should examine each share and compare the balance of storage capacity to transactions to determine your tier. If transaction costs make up a significant percentage of your bill, the savings from using the *cool* access tier often offsets this cost and minimizes the total overall cost. We recommend that you move standard file shares between access tiers only when necessary to optimize for changes in your workload pattern. Each move incurs transactions. For more information, see Switching between standard tiers.	Select the appropriate access tier for standard file shares to considerably reduce your costs.

Recommendation	Benefit
If you use premium shares, ensure that you provision more than enough capacity and performance for your workload but not so much that you incur unnecessary costs. We recommend overprovisioning by two to three times. You can dynamically scale premium file shares up or down depending on your storage and input/output (IO) performance characteristics.	Overprovision premium file shares by a reasonable amount to help maintain performance and account for future growth and performance requirements.
Use Azure Files reservations, also referred to as reserved instances, to precommit to storage usage and get a discount. Use reservations for production workloads or dev/test workloads with consistent footprints. For more information, see Optimize costs with storage reservations. Reservations don't include transaction, bandwidth, data transfer, and metadata storage charges.	Three-year reservations can provide a discount up to 36% on the total cost of file storage. Reservations don't affect performance.
Monitor snapshot usage. Snapshots incur charges, but they're billed based on the differential storage usage of each snapshot. You pay only for the difference in each snapshot. For more information, see Snapshots. Azure File Sync takes share-level and file-level snapshots as part of regular usage, which can increase your total Azure Files bill.	Differential snapshots ensure that you're not billed multiple times for storing the same data. However, you should still monitor snapshot usage to help reduce your Azure Files bill.
Set retention periods for the soft-delete feature, especially when you first start using it. Consider starting with a short retention period to better understand how the feature affects your bill. The minimum recommended retention period is seven days. When you soft delete standard and premium file shares, they're billed as used capacity rather than provisioned capacity. And premium file shares are billed at the snapshot rate while in the soft-delete state. Standard file shares are billed at the regular rate while in the soft-delete state.	Set a retention period so that soft-deleted files don't pile up and increase the cost of capacity. After the configured retention period, permanently deleted data doesn't incur cost.

Operational Excellence

Operational Excellence primarily focuses on procedures for **development practices, observability, and release management**.

The Operational Excellence design principles provide a high-level design strategy for achieving those goals for the operational requirements of the workload.

Design checklist

Start your design strategy based on the design review checklist for Operational Excellence for defining processes for observability, testing, and deployment related to your file storage configuration.

- ✓ **Create maintenance and emergency recovery plans**: Consider data protection features, backup and restore operations, and failover procedures. Prepare for potential data loss and data inconsistencies and the time and cost of failing over.
- ✓ **Monitor the health of your storage account**: Create Storage insights dashboards to monitor availability, performance, and resiliency metrics. Set up alerts to identify and address problems in your system before your customers notice them. Use diagnostic settings to route resource logs to an Azure Monitor Logs workspace. Then you can query logs to investigate alerts more deeply.
- ✓ **Periodically review file share activity**: Share activity can change over time. Move standard file shares to cooler access tiers, or you can provision or deprovision capacity for premium shares. When you move standard file shares to a different access tier, you incur a transaction charge. Move standard file shares only when needed to reduce your monthly bill.

Recommendations

Expand table

Recommendation	Benefit
Use infrastructure as code (IaC) to define the details of your storage accounts in Azure Resource Manager templates (ARM templates), Bicep, or Terraform.	You can use your existing DevOps processes to deploy new storage accounts, and use Azure Policy to enforce their configuration.
Use Storage insights to track the health and performance of your storage accounts. Storage insights provides a unified view of the failures, performance, availability, and capacity for all your storage accounts.	You can track the health and operation of each of your accounts. Easily create dashboards and reports that stakeholders can use to track the health of your storage accounts.
Use Monitor to analyze metrics, such as availability, latency, and usage, and to create alerts.	Monitor provides a view of availability, performance, and resiliency for your file shares.

Performance Efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance.

The Performance Efficiency design principles provide a high-level design strategy for achieving those capacity goals against the expected usage.

Design checklist

Start your design strategy based on the design review checklist for Performance Efficiency. Define a baseline that's based on key performance indicators for your file storage configuration.

- ✓ **Plan for scale**: Understand the scalability and performance targets for storage accounts, Azure Files, and Azure File Sync.
- ✓ **Understand your application and usage patterns to achieve predictable performance**: Determine latency sensitivity, IOPS and throughput requirements, workload duration and frequency, and workload parallelization. Use Azure Files for multi-threaded applications to help you achieve the upper performance limits of a service. If most of your requests are metadata-centric, such as createfile, openfile, closefile, queryinfo, or querydirectory, the requests create poor latency that's higher than the read and write operations. If you have this problem, consider separating the file share into multiple file shares within the same storage account.
- ✓ **Choose the optimal storage account type**: If your workload requires large amounts of IOPS, extremely fast data transfer speeds, or very low latency, then you should choose premium (FileStorage) storage accounts. You can use a standard general-purpose v2 account for most SMB file share workloads. The primary tradeoff between the two storage account types is cost versus performance.

  The provisioned share size, such as IOPS, egress, and ingress, and single-file limits determine premium share performance. For more information, see Understanding provisioning for premium file shares. Premium file shares also offer burst credits as an insurance policy if you need to temporarily exceed a premium file share's baseline IOPS limit.
- ✓ **Create storage accounts in the same regions as connecting clients to reduce latency**: The farther you are from the Azure Files service, the greater the latency and the more difficult to achieve performance scale limits. This consideration is especially true when you access Azure Files from on-premises environments. If possible, ensure that your storage account and your clients are co-located in the same Azure region. Optimize for on-premises clients by minimizing network latency

or by using an ExpressRoute connection to extend on-premises networks into the Microsoft cloud over a private connection.

- ✓ **Collect performance data**: Monitor workload performance, including latency, availability, and usage metrics. Analyze logs to diagnose problems such as timeouts and throttling. Create alerts to notify you if a file share is being throttled, about to be throttled, or experiencing high latency.

- ✓ **Optimize for hybrid deployments**: If you use Azure File Sync, sync performance depends on many factors: your Windows Server and the underlying disk configuration, network bandwidth between the server and the Azure storage, file size, total dataset size, and the activity on the dataset. To measure the performance of a solution that's based on Azure File Sync, determine the number of objects, such as files and directories, that you process per second.

Recommendations

Expand table

Recommendation	Benefit
Enable SMB Multichannel for premium SMB file shares. SMB Multichannel allows an SMB 3.1.1 client to establish multiple network connections to an SMB Azure file share. SMB Multichannel only works when the feature is enabled on both client-side (your client) and service-side (Azure). On Windows clients, SMB Multichannel is enabled by default, but you need to enable it on your storage account.	Increase throughput and IOPS while reducing the total cost of ownership. Performance benefits increase with the number of files that distribute load.
Use the nconnect client-side mount option with NFS Azure file shares on Linux clients. Nconnect enables you to use more TCP connections between the client and the Azure Files premium service for NFSv4.1.	Increase performance at scale, and reduce the total cost of ownership for NFS file shares.
Make sure your file share or storage account isn't being throttled, which can result in high latency, low throughput, or low IOPS. Requests are throttled when the IOPS, ingress, or egress limits are reached. For standard storage accounts, throttling occurs at the account level. For premium file shares, throttling usually occurs at the share level.	Avoid throttling to provide the best possible client experience.

Azure policies

Azure provides an extensive set of built-in policies related to Azure Files. Some of the preceding recommendations can be audited through Azure policies. For example, you can check whether:

- Only requests from secure connections, such as HTTPS, are accepted.
- Shared key authorization is disabled.
- Network firewall rules are applied to the account.
- Diagnostic settings for Azure Files are set to stream resource logs to an Azure Monitor Logs workspace.
- Public network access is disabled.
- Azure File Sync is configured with private endpoints to use private DNS zones.

For comprehensive governance, review the Azure Policy built-in definitions for storage and other policies that might affect the security of the compute layer.

Azure Advisor recommendations

Azure Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments. Here are some recommendations that can help you improve the reliability, security, cost effectiveness, performance, and operational excellence of Azure Files.

- Reliability
- Security
- Cost Optimization
- Performance
- Operational Excellence

Next step

For more information, see Azure Files documentation.

Feedback

Was this page helpful? Yes No

Architecture best practices for Azure Front Door

Article • 02/26/2025

Azure Front Door is a global load balancer and content delivery network (CDN) that routes HTTP and HTTPS traffic. Azure Front Door delivers and distributes traffic that's closest to the application users.

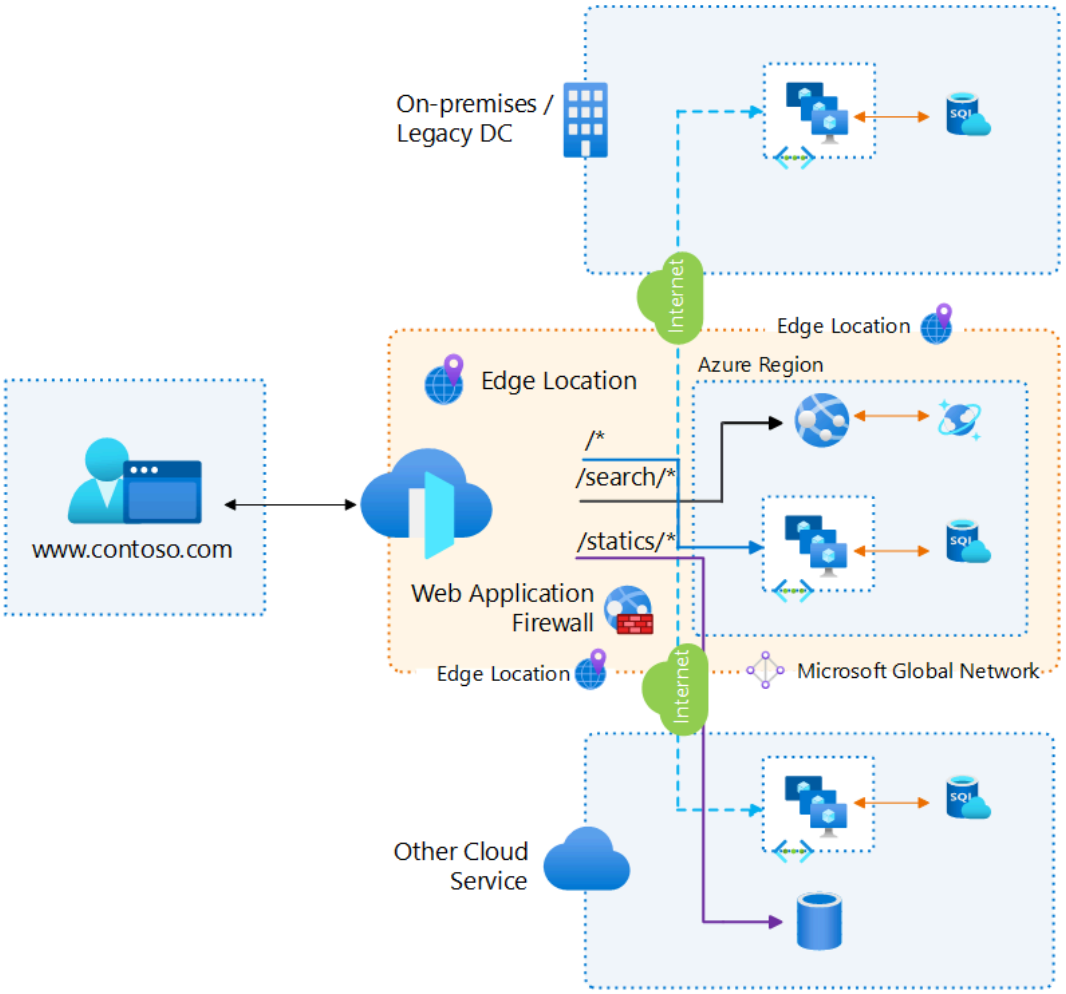
This article assumes that as an architect you've reviewed the load balancing options and chosen Azure Front Door as the load balancer for your workload. It also assumes that your application is deployed to multiple regions in an active-active or active-passive model. The guidance in this article provides architectural recommendations that are mapped to the principles of the Azure Well-Architected Framework pillars.

> **ⓘ Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern and design strategies that are localized to the technology scope.
>
> This article also includes *recommendations* on the technology capabilities that help materialize those strategies. The recommendations don't represent an exhaustive list of all configurations available for Azure Front Door and its dependencies. Instead, they list the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept or optimize your existing environments.
>
> Foundational architecture that demonstrates the key recommendations: **Mission-critical baseline architecture with network controls**.

Technology scope

This review focuses on the interrelated decisions for the following Azure resources:

- Azure Front Door



Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover fast from failures**.

The Reliability design principles provide a high-level design strategy applied for individual components, system flows, and the system as a whole.

Design checklist

Start your design strategy based on the design review checklist for Reliability. Determine its relevance to your business requirements while keeping in mind the tiers and CDN capabilities. Extend the strategy to include more approaches as needed.

- ✓ **Choose your deployment strategy**. The fundamental deployment approaches are *active-active* and *active-passive*. Active-active deployment means that multiple environments or stamps that run the workload serve traffic. Active-passive deployment means that only the primary region handles all traffic, but it fails over to the secondary region when necessary. In a multiregion deployment, stamps or

application instances run in different regions for higher availability with a global load balancer, like Azure Front Door, that distributes traffic. Therefore, it's important to configure the load balancer for the appropriate deployment approach.

Azure Front Door supports several routing methods, which you can configure to distribute traffic in an active-active or active-passive model.

The preceding models have many variations. For example, you can deploy the active-passive model with a warm spare. In this case, the secondary hosted service deploys with the minimum possible compute and data sizing and runs without load. Upon failover, the compute and data resources scale to handle the load from the primary region. For more information, see Key design strategies for multiregion design.

Some applications need the user connections to stay on the same origin server during the user session. From a reliability perspective, we don't recommend keeping user connections on the same origin server. Avoid session affinity as much as possible.

- ✓ **Use the same host name on each layer**. To ensure that cookies or redirect URLs work properly, preserve the original HTTP host name when you use a reverse proxy, like Azure Front Door, in front of a web application.

- ✓ **Implement the health endpoint monitoring pattern**. Your application should expose health endpoints, which aggregate the state of the critical services and dependencies that your application needs to serve requests. Azure Front Door health probes use the endpoint to detect origin servers' health. For more information, see Health Endpoint Monitoring pattern.

- ✓ **Cache static content**. The content delivery feature of Azure Front Door has hundreds of edge locations and can help withstand traffic surges and distributed denial of service (DDoS) attacks. These capabilities help improve reliability.

- ✓ **Consider a redundant traffic management option**. Azure Front Door is a globally distributed service that runs as a singleton in an environment. Azure Front Door is a potential single point of failure in the system. If the service fails, then clients can't access your application during the downtime.

  Redundant implementations can be complex and costly. Consider them only for mission-critical workloads that have a very low tolerance to outage. Carefully consider the tradeoffs.
  - If you absolutely need redundant routing, see Global routing redundancy.
  - If you need redundancy only to serve cached content, see Global content delivery.

Recommendations

Expand table

Recommendation	Benefit
Choose a routing method that supports your deployment strategy. The weighted method, which distributes traffic based on the configured weight coefficient, supports active-active models. A priority-based value that configures the primary region to receive all traffic and send traffic to the secondary region as a backup supports active-passive models. Combine the preceding methods with latency sensitivity configurations so that the origin with the lowest latency receives traffic.	You can select the best origin resource by using a series of decision steps and your design. The selected origin serves traffic within the allowable latency range in the specified ratio of weights.
Support redundancy by having multiple origins in one or more origin groups. Always have redundant instances of your application and make sure each instance exposes an origin. You can place those origins in one or more origin groups.	Multiple origins support redundancy by distributing traffic across multiple instances of the application. If one instance is unavailable, then other origins can still receive traffic.
Set up health probes on the origin. Configure Azure Front Door to conduct health checks to determine if the origin instance is available and ready to continue receiving requests. For more information, see Best practices on health probes.	Enabled health probes are part of the health monitoring pattern implementation. Health probes make sure that Azure Front Door only routes traffic to instances that are healthy enough to handle requests.
Set a timeout on forwarding requests to the origin, and avoid long-running requests. Adjust the timeout setting according to your endpoints' needs. If you don't, Azure Front Door might close the connection before the origin sends the response. You can also lower the default timeout for Azure Front Door if all of your origins have a shorter timeout. For more information, see Troubleshooting unresponsive requests.	Long-running requests consume system resources. Timeouts help prevent performance issues and availability issues by terminating requests that take longer than expected to complete.
Use the same host name on Azure Front Door and your origin. Azure Front Door can rewrite the host header of incoming	Set the same host name to prevent malfunction with session

Recommendation	Benefit
requests, which is useful when you have multiple custom domain names that route to one origin. However, rewriting the host header might cause issues with request cookies and URL redirection. For more information, see [Preserve the original HTTP host name](/azure/architecture/best-practices/host-name-preservation between a reverse proxy and its origin web application).	affinity, authentication, and authorization.
Decide if your application requires session affinity. If you have high reliability requirements, we recommend that you disable session affinity.	With session affinity, user connections stay on the same origin during the user session. In some situations, a single origin might become overloaded with requests while other origins are idle. If that origin becomes unavailable, the user experience might be disrupted.
Take advantage of the rate-limiting rules that are included with a web application firewall (WAF).	Limit requests to prevent clients from sending too much traffic to your application. Rate limiting can help you avoid problems like a retry storm.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload.

The **Security design principles** provide a high-level design strategy for achieving those goals by applying approaches to the technical design in restricting traffic coming through Azure Front Door.

Design checklist

Start your design strategy based on the design review checklist for Security. Identify vulnerabilities and controls to improve the security posture. Extend the strategy to include more approaches as needed.

- ✓ **Review the security baseline for Azure Front Door.**
- ✓ **Protect the origin servers**. Azure Front Door is the front end, and is the single point of ingress to the application.

Azure Front Door uses Azure Private Link to access an application's origin. Private Link creates segmentation so that the origins don't need to expose public IP addresses and endpoints. For more information, see Secure your origin with Private Link in Azure Front Door Premium.

Configure your back-end services to accept only requests with the same host name that Azure Front Door uses externally.

- ✓ **Allow only authorized access to the control plane**. Use Azure Front Door role-based access control (RBAC) to restrict access to only the identities that need it.
- ✓ **Block common threats at the edge**. WAF is integrated with Azure Front Door. Enable WAF rules on the front ends to protect applications from common exploits and vulnerabilities at the network edge, closer to the attack source. Consider geo-filtering to restrict access to your web application by countries or regions.

  For more information, see Azure Web Application Firewall on Azure Front Door.
- ✓ **Protect against unexpected traffic**. The architecture of Azure Front Door provides built-in DDoS protection to protect application endpoints from DDoS attacks. If you need to expose other public IP addresses from your application, consider adding the Azure DDoS Protection standard plan for those addresses for advanced protection and detection capabilities.

  There are also WAF rule sets that detect bot traffic or unexpectedly large volumes of traffic that could potentially be malicious.
- ✓ **Protect data in transit**. Enable end-to-end Transport Layer Security (TLS), HTTP to HTTPS redirection, and managed TLS certificates when applicable. For more information, see TLS best practices for Azure Front Door.
- ✓ **Monitor anomalous activity**. Regularly review the logs to check for attacks and false positives. Send WAF logs from Azure Front Door to your organization's centralized security information and event management (SIEM), such as Microsoft Sentinel, to detect threat patterns and incorporate preventative measures in the workload design.

Recommendations

Expand table

Recommendation	Benefit
Enable WAF rule sets that detect and block potentially malicious traffic. This feature is available on the Premium tier. We recommend these rule sets: - Default - Bot protection - IP restriction - Geo-filtering - Rate limiting	Default rule sets are updated frequently based on OWASP top-10 attack types and information from Microsoft Threat Intelligence. The specialized rule sets detect certain use cases. For example, bot rules classify bots as good, bad, or unknown based on the client IP addresses. They also block bad bots and known IP addresses and restrict traffic based on geographical location of the callers. By using a combination of rule sets, you can detect and block attacks with various intents.
Create exclusions for managed rule sets. Test a WAF policy in detection mode for a few weeks and adjust any false positives before you deploy it.	Reduce false positives and allow legitimate requests for your application.
Send the host header to the origin.	The back-end services should be aware of the host name so that they can create rules to accept traffic only from that host.
Secure the connections from Azure Front Door to your origins. Enable Private Link connectivity to supported origins. If your origin doesn't support Private Link connectivity, use service tags and the `X-Azure-FDID` header to verify the source of the request is your Azure Front Door profile.	Ensure that all traffic flows through Azure Front Door, and gets the security benefits such as DDoS protection and WAF inspection.
Enable end-to-end TLS, HTTP to HTTPS redirection, and managed TLS certificates when applicable. Review the TLS best practices for Azure Front Door. Use TLS version 1.2 as the minimum allowed version with ciphers that are relevant for your application. Azure Front Door managed certificates should be your default choice for ease of operations. However, if you want to manage the lifecycle of the certificates, use your own certificates in Azure Front Door custom domain endpoints and store them in Key Vault.	TLS ensures that data exchanges between the browser, Azure Front Door, and the origins are encrypted to prevent tampering. Key Vault offers managed certificate support and simple certificate renewal and rotation.

Cost Optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget while meeting business requirements.

The Cost Optimization design principles provide a high-level design strategy for achieving those goals and making tradeoffs as necessary in the technical design related to Azure Front Door and its environment.

Design checklist

Start your design strategy based on the design review checklist for Cost Optimization for investments. Fine-tune the design so that the workload is aligned with the budget that's allocated for the workload. Your design should use the right Azure capabilities, monitor investments, and find opportunities to optimize over time.

- ✓ **Review service tiers and pricing**. Use the pricing calculator to estimate the realistic costs for each tier of Azure Front Door. Compare the features and suitability of each tier for your scenario. For instance, only the Premium tier supports connecting to your origin via Private Link.

  The Standard SKU is more cost-effective and suitable for moderate traffic scenarios. In the Premium SKU, you pay a higher unit rate, but you gain access to security benefits and advanced features like managed rules in WAF and Private Link. Consider the tradeoffs on reliability and security based on your business requirements.

- ✓ **Consider bandwidth costs**. The bandwidth costs of Azure Front Door depend on the tier that you choose and the type of data transfer. To learn about Azure Front Door billing, see Understand Azure Front Door billing.

  Azure Front Door provides built-in reports for billable metrics. To assess your costs related to bandwidth and where you can focus your optimization efforts, see Azure Front Door reports.

- ✓ **Optimize incoming requests**. Azure Front Door bills the incoming requests. You can set restrictions in your design configuration.

  Reduce the number of requests by using design patterns like Backend for Frontends and Gateway Aggregation. These patterns can improve the efficiency of your operations.

WAF rules restrict incoming traffic, which can optimize costs. For example, use rate limiting to prevent abnormally high levels of traffic, or use geo-filtering to allow access only from specific regions or countries.

- ✓ **Use resources efficiently**. Azure Front Door uses a routing method that helps with resource optimization. Unless the workload is extremely latency sensitive, distribute traffic evenly across all environments to effectively use deployed resources.

  Azure Front Door endpoints can serve many files. One way to reduce bandwidth costs is to use compression.

  Use caching in Azure Front Door for content that doesn't change often. Because content is served from a cache, you save on bandwidth costs that are incurred when the request is forwarded to the origins.

- ✓ **Consider using a shared instance that's provided by the organization**. Costs incurred from centralized services are shared between the workloads. However, consider the tradeoff with reliability. For mission-critical applications that have high availability requirements, we recommend an autonomous instance.

- ✓ **Pay attention to the amount of data logged**. Costs related to both bandwidth and storage can accrue if certain requests aren't necessary or if logging data is retained for a long period of time.

Recommendations

Expand table

Recommendation	Benefit
Use caching for endpoints that support it.	Caching optimizes data transfer costs because it reduces the number of calls from your Azure Front Door instance to the origin.
Consider enabling file compression. For this configuration, the application must support compression and caching must be enabled.	Compression reduces bandwidth consumption and improves performance.
Disable health checks in origin groups with a single origin. If you have only one origin configured in your Azure Front Door origin group, these calls are unnecessary.	You can save on bandwidth costs by disabling health check requests that aren't required to make routing decisions.

Operational Excellence

Operational Excellence primarily focuses on procedures for **development practices, observability, and release management**.

The Operational Excellence design principles provide a high-level design strategy for achieving those goals for the operational requirements of the workload.

Design checklist

Start your design strategy based on the design review checklist for Operational Excellence for defining processes for observability, testing, and deployment related to Azure Front Door.

- ✓ **Use infrastructure as code (IaC) technologies**. Use IaC technologies like Bicep and Azure Resource Manager templates to provision the Azure Front Door instance. These declarative approaches provide consistency and straightforward maintenance. For example, by using IaC technologies, you can easily adopt new ruleset versions. Always use the latest API version.

- ✓ **Simplify configurations**. Use Azure Front Door to easily manage configurations. For example, suppose your architecture supports microservices. Azure Front Door supports redirection capabilities, so you can use path-based redirection to target individual services. Another use case is the configuration of wildcard domains.

- ✓ **Handle progressive exposure**. Azure Front Door provides multiple routing methods. For a weighted load balancing approach you can use a canary deployment to send a specific percentage of traffic to an origin. This approach helps you test new features and releases in a controlled environment before you roll them out.

- ✓ **Collect and analyze operational data as part of your workload monitoring**. Capture relevant Azure Front Door logs and metrics with Azure Monitor Logs. This data helps you troubleshoot, understand user behaviors, and optimize operations.

- ✓ **Offload certificate management to Azure**. Ease the operational burden associated with certification rotation and renewals.

Recommendations

Expand table

Recommendation	Benefit
Use HTTP to HTTPS redirection to support forward compatibility.	When redirection is enabled, Azure Front Door automatically redirects clients that are using older protocol to use HTTPS for a secure experience.
Capture logs and metrics. Include resource activity logs, access logs, health probe logs, and WAF logs. Set up alerts.	Monitoring ingress flow is a crucial part of monitoring an application. You want to track requests and make performance and security improvements. You need data to debug your Azure Front Door configuration. With alerts in place, you can get instant notifications of any critical operational issues.
Review the built-in analytics reports.	A holistic view of your Azure Front Door profile helps drive improvements based on traffic and security reports through WAF metrics.
Use managed TLS certificates when possible.	Azure Front Door can issue and manage certificates for you. This feature eliminates the need for certificate renewals and minimizes the risk of an outage due to an invalid or expired TLS certificate.
Use wildcard TLS certificates.	You don't need to modify the configuration to add or specify each subdomain separately.

Performance Efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance.

The Performance Efficiency design principles provide a high-level design strategy for achieving those capacity goals against the expected usage.

Design checklist

Start your design strategy based on the design review checklist for Performance Efficiency. Define a baseline that's based on key performance indicators for Azure Front Door.

- ✓ **Plan capacity by analyzing your expected traffic patterns**. Conduct thorough testing to understand how your application performs under different loads. Consider factors like simultaneous transactions, request rates, and data transfer.

Base your SKU choices on that planning. The Standard SKU is more cost-effective and suitable for moderate traffic scenarios. If you anticipate higher loads, we recommend the Premium SKU.

- ✓ **Analyze performance data by regularly reviewing performance metrics**. Azure Front Door reports provide insights into various metrics that serve as performance indicators at the technology level.

  Use Azure Front Door reports to set realistic performance targets for your workload. Consider factors like response times, throughput, and error rates. Align the targets with your business requirements and user expectations.

- ✓ **Optimize data transfers.**

  - Use caching to reduce latency in serving static content, such as images, stylesheets, and JavaScript files, or content that doesn't change frequently.

    Optimize your application for caching. Use cache expiration headers in the application that control how long the content should be cached by clients and proxies. Longer cache validity means less frequent requests to the origin server, which results in reduced traffic and lower latency.

  - Reduce the size of files that are transmitted over the network. Smaller files lead to faster load times and improved user experience.

  - Minimize the number of back-end requests in the application.

    For example, a web page displays user profiles, recent orders, balances, and other related information. Instead of making separate requests for each set of information, use design patterns to structure your application so that multiple requests are aggregated into a single request.

    Update clients to use the HTTP/2 protocol, which can combine multiple requests into a single TCP connection.

    Use WebSockets to support realtime full-duplex communication, rather than making repeated HTTP requests or polling.

    By aggregating requests, you send less data between the front end and the back end and establish fewer connections between the client and the back end, which reduces overhead. Also, Azure Front Door handles fewer requests, which prevents overload.

- ✓ **Optimize the use of health probes**. Get health information from health probes only when the state of the origins change. Strike a balance between monitoring accuracy

and minimizing unnecessary traffic.

Health probes are typically used to assess the health of multiple origins within a group. If you have only one origin configured in your Azure Front Door origin group, disable health probes to reduce unnecessary traffic on your origin server. Because there's only one instance, the health probe status won't impact routing.

- ✓ **Review the origin routing method**. Azure Front Door provides various routing methods, including latency-based, priority-based, weighted, and session affinity-based routing, to the origin. These methods significantly affect your application's performance. To learn more about the best traffic routing option for your scenario, see Traffic routing methods to origin.

- ✓ **Review the location of origin servers**. Your origin servers' location impacts the responsiveness of your application. Origin servers should be closer to the users. Azure Front Door ensures that users from a specific location access the nearest Azure Front Door entry point. The performance benefits include faster user experience, better use of latency-based routing by Azure Front Door, and minimized data transfer time by using caching, which stores content closer to users.

  For more information, see Traffic by location report.

Recommendations

Expand table

Recommendation	Benefit
Enable caching. You can optimize query strings for caching. For purely static content, ignore query strings to maximize your use of the cache. If your application uses query strings, consider including them in the cache key. Including the query strings in the cache key allows Azure Front Door to serve cached responses or other responses, based on your configuration.	Azure Front Door offers a robust content delivery network solution that caches content at the edge of the network. Caching reduces the load on the origin servers and reduces data movement across the network, which helps offload bandwidth usage.
Use file compression when you're accessing downloadable content.	Compression in Azure Front Door helps deliver content in the optimal format, has a smaller payload, and delivers content to the users faster.

Recommendation	Benefit
When you configure health probes in Azure Front Door, consider using `HEAD` requests instead of `GET` requests. The health probe reads only the status code, not the content.	`HEAD` requests let you query a state change without fetching its entire content.
Evaluate whether you should enable session affinity when requests from the same user should be directed to the same origin server. From a reliability perspective, we don't recommend this approach. If you use this option, the application should gracefully recover without disrupting user sessions. There's also a tradeoff on load balancing because it restricts the flexibility of distributing traffic across multiple origins evenly.	Optimize performance and maintain continuity for user sessions, especially when applications rely on maintaining state information locally.

Azure policies

Azure provides an extensive set of built-in policies related to Azure Front Door and its dependencies. Some of the preceding recommendations can be audited through Azure Policies. For example, you can check whether:

- You need the Premium tier to support managed WAF rules and Private Link in Azure Front Door profiles.
- You need to use the minimum TLS version, which is version 1.2.
- You need secure, private connectivity between Azure Front Door Premium and Azure PaaS services.
- You need to enable resource logs. WAF should have request body inspection enabled.
- You need to use policies to enforce the WAF rule set. For example, you should enable bot protection and turn on rate-limiting rules.

For comprehensive governance, review the built-in definitions for Azure Content Delivery Network and other Azure Front Door policies that are listed in Azure Policy built-in policy definitions.

Azure Advisor recommendations

Azure Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments. Advisor recommendations are aligned with Well-Architected Framework pillars.

For more information, see the recommendations in Azure Advisor.

Next steps

Consider the following articles as resources that demonstrate the recommendations highlighted in this article.

- Use the following reference architectures as examples of how you can apply this article's guidance to a workload:
  - Mission-critical baseline with network controls
- Build implementation expertise by using the following product documentation:
  - Azure Front Door
  - Best practices for Azure Front Door

Feedback

Was this page helpful? Yes No

Architecture best practices for Azure IoT Hub

Article • 02/28/2025

Azure IoT Hub is a managed cloud-based service that serves as a central message hub for communication between an IoT application and its connected devices. You can reliably connect almost any device and its back-end solutions to an IoT hub at scale.

This article assumes that as an architect, you've reviewed the technologies and services that you can use to create IoT solutions and chose IoT Hub as the Azure IoT platform service for your workload. The guidance in this article provides architectural recommendations that are mapped to the principles of the Well-Architected Framework pillars.

> **ⓘ Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern along with design strategies localized to the technology scope.
>
> Also included are recommendations for the technology capabilities that can help materialize those strategies. The recommendations don't represent an exhaustive list of all configurations that are available for IoT Hub and its dependencies. Instead, they list the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept or to optimize your existing environments.

Technology scope

This review focuses on the interrelated decisions for the following Azure resources:

- Azure IoT Hub
- Azure IoT Hub Device Provisioning Service (DPS)
- Azure Device Update for IoT Hub
- Azure IoT Edge
- Azure Digital Twins
- Azure Sphere
- Microsoft Defender for IoT

Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover fast from failures.**

Reliability design principles provide a high-level design strategy applied for individual components, system flows, and the system as a whole.

Design checklist

Start your design strategy based on the design review checklist for Reliability. Determine its relevance to your business requirements while keeping in mind the performance of IoT Hub. Extend the strategy to include more approaches as needed.

- ✓ **Design devices for resiliency.** Design your devices to satisfy the uptime and availability requirements of your end-to-end solution. Ensure that your IoT device can operate efficiently with intermittent connectivity to the cloud.
- ✓ **Design for business requirements.** Consider the cost implications of introducing architectural modifications to meet service-level agreements. For example, to increase reliability and high availability, you might implement cross-region redundancies and an autoscale system. Carefully consider the tradeoffs.
- ✓ **Implement safe, simple update procedures.** Create a strategy for operators to manage devices in your enterprise IoT solution. IoT operators require simple and reliable update tools and practices.
- ✓ **Observe application health.** Define service-level indicators (SLIs) and service-level objectives (SLOs) based on observability. Add processes for auditing, monitoring, and alerting in addition to the processes included in IoT Hub.
- ✓ **Implement high availability and disaster recovery for critical components.** Plan for resilient hardware and software components that increase redundancy, including cross-region redundancies.
- ✓ **Plan for capacity.** Plan for service quotas and throttles, and account for latency that occurs between detection and action. Establish benchmarks at the production scale to support uninterrupted data flow.

Recommendations

Expand table

Recommendation	Benefit
Calculate your required IoT Hub capacity, such as the number of messages per day and other limits. If your workload has fluctuating capacity requirements, implement an autoscaling mechanism so that it increases and decreases capacity based on demand. For more information, see Choose the right IoT Hub tier and size for your solution.	Dynamic scaling helps optimize resource usage. It helps ensure that your solution remains responsive and reliable without overprovisioning resources.
Add a device reconnection strategy to IoT Hub to design resilient applications. For more information, see Manage device reconnections.	IoT devices often rely on intermittent or unstable network connections. A reconnection strategy enables your workload to recover without user intervention, which increases availability.
Evaluate the tradeoffs of different high availability and disaster recovery options in IoT Hub. Depending on your IoT solution's uptime goals, determine the options that best suit your business objectives. Choose between Microsoft-initiated failover, manual failover, and cross-region high availability.	A failover strategy helps the workload quickly switch to a backup instance, which minimizes the impact on your users and business operations.
Use DPS to provision your devices and assign them to an IoT hub.	DPS is a helper service that enables zero-touch, just-in-time provisioning without requiring human intervention. DPS enables the provisioning of millions of devices in a highly secure and scalable manner.
Use Device Update for IoT Hub to manage over-the-air updates for your IoT devices. Ensure that your update strategy includes gradual rollout, resilient A/B updates, detailed management, and reporting tools.	This approach helps ensure safe, secure, and reliable updates for IoT devices, which reduce downtime and improve operational efficiency. It helps maintain device compliance and quickly identifies and resolves update failures, which enhances overall system reliability.
Implement DevOps practices to manage IoT solutions, including continuous integration and continuous deployment (CI/CD), monitoring, and automated updates. - Use DevOps to build and release your IoT Edge applications. - Use Azure Monitor to monitor and get alerts from the data that IoT Hub collects. - Use automatic device management in IoT Hub to automate the management of devices at scale.	DevOps practices enhance the reliability and efficiency of IoT solutions. They enable rapid deployment, continuous monitoring, and automated updates. These features reduce downtime, improve system performance, and deploy updates and changes safely and consistently.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload.

The Security design principles provide a high-level design strategy for achieving those goals by applying approaches to the technical design of IoT Hub.

Design checklist

Start your design strategy based on the design review checklist for Security and identify vulnerabilities and controls to improve the security posture. Extend the strategy to include more approaches as needed.

- ✓ **Use a strong identity to authenticate devices and users.** Implement practices to enhance security. For example, you should:
  - Have a hardware root of trust for trusted identity.
  - Register devices.
  - Issue renewable credentials.
  - Use passwordless or multi-factor authentication (MFA).
- ✓ **Automate and use least-privileged access control.** Limit the impact of compromised devices or identities, or unapproved workloads.
- ✓ **Evaluate your device health.** Evaluate your device health to control access or identify devices for remediation. Check security configurations, assess vulnerabilities and insecure passwords, monitor for threats and anomalies, and build ongoing risk profiles.
- ✓ **Implement device updates.** Implement continuous updates to keep devices healthy. Use a centralized configuration and compliance management solution and a robust update mechanism to ensure that devices are up-to-date and healthy.
- ✓ **Monitor system security, and plan incident response.** Proactively monitor for unauthorized or compromised devices, and respond to emerging threats.
- ✓ **Ensure highly secure connections.** Ensure that all data that's transmitted between the IoT device and the IoT cloud services is confidential and tamperproof.

Recommendations

Expand table

Recommendation	Benefit
Apply Zero Trust criteria for devices. Devices that connect to IoT Hub should: - Contain a hardware security module for strong identity. - Use renewable credentials. - Enforce least-privileged access control. - Emit proper health signals for conditional access. - Alert operators to revoke a device from IoT Hub when compromised. Your devices should include: - Update agents for security updates. - Device management capabilities for cloud-driven configuration and automated security response. - Only necessary features enabled to minimize the physical attack footprint. - Data protection at rest via standard encryption algorithms.	Zero Trust criteria for devices that connect to IoT Hub enhances security and reliability. Hardware security modules, renewable credentials, and least-privileged access control minimize unauthorized access and compromised devices. Health signals for conditional access and update agents help ensure that devices remain secure and compliant. Cloud-driven device management and automated security response, along with security agents, strengthen the security posture. A small physical attack footprint and data protection at rest safeguard sensitive information and help maintain system integrity.
Use X.509 certificates to authenticate your devices to IoT Hub. For more information, see Authenticate identities with X.509 certificates.	X.509-based authentication in production environments provides greater security compared to security tokens.
Layer least-privileged access for IoT devices. Use network segmentation to group IoT devices. This practice mitigates the impact of a potential compromise. For example, you can connect IoT devices to an "IoT network" for devices, such as printers, VoIP phones, and smart TVs. Keep this IoT network separate from other organizational resources that your workforce accesses. Use network micro-segmentation to logically separate dedicated operations technology environments from the corporate information technology network. To create this separation, use a perimeter network architecture that includes firewalls. Mature organizations can also implement micro-segmentation policies at multiple layers of the Purdue model, typically by using next-generation firewalls.	Network segmentation groups IoT devices, which can mitigate the impact of a compromise. Network micro-segmentation isolates less-capable devices at the network layer, either behind a gateway or on a discrete network segment.
Use Microsoft Defender for IoT as the frontline of defense to help protect your resources in Azure.	Defender for IoT provides continuous monitoring and threat detection to quickly

Recommendation	Benefit
Defender for IoT is an agentless, network layer security platform that delivers continuous asset discovery, vulnerability management, and threat detection for IoT devices.	identify unauthorized or compromised devices. It enhances the overall security and resilience of the IoT network.
Use IoT Hub with Azure Sphere as a guardian module to help secure other devices, including existing legacy systems that you can't rely on for trusted connectivity. An Azure Sphere guardian module deploys with an application and interfaces with existing devices through Ethernet, serial, or Bluetooth Low Energy (BLE). The devices don't necessarily have direct internet connectivity.	The Azure Sphere guardian module provides a way to implement highly secure connectivity in existing devices without exposing those devices to the internet. This protection includes encrypted data transmission, highly secure OS and application updates, and authentication to ensure communication only with trusted hosts.
Use IoT Edge gateways to enforce strong identity patterns for less-capable devices. IoT Edge provides an edge runtime connection to IoT Hub and supports certificates as strong device identities. IoT Edge supports the PKCS#11 standard for device manufacturing identities and other secrets on a Trusted Platform Module (TPM) or hardware security module.	IoT Edge gateways help enforce strong identity patterns so that only authenticated and authorized devices can communicate within the network. This feature enhances the security and integrity of your IoT ecosystem.

Cost Optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget while meeting business requirements.

The Cost Optimization design principles provide a high-level design strategy for achieving those goals and making tradeoffs as necessary in the technical design related to IoT Hub and its environment.

Design checklist

Start your design strategy based on the design review checklist for Cost Optimization for investments. Fine-tune the design so that the workload is aligned with the budget that's allocated for the workload. Your design should use the right Azure capabilities, monitor investments, and find opportunities to optimize over time.

- ✓ **Develop cost-management discipline.** To understand the total cost of ownership (TCO), account for both functional and nonfunctional costs in your plan.
- ✓ **Use industry-standard strategies and approaches.** For IoT-specific industries like manufacturing, use industry-standard strategies and approaches to build structures, IoT systems, smart cities, or energy grids. This approach helps optimize costs.
- ✓ **Understand the cost to protect your IoT solution.** IoT covers the entire process, from hardware to cloud services, so you incur security-related expenses at every stage. For example, you can incur costs for microprocessor units (MPUs), devices, telecommunication, cloud technology, and operation technology.
- ✓ **Design for rate optimization.** Define implementation plans for each IoT solution-wide concern to optimize costs.
- ✓ **Monitor and optimize over time.** Continuously monitor and optimize IoT Hub costs. Do ongoing cost optimization activities after you implement your solution.
- ✓ **Estimate realistic costs.** Select the appropriate IoT Hub tier and size for the development and operational phases of your IoT solution.
- ✓ **Evaluate one-time costs versus recurring costs.** Consider the cost benefits of one-time costs versus recurring costs. For example, hacking techniques continuously evolve, so you might use a reliable commercial operating system and module such as Azure Sphere. For a one-time payment, such services provide ongoing monthly device security patches.
- ✓ **Optimize resource usage.** Implement autoscaling mechanisms to adjust the IoT Hub capacity based on demand. This approach helps ensure efficient resource usage and cost control.
- ✓ **Use reserved capacity.** Evaluate the size and frequency of message payloads to ensure that your IoT Hub is appropriately sized and prepared to scale.
- ✓ **Implement cost alerts and budgets.** Set up cost alerts and budgets in Microsoft Cost Management to effectively track and control your spending. Use cost management APIs for IoT Hub and DPS.

Recommendations

Expand table

Recommendation	Benefit
Understand IoT Hub quotas and throttling limits and how they impact the TCO when your solution runs at scale in production. Your cost model should account for devices, infrastructure, operations, and monitoring at scale in production.	Each service has its own set of quotas and throttling limits. Understand these limits in IoT Hub to optimize costs.
Understand the device ecosystem and the various roles involved, such as hardware manufacturers, app developers, and operators for planning. Devices can range from small 8-bit MCUs to advanced x86 CPUs, like those found in desktop computers. For more information, see Best practices for device configuration within an IoT solution. The hardware that you choose depends on many variables. Understand the key hardware differentiators in an IoT device to reduce costs.	Understanding the range of devices, and the key hardware differentiators, facilitates allocating resources more effectively. This ensures that you are using the right devices for the right tasks, leading to better performance and cost efficiency.
Use IoT Plug and Play to reduce hardware development and on-boarding time. Adopt a Digital Twin Definition Language (DTDL) industry ontology, which is a model that a device uses to advertise its capabilities to an IoT Plug and Play-enabled application.	Solution builders can use IoT Plug and Play to integrate IoT devices with IoT Hub without manual configuration.
Use Device Update for IoT Hub to deliver over-the-air updates and remotely manage your devices.	A solution to remotely update the device's firmware or software helps reduce long-term manual labor costs.
Use connectors for Azure services, such as Power Apps, Power Automate, and Azure Logic Apps. IoT Hub and Azure Event Grid together provide connectors to implement your requirements within Logic Apps and Power Automate.	Connectors enable seamless integration between IoT devices and various Azure services, which enables automated workflows and enhanced data processing capabilities.
Use device simulators and set up a load testing environment to test the solution at production scale and understand the solution cost. Simulation code samples like the Azure IoT Device Telemetry Simulator help you test and estimate costs at scale with various parameters.	Device simulators can help save costs at scale to model, test, and simulate outcomes for a scenario.
To reduce transmission sizes and costs: - Choose the right protocol for your IoT devices.	An optimal protocol for your scenario enables devices to reduce transmission

Recommendation	Benefit
- Compress telemetry at the edge. - Batch messages on the device. - Choose between keeping connections alive or reconnecting when devices wake up. - Use the device twin feature to exchange status information asynchronously if cost is critical. For battery-powered IoT devices, you can choose between keeping connections alive or reconnecting when the devices wake up. Use keep-alive messages, or heartbeat messages, to check the device status, but consider the extra network costs of transmission. Reconnecting uses about 6 KB of data to establish a TLS connection, authenticate the device, and retrieve a device twin. But it conserves battery capacity if the device wakes up only once or twice a day.	sizes and costs in the transport layer.
Understand how to use hot, warm, and cold path analytics for IoT data, and apply the lambda architecture. Use the built-in message routing feature in IoT Hub.	IoT solutions can store large amounts of data. Storage costs make up a large part of the overall solution cost. Choose an appropriate processing and storage plan based on your business scenario.

Operational Excellence

Operational Excellence primarily focuses on procedures for **development practices, observability, and release management.**

The Operational Excellence design principles provide a high-level design strategy for achieving those goals for the operational requirements of the workload.

Design checklist

Start your design strategy based on the design review checklist for Operational Excellence for defining processes for observability, testing, and deployment related to IoT Hub.

- ✓ **Embrace continuous operations and scaling.** Ensure that the IoT solution can:
  - Successfully manage automated device provisioning.
  - Integrate with other back-end systems.
  - Support different roles, such as solution developers, solution administrators, and operators.

- Adapt and scale efficiently to changes on demand, such as newly deployed IoT devices or higher ingestion throughput.

- ✓ **Optimize build and release processes.** A successful enterprise IoT solution requires a strategy to establish and update a device or a fleet of device's configuration. A device's configuration includes device properties, connection settings, relationships, and firmware. IoT operators require simple and reliable tools to update the configuration of a device or fleet of devices at any point during the device's lifetime.

- ✓ **Understand operational health.** Use IoT solution logging, monitoring, and alerting systems to determine if the solution is functioning as expected and to help troubleshoot problems throughout the lifecycle of the solution.

- ✓ **Use automation and DevOps.** An IoT device is essentially a small computer that has specialized hardware and software. IoT devices are often constrained in hardware. For example, they might have limited memory or compute capacity. Automation and DevOps help ensure the proper upload and deployment of OS and software to IoT devices and gateways, which minimize operational downtime. Use automation and DevOps to monitor and manage the lifecycle of IoT devices.

Recommendations

Expand table

Recommendation	Benefit
Use IoT Hub automatic device management or IoT Edge automatic deployments to implement continuous updates to existing or new devices and to IoT Edge device configurations, such as properties, application specific settings, or relationships. To update an existing device or IoT Edge device configuration based on a one-time or recurring schedule, use IoT Hub scheduled jobs. To update existing device or IoT Edge device firmware, application, or package updates over-the-air, use Device Update for IoT Hub. Have a manual update method for IoT devices. Because of root certificate changes or connectivity problems, you might need to	IoT Hub automatic device management and IoT Edge automatic deployments provide a highly efficient, secure, and reliable way to automate configuration deployments for a fleet or specific group of devices. To ensure that the devices have the specified configurations, the services continuously monitor new and existing targeted devices and their configurations based on tags. IoT Hub scheduled jobs provide an optimal way to update configurations for a fleet or specific group of devices at a scheduled time. Device Update for IoT Hub efficiently updates a fleet or specific group of devices.

Recommendation	Benefit
manually update devices by physically connecting to a local computer or by using a local connectivity protocol, such as BLE.	
Configure the ingestion and other back-end layers of the IoT cloud solution to handle expected and unexpected capacity needs efficiently. If your solution is linked to a connected product, ensure that it can handle fluctuations in expected load within IoT Hub and the associated back-end layers.	Marketing initiatives, such as sales or promotions, and seasonal events, such as holidays, can introduce peaks in load. To ensure that your solution can scale to handle the peaks, test load variations for expected and unexpected events.
Build a centralized management UI by using the REST APIs exposed in IoT Hub REST APIs to help operation teams manage device fleets.	A centralized device management solution streamlines the administration, monitoring, and operation of IoT devices, which helps ensure efficient lifecycle management and consistent configuration across the IoT solution. An integrated UI also helps operation teams manage device fleets effectively, which reduces operational complexity and improves overall system reliability.
Use a centralized identity provider, such as Microsoft Entra ID, to create managed identities. Only allow the appropriate users in those roles to perform management or operation activities, such as creating and provisioning new devices, sending commands to hardware in the field, deploying updates, and modifying user permissions.	Microsoft Entra ID for authentication provides improved security and ease of use compared to traditional security tokens. You can use Azure role-based access control (RBAC) in Microsoft Entra ID to control access to IoT Hub.
In an IoT Hub-based solution, you can use Microsoft Entra ID to authenticate requests to IoT Hub service APIs, such as creating device identities or invoking direct methods. You can develop a custom management UI for solution operators and administrators. This UI authenticates users against Microsoft Entra ID and sends API requests to the IoT solution back end on their behalf.	Custom management UIs improve security by incorporating RBAC and other security measures. They also enhance user experience by providing intuitive interfaces that simplify complex tasks.
Use IoT Hub logging, monitoring, and alerting systems to determine if the solution is functioning as expected and to help troubleshoot and mitigate problems. For more information about the metrics and logs that IoT Hub creates, see IoT Hub monitoring data reference.	Monitoring and logging help determine if devices or systems are experiencing errors, correctly configured, generating accurate data, and meeting defined SLOs.

Recommendation	Benefit
Use CI/CD DevOps principles and processes to boost productivity and create a seamless rapid development cycle. Adopt DevOps for your IoT Edge applications by using the built-in IoT Edge tasks in Azure Pipelines. For more information, see CI/CD to IoT Edge devices.	DevOps tools and processes in IoT Hub and IoT Edge help automate the edge software lifecycle.
Define a process to reprovision and deprovision IoT devices. For more information. see IoT Hub device reprovisioning concepts. The IoT device lifecycle includes defining the procedures to reprovision existing devices to other locations or purposes and securely deprovisioning them when necessary.	A reprovision and deprovision process for IoT devices helps you manage their lifecycle. These processes help maintain continuity of operations when you migrate device state information and configurations between IoT hubs. This practice helps ensure that devices continue to function correctly after the move.
Test the failover and failback of IoT Hub to help ensure high availability. Document recovery steps for Microsoft-initiated failover and manual failover. Codify and automate the steps needed to recover or fail over the application to a secondary Azure region during failures. This practice helps ensure that you can effectively respond to outages and minimize impact. Similarly, codify and automate the steps to failback the application to the primary region after the problem is addressed.	Failover and failback procedure tests help ensure that your IoT solution can handle failures and maintain high availability, which minimizes the impact of failures. Documented recovery steps for both Microsoft-initiated and manual failovers provide a clear and structured approach to handle failover scenarios, which ensures that team members are aware of the procedures and can implement them efficiently.
Use an infrastructure as code (IaC) language, such as Bicep or Azure Resource Manager templates (ARM templates), to define and store all IoT Hub, DPS, and back-end infrastructure configurations. Define your entire infrastructure as IaC so that you can automatically deploy your resources consistently across environments. Use identity and access management (IAM) or governance tooling, such as Azure RBAC or Azure Policy, to control write access to your infrastructure. Restrict configuration setting or infrastructure updates to an automated pipeline. Use this approach so that you can see what changes occur in your environments and when.	A repeatable and predictable process to provision and configure your IoT solution resources, such as IoT Hub or DPS, helps reduce errors and downtime. DevOps tools, such as Azure DevOps or GitHub, can help track configuration, IaC, and firmware versions. Version tracking helps you identify the versions of firmware, configuration, and IaC in each environment.
Create test environments that use the same firmware, configuration settings, and IaC as your	Matching test and production environments help simplify the development and testing of

Recommendation	Benefit
production environments. To easily create these environments, use IaC and automate your processes as much as possible.	new features and hot fixes.

Performance Efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance.

The Performance Efficiency design principles provide a high-level design strategy for achieving those capacity goals against the expected usage.

Design checklist

Start your design strategy based on the design review checklist for Performance Efficiency for defining a baseline based on key performance indicators for IoT Hub.

- ✓ **Design for horizontal scaling.** An IoT solution can start with a few hundred devices or messages and grow to millions of devices and messages per minute. IoT Hub and related cloud services can easily handle increased loads, but IoT devices and gateways add complexity. You can design and deploy IoT devices before you finalize the solution. Industries like industrial IoT have devices that last several years, often decades. Replacing devices to update capacity can incur significant costs, so you must plan ahead.
- ✓ **Test performance early.** Test early and test often to quickly identify problems. Understand variables that can introduce complexity, such as sensors, devices, and gateways in geographically different locations with different characteristics, speed, and reliability of communication. Plan for this complexity in your testing. Test for failure scenarios like network disconnection. Do stress and load testing of all device, edge, and cloud components in your IoT Hub and related cloud services.
- ✓ **Continuously monitor for performance in production.** To monitor different types of devices in multiple geographical regions, use a distributed monitoring solution. Balance memory and performance costs against the amount of information that you monitor and send to the cloud. Tune transmission for diagnostic scenarios, and monitor at multiple levels and layers. Expose gateway metrics for industrial or gateway-enabled solutions.

Recommendations

Expand table

Recommendation	Benefit
Optimize hardware capabilities. Hardware upgrades and replacements can increase cost and time. Size IoT devices for your required capacity and functionality in advance. To optimize existing hardware capabilities, use efficient languages and frameworks like Embedded C and Rust Embedded. When you develop for constrained devices, or when most of the security and communication stack is available on the device, use the Azure IoT Embedded C SDK.	Compute and input-output intensive tasks that run on specific hardware can significantly enhance performance. For example, machine learning algorithms that run on local GPUs can offload processing from the central processor, which creates faster and more efficient computations. Efficient languages and frameworks can improve performance and reduce resource consumption. The Azure IoT device SDK for C efficiently manages all necessary components so that you can connect to the cloud gateway. The Azure IoT device SDKs handle required message translation, error handling, and retry mechanisms for a resilient connection, which optimizes the use of existing hardware capabilities.
Consider running some workloads at the edge, depending on system constraints such as network throughput or latency.	Workloads at the edge reduce latency and improve responsiveness because they process data closer to where it's generated. This approach minimizes the need for constant cloud connectivity, which benefits scenarios that have intermittent or limited network access. It also helps offload processing from the cloud, which reduces bandwidth usage and associated costs.
Don't connect all devices at once, for example after a regional power outage.	When you do a retry, use truncated exponential backoff that introduces jitter. This approach spreads out the reconnection attempts over time, which helps prevent network congestion and server overload. It creates a more stable and reliable reconnection process, which reduces the risk of other outages or performance degradation. Properly managed reconnection attempts can help maintain the overall health and performance of your IoT Hub.
Optimize offline scenarios. Provide devices with enough information and context to work without a cloud connection and to store data locally so they can recover from disconnections and reboots. Use device	A device that can store data locally, including logs and cached telemetry according to priority, when not connected helps maintain functionality during disconnections. Discard less-important data when the device isn't connected to reduce local storage

Recommendation	Benefit
twins and module twins to asynchronously sync state information between devices and the cloud for devices that have intermittent connections to IoT Hub. Set a time to live (TTL) on the data to ensure that expired data is removed automatically. This practice reduces the need for manual intervention. If edge device storage reaches capacity, use a cache eviction strategy, such as a first-in last-out, last-in first-out, or priority-based strategy, to help manage storage efficiently. Consider using a separate disk or disk controller to store data so that the device runtime or application can continue to work even if it has low storage.	requirements and synchronization time when the device reconnects. A separate disk or disk controller to store data helps ensure that storage limitations don't interrupt critical operations.
Optimize messaging efficiency. IoT Hub calculates daily quota message counts based on a 4-KB message size. Sending smaller messages leaves some capacity unused. To optimize usage, aim for message sizes close to 4 KB. To reduce the total number of messages, group smaller device-to-cloud messages into larger messages. But be aware of the introduced latency when combining messages. To implement application-level batching, combine multiple smaller messages at the downstream device and send larger messages to the edge gateway. Use direct methods for request-reply interactions that can succeed or fail immediately, after a user-specified time-out. Use this approach for scenarios where the course of action differs depending on whether the device responded. Use device twins for device state information, including metadata and configurations. IoT Hub maintains a device twin for each device that you connect.	The number and size of device-to-cloud messages is an important parameter for IoT solution performance efficiency. IoT Hub defines message limits per tier, which affects the solution performance and cost. Batching helps limit the message overhead and reduce writes to local edge disk storage.

Recommendation	Benefit
Understand messaging quotas and throttling. The IoT Hub tier sets limits for each cloud gateway unit. The messaging quota defines the sustained throughput and sustained send rates for the tier. IoT Hub can handle loads that exceed these quotas for short durations to resiliently handle bursts or load overshoots. Another important limit is the hourly or daily service load or throttle limit. Throttle limits protect an IoT hub from excessive load over extended periods.	Understand messaging quotas and throttling to help ensure that your IoT solution operates within the defined limits. This practice prevents overload and maintains performance. Manage load effectively so that you can handle bursts and load overshoots without affecting overall system stability. This approach helps maintain a reliable and efficient IoT solution and prevents potential service disruptions because of excessive load.
Optimize message processing. Optimize the format that you use to send data to the cloud. Weigh the benefits of optimizing data formats and reducing cloud processing against the bandwidth costs. Consider using IoT Hub message enrichment to add context to device messages. Do time-critical event processing on ingested data as it arrives, instead of storing unprocessed data and requiring complex queries to acquire the data. For time-critical event processing, consider the effects of late data arrival and windowing. Evaluate your approach based on the use case, such as handling critical alarms versus enriching messages. Select the Basic or Standard IoT Hub tier based on your solution requirements. Understand features that the Basic tier doesn't support. Consider using Event Grid for publish-subscribe event routing. For more information, see React to IoT Hub events by using Event Grid to trigger actions and Compare message routing and Event Grid for IoT Hub.	Before storage, you might need to translate, process, or enrich messages from a device or gateway with more information. This step can be time consuming, so you should evaluate the effect on performance. Some recommendations conflict, such as using compression for optimizing data transfer versus avoiding cloud processing in decrypting messages. Balance and evaluate these recommendations against other architectural pillars and solution requirements.
Use IoT Edge priority queues to prioritize important data that you send to IoT Hub. IoT Edge buffers messages when there's no connectivity. After the connection is	Some data that devices send to the cloud might be more important than other data. Classify and handle the data based on priority to improve performance efficiency. For example, a thermostat

Recommendation	Benefit
restored, it sends all buffered messages in priority order first, followed by new messages. Use IoT Hub message routing to separate routes for different data priorities depending on the use case. IoT Hub message routing adds latency. Save and send low-priority data at longer intervals, or use batch or file uploads. Malware detection on uploaded files increases latency. Separate messages based on time constraints. For example, send messages to IoT Hub directly when there's a time constraint, and use file upload via IoT Hub or batch data transfer like Azure Data Factory if there's no time constraint. You can use the Azure Blob Storage in IoT Edge module for file upload.	sensor sends temperature, humidity, and other telemetry, but also sends an alert when the temperature deviates from a defined range. The system classifies the alarm message as high priority and handles it differently than the temperature telemetry.
Use DPS to set up a connection to an IoT hub during provisioning, when the IoT Hub connection isn't available anymore, or during a device reboot. Use the evenly weighted distribution policy to adjust the weight for provisioning based on the use case. This approach optimizes resource allocation.	DPS helps ensure that devices are provisioned securely and efficiently. Provisioning devices over a period of time or in smaller batches helps balance the DPS load and quota, which ensures a smooth onboarding process. Allocating devices to IoT Hub in different regions based on latency improves performance and reduces connection times. Implementing a caching strategy for the DPS connection string reduces reconnect operations, which enhances overall system efficiency.
Optimize edge versus cloud processing. Use local compute to run real-time and near real-time workloads, or small, optimized, low-latency processing that has time constraints, on devices or at the edge. For larger workloads, or workloads that require extra external data or compute dependencies, use cloud resources. For example, you might run a machine learning algorithm at the edge to count people in a video stream, and send an event	Running real-time and near real-time workloads at the edge provides low-latency processing and immediate response times, which is crucial for time-sensitive applications. Testing with the expected maximum of downstream devices and messages ensures that the edge node can handle the load and maintain performance.

Recommendation	Benefit
that contains the count to the cloud. This approach provides quick local processing and efficient cloud-based trend analysis. Use the Azure Stream Analytics IoT Edge module to run analytics workloads, such as anomaly detection, at the edge. This practice helps label events with detected anomalies before sending them to the cloud, which improves overall system efficiency. Understand the overhead of an edge workload that includes many connected downstream devices. The edge node must forward or process all messages and cache all the data if there's intermittent cloud connectivity. To evaluate the performance effect on your solution, test your solution with the expected maximum of downstream devices and messages for each edge node. Also understand the performance effect that message translation or enrichment can have on edge, IoT Hub, or cloud event processing. Plan accordingly to maintain system efficiency.	
To optimize performance efficiency for high-volume cloud data, use built-in service integration between IoT Hub and data destinations, like Azure Data Lake Storage and Azure Data Explorer. These services are optimized for high-performance throughput. Use the Event Hubs SDK to develop custom ingestion from an IoT hub. The SDK includes an event processor that can rebalance devices and hosts. Use the appropriate number of IoT Hub partitions and consumer groups for the number of simultaneous data readers and required throughput. Separate the storage needed for data ingestion and event processing from the storage needed for reporting and integration. Use the data storage that fits your needs based on required throughput,	Optimized performance efficiency for high-volume cloud data ensures that your IoT solution can handle large amounts of data efficiently. Built-in service integration and the Event Hubs SDK help streamline data ingestion and processing. Proper configuration of IoT Hub partitions and consumer groups and separate storage for different purposes help improve data management. Appropriate data storage based on specific needs helps maintain the performance, scalability, and reliability of your IoT solution.

Recommendation	Benefit
size, retention period, data volume, CRUD requirements, and regional replication. Storage examples include Data Lake Storage, Azure Data Explorer, Azure SQL, and Azure Cosmos DB. For more information, see Select an Azure data store for your application.	
Separate the IoT solution ingestion pipeline from integration processing. Make sure complex queries or loads don't affect data ingestion performance in IoT Hub. Use well-defined and versioned APIs for access to IoT Hub information regarding device twins, module twins, jobs, and message routing. Prevent end users from creating user-defined queries against IoT Hub storage. Consider using separate data stores for integration and reporting.	Separation of the IoT solution ingestion pipeline and integration processing ensures that complex queries or loads from the integration layer don't affect data ingestion performance. This approach maintains the efficiency and reliability of IoT Hub.
Use Azure Monitor to collect IoT Hub metrics and send alerts for critical metrics. Set up Azure Monitor alerts based on your scale limits, such as device-to-cloud messages sent per second. Set the alert to a percentage of the limit, such as 75%, to notify you in advance. Set up Azure Monitor alerts for logs and metrics, such as the number of throttling errors. Set Azure Service Health service alerts to trigger notifications when the IoT Hub status changes.	IoT Hub metrics and alerts for critical metrics ensure proactive monitoring and management of your IoT solution. Alerts based on scale limits and a percentage of those limits help notify you before you reach scalability limits so that you can make timely adjustments. Alerts for logs and metrics, such as throttling errors, help you identify and address problems promptly. Alerts when the IoT Hub status changes provide awareness of potential disruptions and enable you to take necessary actions to maintain the reliability and performance of your IoT solution.

Azure policies

Azure provides an extensive set of built-in policies related to IoT Hub and its dependencies. Some of the preceding recommendations can be audited through Azure Policy. For example, you can check whether:

- IoT Hub has local authentication methods disabled for service APIs.
- Public network access is disabled for your IoT Hub device provisioning instance so that it's not accessible over the public internet.
- IoT Hub can only be accessed from a private endpoint.
- Resource logs in IoT Hub are enabled. These logs recreate activity trails that can help you investigate after a security incident or network compromise.

For comprehensive governance, review the Azure Policy built-in definitions for Internet of Things and other policies that might affect the security of the IoT solution.

Azure Advisor recommendations

Azure Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments. Advisor recommendations are aligned with Well-Architected Framework pillars.

For more information, see the recommendations in Azure Advisor.

Next step

IoT Hub documentation

ⓘ **Note:** The author created this article with assistance from AI. Learn more

---

Feedback

Was this page helpful? Yes No

Architecture best practices for Azure Kubernetes Service (AKS)

Article • 02/06/2025

Azure Kubernetes Service (AKS) is a managed Kubernetes service that you can use to deploy and manage containerized applications. Similar to other managed services, AKS offloads much of the operational overhead to Azure while providing high availability, scalability, and portability features to the workload.

This article assumes that, as an architect, you reviewed the compute decision tree and chose AKS as the compute for your workload. The guidance in this article provides architectural recommendations that are mapped to the principles of the Azure Well-Architected Framework pillars.

> **ⓘ Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern along with design strategies localized to the technology scope.
>
> Also included are recommendations for the technology capabilities that can help materialize those strategies. The recommendations don't represent an exhaustive list of all configurations that are available for AKS and its dependencies. Instead, they list the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept or to optimize your existing environments.
>
> Foundational architecture that demonstrates the key recommendations: **AKS baseline architecture**.

Technology scope

This review focuses on the interrelated decisions for the following Azure resources:

- AKS

When you discuss the Well-Architected Framework pillars' best practices for AKS, it's important to distinguish between *cluster* and *workload*. Cluster best practices are a shared responsibility between the cluster admin and their resource provider, while

workload best practices are the domain of a developer. This article has considerations and recommendations for each of these roles.

> **Note**
>
> The following pillars include a **design checklist** and a **list of recommendations** that indicate whether each choice is applicable to *cluster* architecture, *workload* architecture, or both.

Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover fast from failures**.

Reliability design principles provide a high-level design strategy applied for individual components, system flows, and the system as a whole.

Design checklist

Start your design strategy based on the design review checklist for Reliability. Determine its relevance to your business requirements while keeping in mind the features of AKS and its dependencies. Extend the strategy to include more approaches as needed.

- ✓ (Cluster) **Build redundancy to improve resiliency.** Use availability zones for your AKS clusters as part of your resiliency strategy to increase availability when you deploy to a single region. Many Azure regions provide availability zones. The zones are close enough to have low-latency connections among them, but far enough apart to reduce the likelihood that local outages will affect more than one zone.

  For critical workloads, deploy multiple clusters across different Azure regions. By geographically distributing AKS clusters, you can achieve higher resiliency and minimize the effects of regional failures. A multiregion strategy helps maximize availability and provide business continuity. Internet-facing workloads should use Azure Front Door or Azure Traffic Manager to route traffic globally across AKS clusters. For more information, see Multiregion strategy.

  Plan the IP address space to ensure that your cluster can reliably scale and handle failover traffic in multiple-cluster topologies.

- ✓ (Cluster and workload) **Monitor reliability and overall health indicators of the cluster and workloads.** Collect logs and metrics to monitor workload health,

identify performance and reliability trends, and troubleshoot problems. Review Best practices for monitoring Kubernetes with Azure Monitor and the Well-Architected Health modeling for workloads guide for help designing the reliability and health monitoring solution for your AKS solution.

Ensure that workloads are built to support horizontal scaling and report application readiness and health.

- ✓ (Cluster and workload) **Host application pods in user nodel pools.** By isolating system pods from application workloads, you help ensure that AKS essential services are unaffected by the resource demands or potential problems caused by a workload that runs user node pools.

  Ensure that your workload runs on user node pools and choose the right size SKU. At a minimum, include two nodes for user node pools and three nodes for the system node pool.

- ✓ (Cluster and workload) **Factor the AKS uptime service-level agreement (SLA) into your availability and recovery targets.** To define the reliability and recovery targets for your cluster and workload, follow the guidance in Recommendations for defining reliability targets. Then formulate a design that meets those targets.

- ✓ (Cluster and workload) Protect the AKS cluster service using Azure Backup by storing recovery points in a Backup vault and perform restore during any disaster scenario. To back up and restore the containerized applications and data running in AKS clusters, follow the guidance in the AKS backup overview for configuring protection.

Recommendations

Expand table

Recommendation	Benefit
(Cluster and workload) Control pod scheduling by using node selectors and affinity. In AKS, the Kubernetes scheduler can logically isolate workloads by hardware in the node. Unlike tolerations, pods that don't have a matching node selector can be scheduled on labeled nodes, but priority is given to pods that define the matching node selector.	Node affinity results in more flexibility, which allows you to define what happens if the pod can't be matched with a node.

Recommendation	Benefit
(Cluster) Choose the appropriate network plugin based on network requirements and cluster sizing. Different network plugins provide varying levels of functionality. Azure Container Networking Interface (Azure CNI) is required for specific scenarios, such as Windows-based node pools, some networking requirements, and Kubernetes network policies. For more information, see Kubenet versus Azure CNI.	The right network plugin can help ensure better compatibility and performance.
(Cluster and workload) Use the AKS uptime SLA for production-grade clusters.	The workload can support higher availability targets because of the higher availability guarantees of the Kubernetes API server endpoint for AKS clusters.
(Cluster) Use availability zones to maximize resilience within an Azure region by distributing AKS agent nodes across physically separate datacenters. If colocality requirements exist, use a regular virtual machine scale sets-based AKS deployment into a single zone or use proximity placement groups to minimize internode latency.	By spreading node pools across multiple zones, nodes in one node pool continue to run even if another zone goes down.
(Cluster and workload) Define pod resource requests and limits in application deployment manifests. Enforce those limits by using Azure Policy.	Container CPU and memory resource limits are necessary to prevent resource exhaustion in your Kubernetes cluster.
(Cluster and workload) Keep the system node pool isolated from application workloads. System node pools require a virtual machine (VM) SKU of at least 2 vCPUs and 4 GB of memory. We recommend that you use 4 vCPU or more. For more information, see System and user node pools.	The system node pool hosts critical system pods that are essential for the control plane of your cluster. By isolating these system pods from application workloads, you help ensure that the essential services are unaffected by the resource demands or potential problems caused by a workload.
(Cluster and workload) Separate applications to dedicated node pools based on specific requirements. Avoid large numbers of node pools to reduce management overhead.	Applications can share the same configuration and need GPU-enabled VMs, CPU or memory-optimized VMs, or the ability to scale to zero. By dedicating node pools to specific applications,

Recommendation	Benefit
	you can help ensure that each application gets the resources it needs without overprovisioning or underutilizing resources.
(Cluster) Use a NAT gateway for clusters that run workloads that make many concurrent outbound connections.	Azure NAT Gateway supports reliable egress traffic at scale and helps you avoid reliability problems by applying Azure Load Balancer limitations to high concurrent outbound traffic.
(Cluster and workload) Use Azure Backup to protect AKS cluster and restore to alternate regions during disaster. Azure Backup supports the backup and restore operations of containerized applications and data running for both cluster state and application data. You can use the backups in a regional disaster scenario and recover backups.	Azure Backup with Azure Kubernetes Service (AKS) offers a fully managed, scalable, secure, and cost-effective solution. Enhances the reliability of the workload without the complexities of setting up and maintaining backup infrastructure.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload.

The Security design principles provide a high-level design strategy for achieving those goals by applying approaches to the technical design of AKS.

Design checklist

Start your design strategy based on the design review checklist for Security and identify vulnerabilities and controls to improve the security posture. Familiarize yourself with AKS security concepts and evaluate the security hardening recommendations based on the CIS Kubernetes benchmark. Extend the strategy to include more approaches as needed.

- ✓ (Cluster) **Integrate with Microsoft Entra ID for identity and access mangement.** Centralize identity management for your cluster by using Microsoft Entra ID. Any change in user account or group status is automatically updated in access to the AKS cluster. Establish identity as the primary security perimeter. The developers and application owners of your Kubernetes cluster need access to different resources.

  Use Kubernetes role-based access control (RBAC) with Microsoft Entra ID for least privilege access. Protect configuration and secrets by minimizing the allocation of

administrator privileges.

- ✓ (Cluster) **Integrate with security monitoring and security information and event management tools.** Use Microsoft Defender for Containers with Microsoft Sentinel to detect and quickly respond to threats across your clusters and the workloads that run on them. Enable AKS connector for Microsoft Sentinel to stream your AKS diagnostics logs into Microsoft Sentinel.

- ✓ (Cluster and workload) **Implement segmentation and network controls.** To prevent data exfiltration, ensure that only authorized and safe traffic is allowed, and contain the blast radius of a security breach.

  Consider using a private AKS cluster to help ensure that cluster-management traffic to your API server remains on your private network. Or use the API server allowlist for public clusters.

- ✓ (Workload) **Use a web application firewall (WAF) to scan incoming traffic for potential attacks.** WAF can detect and mitigate threats in real time to help block malicious traffic before it reaches your applications. It provides robust protection against common web-based attacks, such as SQL injection, cross-site scripting, and other Open Web Application Security Project vulnerabilities. Some load balancers, such as Azure Application Gateway or Azure Front Door have an integrated WAF.

- ✓ (Workload) **Maintain a hardened workload's software supply chain.** Ensure that your continuous integration and continuous delivery pipeline is hardened with container-aware scanning.

- ✓ (Cluster and workload) **Implement extra protection for specialized secure workloads.** If your cluster needs to run a sensitive workload, you might need to deploy a private cluster. Here are some examples:
  - Payment Card Industry Data Security Standard (PCI-DSS 3.2.1): AKS regulated cluster for PCI-DSS 3.2.1
  - DoD Impact Level 5 (IL5) support and requirements with AKS: Azure Government IL5 isolation requirements.

Recommendations

Expand table

Recommendation	Benefit
(Cluster) Use managed identities on the cluster.	You can avoid the overhead associated with managing and rotating service principles.

Recommendation	Benefit
(Workload) Use Microsoft Entra Workload ID with AKS to access Microsoft Entra protected resources, such as Azure Key Vault and Microsoft Graph, from your workload.	Use AKS Workload IDs to protect access to Azure resources by using Microsoft Entra ID RBAC without having to manage credentials directly in your code.
(Cluster) Use Microsoft Entra ID to authenticate with Azure Container Registry from AKS.	By using Microsoft Entra ID, AKS can authenticate with Container Registry without the use of `imagePullSecrets` secrets.
(Cluster) Secure network traffic to your API server by using private AKS cluster if the workload requirements require higher levels of segmentation.	By default, network traffic between your node pools and the API server travels the Microsoft backbone network. By using a private cluster, you can help ensure that network traffic to your API server remains on the private network only.
(Cluster) For public AKS clusters, use API server-authorized IP address ranges. Include sources like the public IP addresses of your deployment build agents, operations management, and node pools' egress point, such as Azure Firewall.	When you use public clusters, you can significantly reduce the attack surface of your AKS cluster by limiting the traffic that can reach the API server of your clusters.
(Cluster) Protect the API server by using Microsoft Entra ID RBAC. Disable local accounts to enforce all cluster access by using Microsoft Entra ID-based identities.	Securing access to the Kubernetes API server is one of the most important things that you can do to secure your cluster. Integrate Kubernetes RBAC with Microsoft Entra ID to control access to the API server.
(Cluster) Use Azure network policies or Calico.	By using policies, you can secure and control network traffic between pods in a cluster. Calico provides a richer set of capabilities, including policy ordering and priority, deny rules, and more flexible match rules.
(Cluster) Secure clusters and pods by using Azure Policy.	Azure Policy can help apply at-scale enforcement and safeguards on your clusters in a centralized, consistent manner. It can also control what functions pods are granted and detect whether anything is running against company policy.
(Cluster) Secure container access to resources. Limit access to actions that containers can perform. Provide the least number of permissions, and avoid the use of root or privileged escalation. For Linux based containers, see Security	By restricting permissions and avoiding the use of root or privileged escalation, you help reduce the risk of security breaches. You can help ensure that, even if a container is compromised, the potential damage is minimized.

Recommendation	Benefit
container access to resources using built-in Linux security features.	
(Cluster) Control cluster egress traffic by ensuring that your cluster's outbound traffic passes through a network security point such as Azure Firewall or an HTTP proxy.	By routing outbound traffic through Azure Firewall or an HTTP proxy, you can help enforce security policies that prevent unauthorized access and data exfiltration. This approach also simplifies the administration of security policies and makes it easier to enforce consistent rules across your entire AKS cluster.
(Cluster) Use the open-source Microsoft Entra Workload ID and Secrets Store CSI Driver with Key Vault.	These features help you protect and rotate secrets, certificates, and connection strings in Key Vault by using strong encryption. They provide an access audit log and keep core secrets out of the deployment pipeline.
(Cluster) Use Microsoft Defender for Containers.	Microsoft Defender for Containers helps you monitor and maintain the security of your clusters, containers, and their applications.

Cost Optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget while meeting business requirements.

The Cost Optimization design principles provide a high-level design strategy for achieving those goals and making tradeoffs as necessary in the technical design related to AKS and its environment.

Design checklist

Start your design strategy based on the design review checklist for Cost Optimization for investments. Fine-tune the design so that the workload is aligned with the budget that's allocated for the workload. Your design should use the right Azure capabilities, monitor investments, and find opportunities to optimize over time.

- ✓ (Cluster) **Include the pricing tiers for AKS in your cost model.** To estimate costs, use the Azure pricing calculator and test different configuration and payment plans in the calculator.

- ✓ (Cluster) **Get the best rates for your workload.** Use the appropriate VM SKU for each node pool because it directly affects the cost to run your workloads. Choosing a high-performance VM without proper utilization can lead to wasteful spending. Selecting a less powerful VM can cause performance problems and increased downtime.

  If you properly planned for capacity and your workload is predictable and will exist for an extended period of time, sign up for Azure Reservations or a savings plan to reduce your resource costs.

  Choose Azure Spot Virtual Machines to use unutilized Azure capacity with significant discounts. These discounts can reach up to 90% of pay-as-you-go prices. If Azure needs capacity back, the Azure infrastructure evicts the Spot nodes.

  If you run AKS on-premises or at the edge, you can also use Azure Hybrid Benefit to reduce costs when you run containerized applications in those scenarios.

- ✓ (Cluster and workload) **Optimize workload components costs.** Choose the most cost-effective region for your workload. Evaluate the cost, latency, and compliance requirements to ensure that you run your workload cost-effectively and that it doesn't affect your customers or create extra networking charges. The region where you deploy your workload in Azure can significantly affect the cost. Because of many factors, the cost of resources varies for each region in Azure.

  Maintain small and optimized images to help reduce costs because new nodes need to download those images. User request failures or timeouts when the application is starting up can lead to overprovisioning. Build images in a way that allows the container to start as soon as possible to help avoid failures and timeouts.

  Review the Cost Optimization recommendations in Best practices for monitoring Kubernetes with Azure Monitor to determine the best monitoring strategy for your workloads. Analyze performance metrics, starting with CPU, memory, storage, and network, to identify cost optimization opportunities by cluster, nodes, and namespace.

- ✓ (Cluster and workload) **Optimize workload scaling costs.** Consider alternative vertical and horizontal scaling configurations to reduce scaling costs while still meeting all workload requirements. Use autoscalers to scale in when workloads are less active.

- ✓ (Cluster and workload) **Collect and analyze cost data.** The foundation of enabling cost optimization is the spread of a cost-saving cluster. Develop a cost-efficiency mindset that includes collaboration between finance, operations, and engineering

teams to drive alignment on cost-saving goals and bring transparency to cloud costs.

Recommendations

Expand table

Recommendation	Benefit
(Cluster and workload) Align AKS SKU selection and managed disk size with workload requirements.	Matching your selection to your workload demands helps ensure that you don't pay for unneeded resources.
(Cluster) Choose the right VM instance types for your AKS node pools. To determine the right VM instance types, consider workload characteristics, resource requirements, and availability needs.	Selecting the right VM instance type is crucial because it directly affects the cost to run applications on AKS. Choosing a high-performance instance without proper utilization can lead to wasteful spending. Choosing a less powerful instance can lead to performance problems and increased downtime.
(Cluster) Choose VMs based on the more power efficient Azure Resource Manager architecture. AKS supports creating Arm64 node pools and a mix of Intel and Resource Manager architecture nodes within a cluster.	The Arm64 architecture provides a better price-to-performance ratio because of its lower power utilization and efficient compute performance. These capabilities can bring better performance at a lower cost.
(Cluster) Enable the cluster autoscaler to automatically reduce the number of agent nodes in response to excess resource capacity.	Automatically scaling down the number of nodes in your AKS cluster lets you run an efficient cluster when demand is low and scale up when demand increases.
(Cluster) Enable node autoprovisioning to automate VM SKU selection.	Node autoprovision simplifies the SKU selection process and decides, based on pending pod resource requirements, the optimal VM configuration to run workloads in the most efficient and cost-effective manner.
(Workload) Use HorizontalPodAutoscaler to adjust the number of pods in a deployment depending on CPU utilization or other metrics.	Automatically scaling down the number of pods when demand is low and scaling out when demand increases results in a more cost-effective operation of your workload.
(Workload) Use VerticalPodAutoscaler (preview) to rightsize your pods and dynamically set requests and limits based on historic usage.	By setting resource requests and limits on containers for each workload, VerticalPodAutoscaler frees up CPU and memory for other pods and helps ensure effective utilization of your AKS clusters.

Recommendation	Benefit
(Cluster) Configure the AKS cost analysis add-on.	The cost analysis cluster extension enables you to obtain granular insight into costs that are associated with various Kubernetes resources in your clusters or namespaces.

Operational Excellence

Operational Excellence primarily focuses on procedures for **development practices, observability, and release management**.

The Operational Excellence design principles provide a high-level design strategy for achieving those goals for the operational requirements of the workload.

Design checklist

Start your design strategy based on the design review checklist for Operational Excellence for defining processes for observability, testing, and deployment. See AKS best practices and Day-2 operations guide to learn about key considerations to understand and implement.

- ✓ (Cluster) **Implement an infrastructure as code (IaC) deployment approach.** Use a declarative, template-based deployment approach by using Bicep, Terraform, or similar tools. Make sure that all deployments are repeatable, traceable, and stored in a source code repo. For more information, see the quickstarts in the AKS product documentation.

- ✓ (Cluster and workload) **Automate infrastructure and workload deployments.** Use standard software solutions to manage, integrate, and automate the deployment of your cluster and workloads. Integrate deployment pipelines with your source control system and incorporate automated tests.

  Build an automated process to help ensure that your clusters are bootstrapped with the necessary cluster-wide configurations and deployments. This process is typically performed by using GitOps.

  Use a repeatable and automated deployment processes for your workload within your software development lifecycle.

- ✓ (Cluster and workload) **Implement a comprehensive monitoring strategy.** Collect logs and metrics to monitor the health of the workload, identify trends in performance and reliability, and troubleshoot problems. Review the Best practices

for monitoring Kubernetes with Azure Monitor and the Well-Architected Recommendations for designing and creating a monitoring system to determine the best monitoring strategy for your workloads.

Enable diagnostics settings to ensure that control plane or core API server interactions are logged.

The workload should be designed to emit telemetry that can be collected, which should also include liveness and readiness statuses.

- ✓ (Cluster and workload) **Implement testing in production strategies.** Testing in production uses real deployments to validate and measure an application's behavior and performance in the production environment. Use chaos engineering practices that target Kubernetes to identify application or platform reliability issues.

  Azure Chaos Studio can help simulate faults and trigger disaster recovery situations.

- ✓ (Cluster and workload) **Enforce workload governance.** Azure Policy helps ensure consistent compliance with organizational standards, automates policy enforcement, and provides centralized visibility and control over your cluster resources.

  Review the Azure policies section to learn more about the available built-in policies for AKS.

- ✓ (Cluster and workload) **Use stamp-level, blue-green deployments for mission-critical workloads.** A stamp-level, blue-green deployment approach can increase confidence in releasing changes and enables zero-downtime upgrades because compatibilities with downstream dependencies like the Azure platform, resource providers, and IaC modules can be validated.

  Kubernetes and ingress controllers support many advanced deployment patterns for inclusion in your release engineering process. Consider patterns like blue-green deployments or canary releases.

- ✓ (Cluster and workload) **Make workloads more sustainable.** Making workloads more sustainable and cloud efficient requires combining efforts around *cost optimization*, *reducing carbon emissions*, and *optimizing energy consumption*. Optimizing the application's cost is the initial step in making workloads more sustainable.

  See Sustainable software engineering principles in AKS to learn how to build sustainable and efficient AKS workloads.

Recommendations

Expand table

Recommendation	Benefit
(Cluster) Operationalize cluster and pod configuration standards by using Azure policies for AKS.	Azure policies for AKS can help you apply at-scale enforcement and safeguards on your clusters in a centralized, consistent manner. Use policies to define the permissions granted to pods and ensure compliance with company policies.
(Workload) Use Kubernetes Event Driven Autoscaler (KEDA).	KEDA allows your applications to scale based on events, like the number of events being processed. You can choose from a rich catalog of more than 50 KEDA scalers.

Performance Efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance.

The Performance Efficiency design principles provide a high-level design strategy for achieving those capacity goals against the expected usage.

Design checklist

Start your design strategy based on the design review checklist for Performance Efficiency for defining a baseline based on key performance indicators for AKS.

- ✓ (Cluster and workload) **Conduct capacity planning.** Perform and iterate on a detailed capacity plan exercise that includes SKU, autoscale settings, IP addressing, and failover considerations.

  After you formalize your capacity plan, frequently update the plan by continuously observing the resource utilization of the cluster.

- ✓ (Cluster) **Define a scaling strategy.** Configure scaling to ensure that resources are adjusted efficiently to meet workload demands without overuse or waste. Use AKS features like cluster autoscaling and HorizontalPodAutoscaler to dynamically meet your workload needs with less strain on operations. Optimize your workload to operate and deploy efficiently in a container.

  Review the Scaling and partitioning guide to understand the various aspects of scaling configuration.

- ✓ (Cluster and workload) **Conduct performance testing.** Perform ongoing load testing activities that exercise both the pod and cluster autoscaler. Compare results against the performance targets and the established baselines.
- ✓ (Cluster and workload) **Scale workloads and flows independently.** Separate workloads and flows into different node pools to allow independent scaling. Follow the guidance in Optimize workload design using flows to identify and prioritize your flows.

Recommendations

Expand table

Recommendation	Benefit
(Cluster) Enable cluster autoscaler to automatically adjust the number of agent nodes in response to workload demands. Use the HorizontalPodAutoscaler to adjust the number of pods in a deployment depending on CPU utilization or other metrics.	The ability to automatically scale up or scale down the number of nodes and the number of pods in your AKS cluster lets you run an efficient, cost-effective cluster.
(Cluster and workload) Separate workloads into different node pools and consider scaling user node pools.	Unlike system node pools that always require running nodes, user node pools allow you to scale up or scale down.
(Workload) Use AKS advanced scheduler features to implement advanced balancing of resources for workloads that require them.	As you manage AKS clusters, you often need to isolate teams and workloads. Advanced features that the Kubernetes scheduler provides let you control which pods can be scheduled on certain nodes. They also let you control how multipod applications can be appropriately distributed across the cluster.
(Workload) Use KEDA to build a meaningful autoscale ruleset based on signals that are specific to your workload.	Not all scale decisions can be derived from CPU or memory metrics. Scale considerations often come from more complex or even external data points. KEDA allows your applications to scale based on events, such as the number of messages in a queue or the length of a topic lag.

Azure policies

Azure provides an extensive set of built-in policies related to AKS that apply to the Azure resource, like typical Azure policies and the Azure Policy add-on for Kubernetes, and within the cluster. Many of the Azure resource policies come in both *Audit/Deny* and a *Deploy If Not Exists* variants. In addition to the built-in Azure Policy definitions, you can create custom policies for both the AKS resource and for the Azure Policy add-on for Kubernetes.

Some of the recommendations in this article can be audited through Azure Policy. For example, you can check the following cluster policies:

- Clusters have readiness or liveness health probes configured for your pod spec.
- Microsoft Defender for cloud-based policies.
- Authentication mode and configuration policies, like Microsoft Entra ID, RBAC, and disable local authentication.
- API server network access policies, including private cluster.
- GitOps configuration policies.
- Diagnostics settings policies.
- AKS version restrictions.
- Prevent command invoke.

You can also check the following cluster and workload policies:

- Kubernetes cluster pod security initiatives for Linux-based workloads.
- Include pod and container capability policies, such as AppArmor, sysctl, security caps, SELinux, seccomp, privileged containers, and automount cluster API credentials.
- Mount, volume drivers, and filesystem policies.
- Pod and container networking policies, such as host network, port, allowed external IPs, HTTPs, and internal load balancers.
- Namespace deployment restrictions.
- CPU and memory resource limits.

For comprehensive governance, review the Azure Policy built-in definitions for Kubernetes and other policies that might affect the security of the compute layer.

Azure Advisor recommendations

Azure Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments. Here are some recommendations that can help you improve the reliability, security, cost effectiveness, performance, and operational excellence of AKS.

- Reliability
- Security
- Cost Optimization
- Operational Excellence
- Performance

Related content

Consider the following articles as resources that demonstrate the recommendations highlighted in this article.

- AKS baseline architecture
- Advanced AKS microservices architecture
- AKS cluster for a PCI-DSS workload
- AKS baseline for multiregion clusters
- AKS Landing Zone Accelerator

Build implementation expertise by using the following product documentation:

- AKS product documentation

Feedback

Was this page helpful? Yes No

Architecture best practices for Azure Load Balancer

Article • 02/04/2025

The load balancing process distributes network traffic to a group of two or more back-end servers. Azure Load Balancer is an Azure-native service that does Layer-4 load balancing for User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). Load Balancer helps provide low latency and high availability for regional and global deployments.

This article assumes that as an architect, you've reviewed the load-balancing options in Azure and chose Load Balancer for your workload. The guidance in this article provides architectural recommendations that are mapped to the principles of the Well-Architected Framework pillars.

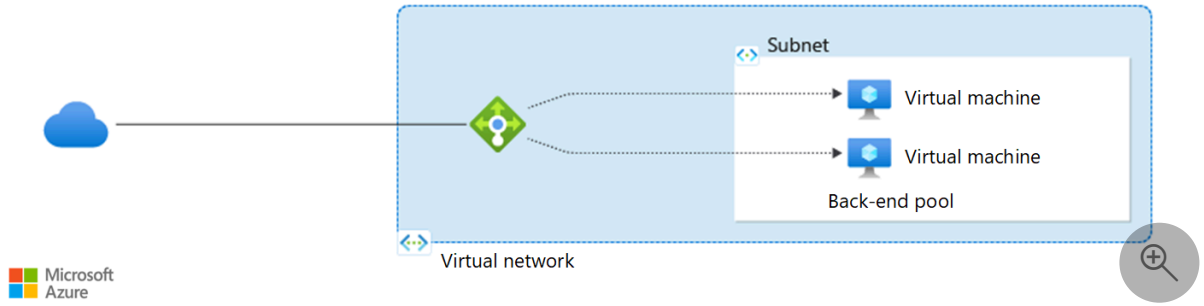
> **ⓘ Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern along with design strategies localized to the technology scope.
>
> Also included are recommendations for the technology capabilities that can help materialize those strategies. The recommendations don't represent an exhaustive list of all configurations that are available for Load Balancer and its dependencies. Instead, they list the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept or to optimize your existing environments.
>
> Foundational architecture that demonstrates the key recommendations: Azure Virtual Machines baseline architecture.

Technology scope

This review focuses on the interrelated decisions for the following Azure resources:

- Load Balancer

This guidance focuses the Standard Load Balancer SKU. Basic Load Balancer and Gateway Load Balancer SKUs are out of scope for this article.



> **Note**
>
> For HTTP applications, consider Azure Application Gateway or Azure Front Door instead of Load Balancer. These alternatives manage load balancing and also provide features like Web Application Firewall (WAF) and Transport Layer Security (TLS) termination.
>
> For more information, see:
>
> - Well-Architected Framework perspective on Azure Front Door
> - Well-Architected Framework perspective on Azure Application Gateway

Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover fast from failures**.

Reliability design principles provide a high-level design strategy applied for individual components, system flows, and the system as a whole.

Design checklist

Start your design strategy based on the design review checklist for Reliability. Determine its relevance to your business requirements while keeping in mind the tiers and features of virtual machines (VMs). Extend the strategy to include more approaches as needed.

- ✓ **Understand the impact of Microsoft-backed guarantees.** In addition to other components in your architecture, factor service-level agreements (SLA) into the reliability target of your workload. Keep in mind the following important points:
  - If the load-balanced endpoint can't connect to all healthy back-end servers for an entire minute, that minute is considered unavailable. However, if at least one

request succeeds within the same minute, even if others fail, that minute isn't considered downtime.

  - Downtime doesn't include minutes caused by Source Network Address Translation (SNAT) port exhaustion. Ensure that you configure your workload to handle the expected number of connections and open ports accordingly.

- ✓ **Support zone redundancy in your workload architecture.** We recommend the Standard Load Balancer SKU. It has reliability features, such as availability zone support, traffic distribution across multiple regions, and the ability to handle more instances in the back-end pool. These feature help withstand failures at zonal, regional, and individual VM instance levels. Be aware of limitations, such as the maximum back-end pool size.

  > **Note**
  >
  > In Load Balancer, you manage the number of load-balanced VMs but not the number of Load Balancer instances. You can configure a Load Balancer instance to be zone redundant, or pin it to a zone if the workload needs to collocate VMs in a single zone. The front-end IP address's zonal or multizone configuration dictates the load-balancing redundancy.

- ✓ **Support regional redundancy in your workload architecture.** You can configure Load Balancer to be a global load balancer. In this setup, Load Balancer has a static anycast public IP address that broadcasts to multiple regions. When clients request this IP address, their requests go to the closest server instance. Load Balancer connects to regional load balancers to distribute traffic efficiently.

- ✓ **Evaluate changes in your networking stack to support reliable scaling.** Consider scaling out the back-end pool by using autoscale rules. Be aware of potential SNAT port exhaustion for outbound traffic. To address this problem, use Azure NAT Gateway for easier configuration, but understand that it lacks availability zone redundancy. Alternatively, use Load Balancer for added zone redundancy. For more information, see Outbound connections.

- ✓ **Mitigate potential failures.** Do failure mode analysis and identify mitigations. The following table shows types of failures and how to mitigate them.

Expand table

Failure	Mitigation
Traffic is routed to unhealthy application instances.	Monitor workload instance health. Implement HTTP health probes that include checks for workload dependencies.
Traffic is routed to a region that has an outage.	Deploy extra instances in another region. Add a global load balancer to redirect traffic to the new region.
The workload's user base was expanded to support users in a new region, and they have high latency. The application now experiences a high number of timeouts and failures.	Deploy extra instances in a new region, and add them in the service configuration. As a global load balancer, Azure Load Balancer routes traffic closer to the users.

✓ **Route traffic to healthy instances.** You can use HTTP or TCP for health probes. To provide richer status responses, consider creating an HTTP endpoint for health checks, even for non-HTTP apps. This approach is especially useful for checking dependencies and databases. Without HTTP probes, the load balancer relies on TCP connections, which might not accurately reflect VM health.

You can configure the health probe on Load Balancer. For more information, see Design guidance on health probes.

Recommendations

Expand table

Recommendation	Benefit
Select the Standard Load Balancer SKU. For more information, see SKU comparison.	This SKU supports reliability features, such as availability zones and multiregion load balancing.
Configure rules to map the front-end IP address to the back-end servers' IP addresses to enable load balancing. The back-end address pool should have at least two back-end endpoints to load balance for redundancy.	Rules are the core of the load balancing algorithm. Without this configuration, the distribution modes are disabled.
Configure health probes. - Set the probing interval and threshold values. Consider the tradeoff between how fast you can detect failures and the number of requests to the	Only healthy back-end pool instances receive new connections. This configuration helps maintain high availability and reliability because it

Recommendation	Benefit
endpoint. - Evaluate whether you want to send traffic to the instances when all instances have an unhealthy status. You might use this configuration to implement a graceful degradation experience. For more information, see AllProbedUp.	routes traffic away from unhealthy instances.
Configure private and public IP addresses to be zone redundant. The IP address determines the zone redundancy of Load Balancer.	Zone redundancy helps the workload withstand zonal failures. When one zone fails, services can fail over to one of the remaining zones.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload.

The Security design principles provide a high-level design strategy for achieving those goals by applying approaches to the technical design of Load Balancer.

Design checklist

Start your design strategy based on the design review checklist for Security and identify vulnerabilities and controls to improve the security posture. Extend the strategy to include more approaches as needed.

- ✓ **Review security baselines.** To enhance the security posture of your application that's load balanced by Azure Load Balancer, review the security baseline for Load Balancer.
- ✓ **Protect the back-end servers.** Deploy resources in a virtual network that doesn't have direct internet exposure. Front the virtual network with a load balancer. Ideally, the load balancer should have firewall capabilities. For HTTP applications, consider Application Gateway or Azure Front Door. For non-HTTP applications, consider Load Balancer with a private IP address (internal load balancer), and route traffic through Azure Firewall for added security. For more information, see Internal load balancer.

  You can also use Load Balancer as a reverse proxy. In that case, the load balancer has a public IP address with SNAT, which exposes resources while masking their IP addresses.

> **Note**
>
> To filter traffic to back-end servers, use network security groups (NSGs) on the subnets that contain the front end and back end. Don't apply NSGs directly to the Load Balancer service. When NSGs enforce rules, they consider the source ports, destination ports, and address ranges of the originating and destination computers, not the load balancer.

- ✓ **Design for private connectivity.** Load Balancer works with Azure Private Link. If you spread application resources across virtual networks, you can connect resources in different virtual networks. Use virtual network peering or place Private Link in front of the internal load balancer. The Private Link option provides more secure access without needing a public IP address. It also restricts access from nonpeered networks.

  You can authorize private links via role-based access control to restrict access to only the identities that need it.

- ✓ **Protect your application from threats at the network edge.** For designs that use Load Balancer as the point of entry, implement traffic inspection at the endpoint level. This design doesn't have built-in security features like a WAF, so you must add extra measures to help secure HTTP applications. For more information, see Public load balancer. Also ensure that you protect the load balancer endpoints from distributed denial-of-service (DDoS) attacks.

- ✓ **Encrypt network traffic.** Load Balancer works at Layer 4 and fully supports load balancing TCP and UDP traffic. Load Balancer doesn't support Secure Sockets Layer (SSL) and TLS termination. For HTTPS load balancing at the application layer, use Application Gateway.

Recommendations

Expand table

Recommendation	Benefit
Configure the front-end IP address to a private IP address in a virtual network.	This approach helps ensure that the front-end IP address and virtual network remain isolated from direct internet exposure. The internal load balancer can't accept incoming traffic from the internet, which reduces potential attack vectors.

Recommendation	Benefit
Protect the public load balancer with Azure DDoS Protection.	A DDoS Protection plan provides advanced protection, including detection capabilities that monitor the endpoint for threats and signs of abuse.

Cost Optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget while meeting business requirements.

The Cost Optimization design principles provide a high-level design strategy for achieving those goals and making tradeoffs as necessary in the technical design related to Load Balancer and its environment.

Design checklist

Start your design strategy based on the design review checklist for Cost Optimization for investments. Fine-tune the design so that the workload is aligned with the budget that's allocated for the workload. Your design should use the right Azure capabilities, monitor investments, and find opportunities to optimize over time.

- ✓ **Factor load balancing expenses into your cost model.** Consider primary factors, such as the amount of data that Load Balancer processes and the number of inbound and outbound load balancing rules. For a precise cost estimation, use traffic logs to gauge your inbound and outbound traffic needs.
- ✓ **Set controls on spending.** Log and analyze Load Balancer costs. To manage costs effectively, use Microsoft Cost Management to create budgets and configure alerts. Costs can accumulate based on the amount of logged data and the storage duration, which affects bandwidth and storage expenses.
- ✓ **Remove unused resources.** Identify and remove unused load balancer instances. Analyze logs to evaluate usage. Delete load balancer instances that aren't associated with back-end VMs. Examine traffic logs to find underused resources.
- ✓ **Optimize flow costs.** Use efficient protocols and data compression to reduce the load on the traffic flow and minimize costs.

  To optimize costs, you can reduce the number of rules. Instead of having rules that use individual IP addresses and ports for each endpoint, define a rule for a range of ports in the front end that connects to a back-end pool.

Implement optimization in the back-end flows. For example, multiple database queries that a load balancer intercepts can increase costs per query. To prevent this extra cost, consider implementing a stored procedure to consolidate the sequence of queries.

- ✓ **Evaluate the cost of operations.** Consider resource expenses and operational costs, like maintenance, scaling, and compliance. Load balancer rules can significantly affect costs. Reduce the number of rules to optimize financial and management costs.

Recommendations

Expand table

Recommendation	Benefit
Use the Azure pricing calculator to estimate costs.	You can convert anticipated traffic usage into cost estimates, which makes it easier to plan and budget.
Evaluate the number of rules and reduce them, if possible. Evaluate whether you can use one rule to summarize a range of ports instead of defining several rules for individual IP addresses. For example, you can use inbound NAT rules to map IP addresses and ports to a back-end pool rather than to individual VMs.	Consolidated rules optimize costs and simplify operations. When you scale up or down, you can add or remove IP addresses from the back-end pool without changing any rules.

Operational Excellence

Operational Excellence primarily focuses on procedures for **development practices, observability, and release management**.

The Operational Excellence design principles provide a high-level design strategy for achieving those goals for the operational requirements of the workload.

Design checklist

Start your design strategy based on the design review checklist for Operational Excellence for defining processes for observability, testing, and deployment related to Load Balancer.

- ✓ **Use infrastructure as code.** Deploy and configure Load Balancer along with other networking components, like virtual networks, network peerings, private endpoints, and NSGs. Familiarize yourself with the Microsoft.Network loadBalancers resource type.
- ✓ **Use layered deployment for hub-and-spoke architectures.** Deploy the hub first because it changes less frequently than the workload that's deployed in the spoke network. Deploy the load balancer with the workload. If you reuse a single load balancer across multiple workloads, consider placing it in the hub.
- ✓ **Implement a comprehensive networking monitoring system.** Impelement diagnostic capabilities, like multidimensional metrics for real-time insights and alerts, resource logs based on the health event schema, and the Azure Monitor Insights dashboard for comprehensive load balancer monitoring.

Recommendations

Expand table

Recommendation	Benefit
Use multidimensional metrics. To minimize excessive alerts, set the aggregation type to `Average`, and use a five-minute data window that has a 95% threshold. For more information, see Configure alerts for multidimensional metrics. Review the examples for inbound and outbound availability.	Comprehensive real-time insights and alert configuration provide enhanced detection of problems and enable prompt responses.
Capture resource logs. Load Balancer entries depend on the health event schema.	Logs provide detailed records of events so that you can quickly identify and resolve problems.
Use the built-in Azure Monitor Insights dashboard for Load Balancer.	Visualization facilitates well-informed design choices and helps you quickly identify, diagnose, and fix problems.
During maintenance operations, set the Admin state to `Down` to take a back-end instance out of rotation without disrupting existing connections. This configuration helps ensure that no new connections are forwarded to the back-end instance, while existing connections are gracefully terminated.	This *Admin state* configuration helps reduce overhead and complexity when you take a VM out of the load balancing rotation for regular maintenance or patching. As an alternative option to take a back-end instance out of rotation, you can apply NSGs to block traffic from the Load Balancer health probe or the

Recommendation	Benefit
	client's IP addresses and ports. This option increases complexity.

Performance Efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance.

The Performance Efficiency design principles provide a high-level design strategy for achieving those capacity goals against the expected usage.

Design checklist

Start your design strategy based on the design review checklist for Performance Efficiency for defining a baseline that's based on key performance indicators for Load Balancer.

- ✓ **Determine the network performance targets.** The load balancer has no limit on the traffic that it can support. But when you define performance targets and plan for capacity, you should test network performance.

  Use stress testing to understand the workload's bandwidth requirements. Include the load balancer in those tests. If a single virtual machine scale set with multiple VMs isn't sufficient, you might add another scale set by using the same load balancer. If the VMs don't receive requests quickly enough, you might need to adjust the networking components, such as adding more load balancers. But instead of changing the load balancer, consider making design changes and optimizing your workload to handle the load better.

- ✓ **Understand the limits when you design your scaling strategy.** To meet performance requirements and scale your workload, add or remove VMs from the back-end pool. A single back-end pool in Standard Load Balancer can handle up to 5,000 VMs.

  Load Balancer doesn't apply throughput limits. But throughput limits for VMs and virtual networks still apply. For more information, see VM network bandwidth.

- ✓ **Serve requests quickly.** Standard Load Balancer has a tier that routes traffic to back-end endpoints based on their geographic proximity to the user.

Load Balancer also supports load distribution based on session persistence. When you enable this feature, requests from the same client are consistently directed to the same back-end server that handled their previous sessions.

- ✓ **Collect data to analyze performance.** Load Balancer multidimensional metrics can analyze the service's performance. Configure alerts to detect performance changes. Use tools like the Azure Monitor Insights dashboard to visualize the status of Load Balancer. Ensure that the resource health feature monitors the health status, and stay informed about performance problems and outages.

- ✓ **Optimize network traffic.** Don't process the same data multiple times in separate steps. Instead, perform all necessary calculations in a batch, and then persist the data. This approach reduces latency and minimizes network traffic, which improves overall performance.

Recommendations

Expand table

Recommendation	Benefit
Select the global tier in Standard Load Balancer if you have global users.	This tier's geo-proximity distribution mode serves user requests from endpoints in the nearest region, which improves performance.
Evaluate whether you should enable session persistence when you want requests from the same user to direct to the same back-end server. From a reliability perspective, we don't recommend this approach. If you use this option, the application should gracefully recover without disrupting user sessions. There's also a load balancing tradeoff because it restricts the flexibility of distributing traffic across multiple back ends evenly.	Session persistence can optimize performance and maintain continuity for user sessions, especially when applications rely on maintaining state information locally. But there are tradeoffs.
During scale-out, send a probe-down signal until the application is fully initialized and ready to handle requests. During scale-in, send a probe-down signal for new connections on the endpoint that's	Health probes can help optimize scaling operations. They help ensure that during scale-out, the application can handle the incoming load. Before a scale-in operation, they allow for a smooth reduction in instances without disrupting ongoing operations.

Recommendation	Benefit
being scaled back. Pending requests on existing connections continue to be processed.	

Azure policies

Azure provides an extensive set of built-in policies related to Load Balancer and its dependencies. Some of the preceding recommendations can be audited through Azure Policy. For example, you can check whether:

- Load balancers, excluding Basic SKU load balancers, have resiliency features enabled for public IP addresses in their front end.
- Resource logs are enabled to track activities and events that occur on your resources and provide visibility and insights into changes.

For comprehensive governance, review the Azure Policy built-in definitions for Load Balancer and other policies that might impact the security of traffic distribution.

Azure Advisor recommendations

Azure Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments. Advisor recommendations are aligned with Well-Architected Framework pillars.

For more information, see the recommendations in Azure Advisor.

Related resources

- Virtual Machines baseline architecture
- Load Balancer documentation

Feedback

Was this page helpful? Yes No

Azure Well-Architected Framework perspective on Azure Machine Learning

Article • 03/18/2024

Azure Machine Learning is a managed cloud service that you can use to train, deploy, and manage machine learning models. There are a wide range of choices and configurations for both training and deploying models, including compute SKUs and configurations. You can deploy Machine learning models to Machine Learning compute or to other Azure services such as Azure Kubernetes Service (AKS).

This article provides architectural recommendations for making informed decisions when you use Machine Learning to train, deploy, and manage machine learning models. The guidance is based on the Azure Well-Architected Framework pillars.

> **ⓘ Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern along with design strategies localized to the technology scope.
>
> Also included are *recommendations* on the technology capabilities that can help materialize those strategies. The recommendations don't represent an exhaustive list of all configurations available for Machine Learning and its dependencies. Instead, they list the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept or to optimize your existing environments.
>
> The foundational architecture **baseline OpenAI end-to-end chat reference architecture** demonstrates many of the key recommendations.

Technology scope

This review focuses on the interrelated decisions for these Azure resources:

- Machine Learning
- Machine Learning compute clusters
- Machine Learning compute instances

The review doesn't address connected resources such as data stores or Azure Key Vault.

Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover fast from failures**.

The Reliability design principles provide a high-level design strategy applied for individual components, system flows, and the system as a whole.

Design checklist

Start your design strategy based on the design review checklist for Reliability and determine its relevance to your business requirements. Extend the strategy to include more approaches as needed.

- ✓ **Resiliency**: Deploy models to environments that support availability zones, such as AKS. By ensuring deployments are distributed across availability zones, you're ensuring a deployment is available even in the event of a datacenter failure. For enhanced reliability and availability, consider a multi-region deployment topology.
- ✓ **Resiliency**: Ensure you have sufficient compute for both training and inferencing. Through resource planning, make sure your compute SKU and scale settings meet the requirements of your workload.
- ✓ **Resiliency**: Segregate Machine Learning workspaces used for exploratory work from those used for production.
- ✓ **Resiliency**: When using managed online endpoints for inferencing, use a release strategy such as blue-green deployments to minimize downtime and reduce the risk associated with deploying new versions.
- ✓ **Business requirements**: Select your use of compute clusters, compute instances, and externalized inference hosts based on reliability needs, considering service-level agreements (SLAs) as a factor.
- ✓ **Recovery**: Ensure you have self-healing capabilities, such as checkpointing features supported by Machine Learning, when training large models.
- ✓ **Recovery**: Ensure you have a recovery strategy defined. Machine Learning doesn't have automatic failover. Therefore, you must design a strategy that encompasses the workspace and all its dependencies, such as Key Vault, Azure Storage, and Azure Container Registry.

Recommendations

Expand table

Recommendation	Benefit
Multi-region model deployment: For enhanced reliability and availability, consider a multi-region deployment environment when possible.	A multi-region deployment ensures that your Machine Learning workloads continue to run even if one region experiences an outage. Multi-region deployment improves load distribution across regions, potentially enhancing performance for users located in different geographical areas. For more information, see Failover for business continuity and disaster recovery.
Model training resiliency: Use checkpointing features supported by Machine Learning including Azure Container for PyTorch, the TensorFlow Estimator class, or the Run object and the FileDataset class that support model checkpointing.	Model checkpointing periodically saves the state of your machine learning model during training, so that it can be restored in case of interruption, failure, or termination. For more information, see Boost checkpoint speed and reduce cost with Nebula.
Use the Dedicated virtual machine tier for compute clusters: Use the Dedicated virtual machine tier for compute clusters for batch inferencing to ensure your batch job isn't preempted.	Low-priority virtual machines come at a reduced price but are preemptible. Clusters that use the Dedicated virtual machine tier aren't preempted.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload.

The Security design principles provide a high-level design strategy for achieving those goals by applying approaches to the technical design around Machine Learning.

Design checklist

Start your design strategy based on the design review checklist for Security and identify vulnerabilities and controls to improve the security posture. Extend the strategy to include more approaches as needed.

- ✓ **Availability**: Reduce the attack surface of the Machine Learning workspace by restricting access to the workspace to resources within the virtual network.
- ✓ **Confidentiality**: Guard against data exfiltration from the Machine Learning workspace by implementing network isolation. Ensure access to all external

resources is explicitly approved and access to all other external resources isn't permitted.

- ✓ **Integrity**: Implement access controls that authenticate and authorize the Machine Learning workspace for external resources based on the least privilege principle.
- ✓ **Integrity**: Implement use case segregation for Machine Learning workspaces by setting up workspaces based on specific use cases or projects. This approach adheres to the principle of least privilege by ensuring that workspaces are only accessible to individuals that require access to data and experimentation assets for the use case or project.
- ✓ **Integrity**: Regulate access to foundational models. Ensure only approved registries have access to models in the model registry.
- ✓ **Integrity**: Regulate access to approved container registries. Ensure Machine Learning compute can only access approved registries.
- ✓ **Integrity**: Regulate the Python packages that can be run on Machine Learning compute. Regulating the Python packages ensures only trusted packages are run.
- ✓ **Integrity**: Require code used for training in Machine Learning compute environments to be signed. Requiring code signing ensures that the code running is from a trusted source and hasn't been tampered with.
- ✓ **Confidentiality**: Adhere to the principle of least privilege for role-based access control (RBAC) to the Machine Learning workspace and related resources, such as the workspace storage account, to ensure individuals have only the necessary permissions for their role, thereby minimizing potential security risks.
- ✓ **Integrity**: Establish trust and verified access by implementing encryption for data at rest and data in transit.

Recommendations

Expand table

Recommendation	Benefit
Security baseline: To enhance the security and compliance of your Machine Learning Service, apply the Azure security baseline for Machine Learning.	The security baseline provides tailored guidance on crucial security aspects such as network security, identity management, data protection, and privileged access. For optimal security, use

Recommendation	Benefit
	Microsoft Defender for Cloud to monitor these aspects.
Managed virtual network isolation: Configure managed virtual network isolation for Machine Learning. When you enable managed virtual network isolation, a managed virtual network is created for the workspace. Managed compute resources you create for the workspace automatically use this managed virtual network. If you can't implement managed virtual network isolation, then you must follow the network topology recommendations to separate compute into a dedicated subnet away from the rest of the resources in the solution, including the private endpoints for workspace resources.	Managed virtual network isolation enhances security by isolating your workspace from other networks, reducing the risk of unauthorized access. In a scenario in which a breach occurs in another network within your organization, the isolated network of your Machine Learning workspace remains unaffected, protecting your machine learning workloads.
Machine Learning network isolation: Configure a private endpoint for your Machine Learning workspace and connect to the workspace over that private endpoint.	Machine Learning network isolation enhances security by ensuring that access to your workspace is secure and controlled. With a private endpoint configured for your workspace, you can then limit access to your workspace to only occur over the private IP addresses.
Allow only approved outbound access: Configure the outbound mode on the Machine Learning workspace managed outbound access to `Allow only approved outbound` to minimize the risk of data exfiltration. Configure private endpoints, service tags, or fully qualified domain names (FQDNs) for resources that you need to access.	This configuration minimizes the risk of data exfiltration, improving data security. With this configuration enabled, a malicious actor who gains access to your system can't send your data to an unapproved external destination.
Virtual network isolation for dependent services: Configure dependent services, such as Storage, Key Vault, and Container Registry with private endpoints and disable public access.	Network isolation bolsters security by restricting access to Azure platform as a service (PaaS) solutions to private IP addresses only.
Managed identity: Use managed identities for authentication between Machine Learning and other services.	Managed identities improve security by eliminating the need to store credentials and manually manage and rotate service principals.

Recommendation	Benefit
Disable local authentication: Disable local authentication for Machine Learning compute clusters and instances.	Disabling local authentication increases the security of your Machine Learning compute and provides centralized control and management of identities and resource credentials.
Disable the public SSH port: Ensure the public Secure Shell (SSH) port is closed on the Machine Learning compute cluster by setting `remoteLoginPortPublicAccess` to `Disabled`. Apply a similar configuration if you use a different compute.	Disabling SSH access helps prevent unauthorized individuals from gaining access and potentially causing harm to your system and protects you against brute force attacks.
Don't provision public IP addresses for Machine Learning compute: Set enableNodePublicIp to `false` when provisioning Machine Learning compute clusters or compute instances. Apply a similar configuration if you use a different compute.	Refrain from provisioning public IP addresses to enhance security by limiting the potential for unauthorized access to your compute instance or clusters.
Get the latest operating system image: Recreate compute instances to get the latest operating system image.	Using the latest images ensures you're maintaining a consistent, stable, and secure environment, including ensuring you have the latest security patches.
Strict Machine Learning workspace access controls: Use Microsoft Entra ID groups to manage workspace access and adhere to the principle of least privilege for RBAC.	Strict workspace access controls enhance security by ensuring that individuals have only the necessary permissions for their role. A data scientist, for instance, might have access to run experiments but not to modify security settings, minimizing potential security risks.
Restrict model catalog deployments: Restrict model deployments to specific registries.	Restricting the deployments from the model catalog to specific registries ensures you only deploy models to approved registries. This approach helps regulate access to the open-source foundational models.
Encrypt data at rest: Consider using customer-managed keys with Machine Learning.	Encrypting data at rest enhances data security by ensuring that sensitive data is encrypted by using keys directly managed by

Recommendation	Benefit
	you. If you have a regulatory requirement to manage your own encryption keys, use this feature to comply with that requirement.
Minimize the risk of data exfiltration: Implement data exfiltration prevention. For example, create a service endpoint policy to filter egress virtual network traffic and permit data exfiltration only to specific Azure Storage accounts.	Minimize the risk of data exfiltration by limiting inbound and outbound requirements.

Advisor

The following are some examples of the Advisor security best practice recommendations for Machine Learning:

- Workspaces should be encrypted with a customer-managed key (CMK).
- Workspaces should use Azure Private Link.
- Workspaces should disable public network access.
- Compute should be in a virtual network.
- Compute instances should be recreated to get the latest software updates.

Azure Policy

The following are examples of built-in Azure Policy definitions for Machine Learning security:

- Configure allowed registries for specified Machine Learning computes.
- Configure allowed Python packages for specified Machine Learning computes.
- Machine Learning Workspaces should disable public network access.
- Machine Learning compute instances should be recreated to get the latest software updates.
- Machine Learning computes should be in a virtual network.
- Machine Learning computes should have local authentication methods disabled.
- Machine Learning workspaces should be encrypted with a CMK.
- Machine Learning workspaces should use Private Link.
- Machine Learning workspaces should use a user-assigned managed identity.
- Require an approval endpoint called prior to jobs running for specified Machine Learning computes.
- Require code signing for training code for computes.
- Restrict model deployment to specific registries.

Cost Optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget while meeting business requirements.

Read the Cost Optimization design principles to understand the approaches to achieve those goals and the necessary tradeoffs in technical design choices related to training and deploying models in their environments.

Design checklist

Start your design strategy based on the design review checklist for Cost Optimization for investments and fine tune the design so that the workload is aligned with the budget allocated for the workload. Your design should use the right Azure capabilities, monitor investments, and find opportunities to optimize over time.

- ✓ **Usage optimization**: Choose the appropriate resources to ensure that they align with your workload requirements. For example, choose between CPUs or GPUs, various SKUs, or low versus regular-priority VMs.
- ✓ **Usage optimization**: Ensure compute resources that aren't being used are scaled down or shut down when idle to reduce waste.
- ✓ **Usage optimization**: Apply policies and configure quotas to comply with the design's upper and lower limits.
- ✓ **Usage optimization**: Test parallelizing training workloads to determine if training requirements can be met on lower cost SKUs.
- ✓ **Rate optimization**: Purchase Azure Reserved Virtual Machine Instances if you have a good estimate of usage over the next one to three years.
- ✓ **Monitor and optimize**: Monitor your resource usage such as CPU and GPU usage when training models. If the resources aren't being fully used, modify your code to better use resources or scale down to smaller or cheaper VM sizes.

Recommendations

Expand table

Recommendation	Benefit
Optimize compute resources: Optimize your compute resources based on the requirements of your workload. Choose the SKU that best suits your workload: • General Purpose – Balanced CPU to memory ratio, good for all purposes. • Compute Optimized – High CPU to memory ratio, good for math-heavy computations. • Memory Optimized – High memory to CPU, good for in-memory computations or database applications. • M Series – Very large machines that have huge amounts of memory and CPU. • GPU – Better for models with a high number of variables that can benefit from higher parallelism and specialized core instructions. Typical applications are deep learning, image or video processing, scientific simulations, data mining, and taking advantage of GPU development frameworks. Test with multiple families and document the results as your baseline. As your model and data evolve, the most adequate compute resource might change. Monitor execution times and reevaluate as needed.	Selecting the right compute is critical as it directly impacts the cost of running your workload. Choosing a GPU or a high-performance SKU without proper usage can lead to wasteful spending, while choosing undersized compute can lead to prohibitively long training times and performance problems.
Optimize compute scaling: Configure your compute clusters for autoscaling to ensure you only use what you need. For training clusters, set the minimum number of nodes to 0 and configure the amount of time the node is idle to an appropriate time. For less iterative experimentation, reduce the time to save costs. For more iterative experimentation, use a higher time to prevent paying for scaling up or down after each change.	Configure autoscaling for compute clusters to scale down when their usage is low. Set the minimum number of nodes to 0 for training clusters to scale down to 0 when not in use.
Set training termination policies: Set early termination policies to limit the duration of training runs or terminate them early.	Setting termination policies can help you save costs by stopping nonperforming runs early.
Use low-priority virtual machines for batch workloads: Consider using low-priority virtual machines for batch workloads that aren't time-sensitive and in which interruptions are recoverable.	Low-priority virtual machines enable a large amount of compute power to be used for a low cost. They take advantage of surplus capacity in Azure.

Recommendation	Benefit
Enable idle shutdown for compute instances: Enable idle shutdown for compute instances or schedule a start and stop time if usage time is known.	By default, compute instances are available to you, accruing cost. Configuring compute instances to shut down when idle or configuring a schedule for them saves cost when they aren't in use.
Parallelize training workloads: Consider parallelizing training workloads. Test running them with the help of the parallel components in Machine Learning.	Parallel workloads can be run on multiple smaller instances, potentially yielding cost savings.
Azure Reserved VM Instances: Purchase Azure Reserved VM Instances if you have a good estimate of usage over the next one to three years. Take advantage of reserved capacity options for services when you have good estimates of usage.	Purchase Azure Reserved VM Instances to prepay for virtual machine usage and provide discounts with pay-as-you-go pricing. The discount is automatically applied for virtual machine usage that matches the reservation.

Operational Excellence

Operational Excellence primarily focuses on procedures for **development practices, observability, and release management**.

The Operational Excellence design principles provide a high-level design strategy for achieving those goals towards the operational requirements of the workload.

Design checklist

Start your design strategy based on the design review checklist for Operational Excellence for defining processes for observability, testing, and deployment related to Machine Learning.

- ✓ **Development standards**: Take advantage of Machine Learning model catalogs and registries to store, version, and share machine learning assets.
- ✓ **Automate for efficiency**: Follow good machine learning operations (MLOps) practices. When possible, build end-to-end automated pipelines for data preparation, training, and scoring processes. In development, use scripts instead of notebooks for training models, as scripts are easier to integrate into automated pipelines.

- ✓ **Deploy with confidence**: Implement infrastructure as code (IaC) for Machine Learning workspaces, compute clusters, compute instances, and other deployment environments.
- ✓ **Observability**: Monitor the performance of your deployed models including data drift.
- ✓ **Observability**: If your models are deployed to online endpoints, enable Application Insights to monitor online endpoints and deployments. Monitor training infrastructure to ensure you're meeting your baseline requirements.
- ✓ **Simplicity**: Use curated environments optimized for Machine Learning, when available.

Recommendations

Expand table

Recommendation	Benefit
Minimize Machine Learning workspace instances: Minimize the number of workspaces, when possible, to reduce maintenance.	Limiting the number of workspaces reduces the maintenance effort and cost of operation. For requirements, such as security, you might need multiple separate workspaces. Minimize the number of workspaces when possible.
Take advantage of model catalogs and registries: Take advantage of Machine Learning model catalogs and registries to store, version, and share machine learning assets. Use Machine Learning model catalogs to help you implement A/B testing and deployment of models.	Use Machine Learning model registries to store and version your machine learning models to track changes and maintain lineage with the job and datasets used for training. With Machine Learning model catalogs, your data science teams can discover, evaluate, and fine tune pretrained foundational machine learning models. Storing versioned models in Machine Learning model registries supports deployment strategies such as A/B releases, canary releases, and rollbacks.
Monitor model performance: Monitor the performance of your deployed models, and detect data drift on datasets.	Monitoring deployed models ensures your models meet the performance requirements. Monitoring data drift helps you detect changes in the input data that can lead to a decline in your model's performance. Managing data drift helps you ensure that your model provides accurate results over time.

Recommendation	Benefit
Monitor infrastructure: If your models are deployed to online endpoints, enable Application Insights to monitor online endpoints and deployments. Monitor training infrastructure to ensure you're meeting your baseline requirements. Ensure you're collecting resource logs for Machine Learning.	Monitoring endpoints gives you visibility into metrics such as request latency and requests per minute. You can compare your performance versus your baseline and use this information to make changes to compute resources accordingly. Monitoring metrics such as network bytes can alert you if you're approaching quota limits and prevent throttling. Likewise, monitoring your training environment provides you with the information to make changes to your training environment. Use that information to decide to scale in or out, scale up or down with different performant SKUs, or choose between CPUs or GPUs.
Curate model training environments: Use curated environments optimized for Machine Learning, when available.	Curated environments are pre-created environments provided by Machine Learning that speed up deployment time and reduce deployment and training latency. Using curated environments improves training and deployment success rates and avoids unnecessary image builds. Curated environments, such as Azure Container for PyTorch, can also be optimized for training large models on Machine Learning.

Performance Efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance.

The Performance Efficiency design principles provide a high-level design strategy for achieving those capacity goals against the expected usage.

Design checklist

Start your design strategy based on the design review checklist for Performance Efficiency for defining a baseline based on key performance indicators for Machine Learning workloads.

- ✓ **Performance targets**: Determine the acceptable training time and retrain frequency for your model. Setting a clear target for training time, along with testing, helps you

determine the compute resources, CPU versus GPU, and CPU SKUs required to meet the training time goal.

- ✓ **Performance targets**: Define the acceptable performance targets for your deployed models including response time, requests per second, error rate, and uptime. Performance targets act as a benchmark for your deployed model's efficiency. Targets can help you make CPU versus GPU determinations, CPU SKU choices, and scaling requirements.
- ✓ **Meet capacity requirements**: Choose the right compute resources for model training.
- ✓ **Meet capacity requirements**: Choose the right compute resources for model deployments.
- ✓ **Meet capacity requirements**: Choose deployment environments with autoscaling capabilities to add and remove capacity as demand fluctuates.
- ✓ **Achieve and sustain performance**: Continuously monitor the performance of your deployed models, review results, and take appropriate actions.
- ✓ **Achieve and sustain performance**: Continuously monitor the performance of your infrastructure of deployed models, review results, and take appropriate actions. Monitor training infrastructure to ensure you're meeting your requirements for training time.

Recommendations

Expand table

Recommendation	Benefit
Select appropriate compute services for model training: Consider Machine Learning compute clusters over compute instances for model training if you require autoscaling. Optimize your compute resources based on the training requirements. First choose between CPUs and GPUs. Default to CPUs, but consider GPUs for workloads such as deep learning, image or video processing, or large amounts of data. Next, choose the image SKU that best suits your workload. Use testing to choose the compute option that	Selecting the right compute is critical as it directly impacts the training time. Choosing the right SKU and CPU versus GPU ensures your model training can meet your requirements and performance targets. Choosing a low-performance SKU that's overused can lead to prohibitively long training times and performance problems. Compute clusters provide the ability to improve performance by scaling out workloads that support horizontal scaling. This method provides flexibility for handling workloads with

Recommendation	Benefit
optimizes cost against training time when determining your baseline.	different demands and lets you add or remove machines as needed.
Model deployment environment scaling: Use the deployment environment's autoscale capabilities. For AKS deployment environments, use the cluster autoscaler to scale to meet demand. For online endpoints, automatically scale via integration with the Azure Monitor autoscale feature.	Autoscaling adjusts the number of instances of the deployed model to match demand.
Monitor model performance: Monitor the performance of your deployed models.	Tracking the performance of models in production alerts you to potential problems such as data drift, prediction drift, data quality, and feature attribution drift. Monitoring data drift helps you detect changes in the input data that can lead to a decline in your model's performance. Managing data drift helps you ensure that your model provides accurate results over time.
Monitor infrastructure: Monitor online endpoints and integrate with Monitor to track and monitor the appropriate metrics and logs. Enable Application Insights when creating online deployments. Monitor training infrastructure and review resource usage such as memory and CPU or GPU usage when training models to ensure you're meeting your baseline requirements.	Monitoring endpoints gives you visibility into metrics such as request latency and requests per minute. You can compare your performance versus your baseline and use this information to make changes to compute resources accordingly. Monitoring metrics such as network bytes can alert you if you're approaching quota limits and prevent throttling. Likewise, monitoring your training environment provides you with the information to make changes to your training environment. Use that information to decide to scale in or out, scale up or down with different performant SKUs, or choose between CPUs or GPUs.

Azure policies

Azure provides an extensive set of built-in policies related to Machine Learning and its dependencies. Some of the preceding recommendations can be audited through Azure policies. Consider the following policies that are related to security:

- Allowed registries for specified Machine Learning computes.
- Configure allowed Python packages for specified Machine Learning computes.

- Machine Learning computes should be in a virtual network.
- Machine Learning computes should have local authentication methods disabled.
- Machine Learning workspaces should disable public network access.
- Machine Learning compute instances should be recreated to get the latest software updates.
- Machine Learning workspaces should be encrypted with a customer-managed key.
- Machine Learning workspaces should use private link.
- Machine Learning workspaces should use user-assigned managed identity.
- Require an approval endpoint called prior to jobs running for specified Machine Learning computes.
- Require code signing for training code for computes.
- Restrict model deployment to specific registries.

Consider the following policy that's related to cost optimization:

- Machine Learning Compute instance should have idle shutdown.

Consider the following policies that are related to operational excellence:

- Require log filter expressions and datastore to be used for full logs for specified Machine Learning computes.
- Resource logs in Machine Learning workspaces should be enabled.

For comprehensive governance, review the Azure Policy built-in definitions for Machine Learning.

Advisor recommendations

Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments. Advisor recommendations can help you improve the reliability, security, cost effectiveness, performance, and operational excellence of Machine Learning.

Consider the following Advisor recommendations for security:

- Workspaces should be encrypted with a customer-managed key (CMK).
- Workspaces should use private link.
- Workspaces should disable public network access.
- Compute should be in a virtual network.
- Compute instances should be recreated to get the latest software updates.

Consider the following Advisor recommendation for operational excellence:

- Resource logs in Machine Learning workspaces should be enabled.

Next steps

Consider these articles as resources that demonstrate the recommendations highlighted in this article.

- Use the baseline OpenAI end-to-end chat reference architecture as an example of how these recommendations can be applied to a workload.
- Use Machine Learning product documentation to build implementation expertise.

Feedback

Was this page helpful? Yes No

Architecture best practices for Azure NetApp Files

Article • 04/10/2025

Azure NetApp Files is an Azure-native, enterprise-class service that you can use to run file workloads in the cloud and store files. Use this fully managed, cloud-based service to:

- Choose service and performance levels.
- Manage data protection.
- Create Azure NetApp Files accounts.
- Create capacity pools.
- Create volumes.
- Create and manage file shares that have high performance, low latency, high availability, and scalability.

Use the familiar protocols and tools that you use for on-premises enterprise applications. Azure NetApp Files supports Server Message Block (SMB), Network File System (NFS), and dual-protocol volumes. You can use Azure NetApp Files for file sharing, high-performance computing, home directories, and databases.

This article assumes that as an architect, you reviewed the file storage options and chose Azure NetApp Files as the service for your workloads. The guidance in this article provides architectural recommendations that are mapped to the principles of the Well-Architected Framework pillars.

> **ⓘ Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern along with design strategies localized to the technology scope.
>
> Also included are *recommendations* for the technology capabilities that can help materialize those strategies. The recommendations don't represent an exhaustive list of all configurations that are available for Azure NetApp Files and its dependencies. Instead, they list the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept or to optimize your existing environments.
>
> Foundational architecture that demonstrates the key recommendations: **Moodle deployment with Azure NetApp Files**.

Technology scope

This review focuses on the interrelated decisions for the following Azure resource:

- Azure NetApp Files

Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover fast from failures**.

Reliability design principles provide a high-level design strategy applied for individual components, system flows, and the system as a whole.

Design checklist

Start your design strategy based on the design review checklist for Reliability. Determine its relevance to your business requirements while keeping in mind the scalability and performance of your workloads. Extend the strategy to include more approaches as needed.

- ✓ **Design your workload to align with business objectives and avoid unnecessary complexity or overhead.** Use a practiced and balanced approach to make decisions that reflect the needs of your workload. Your Azure NetApp Files deployment choices can affect other components. For example, the subnet size for your Azure NetApp Files deployment determines the number of available IP addresses. And your network features setting determines the available networks and features.

- ✓ **Identify user flows and system flows.** Understand customer needs and business requirements so that you can effectively plan for reliability and optimize security. To limit access to and from necessary networks, use the principle of least privilege when you assign permissions. To authorize access to Azure NetApp Files data, use features or services such as network security groups (NSGs), Microsoft Entra ID hybrid identities, Microsoft Entra Domain Services, Active Directory Domain Services (AD DS), and Lightweight Directory Access Protocol (LDAP). To restrict default access to Azure NetApp Files volumes, use features such as file locking.

- ✓ **Define reliability targets and recovery targets.** Visualize recovery targets and drive actions to achieve reliability goals and recoverability goals for your workload. To improve reliability and recovery, define these targets and develop an understanding of Azure NetApp Files solutions. Targets help optimize snapshots, high availability between availability zones, cross-zone and cross-region replication, and SMB continuous availability for supported applications.

- ✓ **Build redundancy.** Deploy your workload across availability zones and regions to build redundancy in the workload and supporting infrastructure. This approach ensures that you can quickly recover from failures. Active-passive deployments can handle production loads only in the primary region, but the deployment fails over to the secondary passive region when necessary.

Recommendations

Expand table

Recommendation	Benefit
Use future-proof sizing to appropriately determine your delegated subnet size.	Appropriately size your subnet to ensure the long-term resiliency of your design and the scalability of your applications and workloads.
Enable standard networking features on your delegated subnet for enhanced resiliency and extra connectivity patterns.	Select standard network features so you can have high IP limits and standard virtual network features, including extra connectivity patterns and NSGs and user-defined routes on delegated subnets.
Use built-in Azure NetApp Files features to build redundancy and ensure the reliability of recovery targets. Use space-efficient snapshots for primary data protection of business-critical data and enhance recoverability with short recovery time objective (RTO) and recovery point objective (RPO). Add long term retention with Azure NetApp Files backup. Use the Azure NetApp Files availability zone volume placement feature to deploy volumes in a specific availability zone so that they align with Azure compute and other services in the same zone. Use cross-zone or cross-region replication to design for local or remote disaster recovery. To protect your data from zonal or regional failures, use snapshot technology so you can replicate your Azure NetApp Files instances across specific Azure availability zones or regions.	Snapshots add stability, scalability, and fast recoverability without affecting performance. They provide the foundation for other redundancy solutions, including backup, cross-region replication, and cross-zone replication. Azure availability zones are highly available, fault tolerant, and more scalable than traditional single or multiple datacenter infrastructures. To design resilient solutions, use Azure services that support availability zones. Cross-zone and cross-region replication minimizes data transfer, which reduces cost, and replicates only changed blocks across regions, which creates a low restore point objective.
Ensure application resilience during service maintenance events, for example platform, service, or	Some applications might require tuning to handle input/output pauses for up to 30-45 seconds. To maintain application performance

Recommendation	Benefit
software upgrades. For more information, see Address application disruptions.	and continuity, familiarize yourself with the application's resiliency settings so you can handle storage service maintenance events.
For SMB or dual-protocol workloads, enable continuous availability for SMB to ensure availability during maintenance events.	If you create an SMB volume, continuous availability allows for SMB transparent failover for maintenance without interrupting the server applications storing their data on the volume. This feature is only available for certain workloads.
Use the right performance tier, volume sizes, and quality of service (QoS) settings to ensure proper performance. For volumes that have automatic QoS settings, the quota that's assigned to the volume and the selected service level determine the throughput limit. For volumes that have manual QoS settings, you can define the throughput limit individually. Understand how your workload's needs compare to Azure NetApp Files offerings. For more information, see Performance considerations.	Choose the correct performance tier to avoid unnecessary complexity and overhead and to help manage the total cost of ownership (TCO).
Configure application filtering properly for your firewalls. To ensure that you use the correct configuration, see the following resources: - Understand network-attached storage (NAS) protocols - Guidelines for network planning - Understand guidelines for AD DS site design and planning	Configure application filtering correctly to prevent problems such as dropped connections when you attempt to search directories or create items. Windows Server Active Directory, LDAP, and SMB filtering commonly have these problems. Use application filtering to ensure seamless connectivity and efficient network operations.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload.

The Security design principles provide a high-level design strategy for achieving those goals by applying approaches to the technical design of your file storage configuration.

Design checklist

Start your design strategy based on the design review checklist for Security. Identify vulnerabilities and controls to improve the security posture. Extend the strategy to include

more approaches as needed.

- ✓ **Isolate, filter, and control network traffic across both ingress and egress flows.** Control network traffic to mitigate potential security events. You can filter traffic between Azure resources with an NSG. Or use user-defined routes on the Azure NetApp Files delegated subnet to limit traffic.
- ✓ **Implement strict, conditional, and auditable identity and access management.** Strict access to resources provides a frontline of defense. To protect your data resources, use custom role-based access control (RBAC) roles, protocol-based locking tools, and built-in storage policies.
- ✓ **Establish a security baseline** that aligns with compliance requirements, industry standards, and platform recommendations. Storage policies and key management can help you establish a baseline of security with industry-standard tools. Azure NetApp Files also supports many of the built-in security recommendations for NFS, SMB, and dual-protocol volumes.
- ✓ **Use modern, industry-standard methods to encrypt data.** Ensure that your data remains secure. Azure NetApp Files provides solutions and features to ensure data security, including double encryption at rest. For Windows Server Active Directory users, Azure NetApp Files supports Advanced Encryption Standard (AES) encryption.
- ✓ **Develop a security strategy.** Azure NetApp Files supports NFS, SMB, and dual-protocol volumes. If you use dual-protocol volumes, understand the security needs of those volumes and determine which security style suits your workload's needs. If you need a Windows Server Active Directory connection for SMB, NFSv4.1 Kerberos, or LDAP lookups, you should align that security baseline to compliance requirements, industry standards, and platform recommendations. Azure NetApp Files uses standard CryptoMod to generate AES-256 encryption keys, which you can apply to your SMB server. To develop an appropriate security strategy, enable LDAP over TLS, encrypt SMB connections to the domain controller, assign administrator privileged users for SMB volumes, and ensure that you have a backup policy and security protocols.
- ✓ **Set appropriate access and ownership configurations.** Set user privileges and restricted roles to help mitigate mistakes and improper actions. To maintain security, set appropriate share-access permissions, ownership roles, and the ownership mode for shares, files, and folders. To achieve optimal security and mitigate mistakes, identify and understand the various access management solutions for NFS, SMB, and dual-protocol shares.

Recommendations

Expand table

Recommendation	Benefit
Implement access management. Use a custom RBAC role for Azure NetApp Files to limit access for operations. For more information, see Azure custom roles. Review the list of Azure NetApp Files resource provider operations. Restrict default access to Azure NetApp Files volumes. Lock Azure subscriptions, resource groups, and resources to protect them from accidental user deletions and modifications.	Lock resources so that users don't accidentally create or delete new volumes. You can also prevent users from creating snapshots or backups.
Set appropriate protocol-specific access controls. Enable access control lists (ACLs) on NFSv4.1 volumes if you create an NFSv4.1 or dual-protocol volume. Restrict mount path change permissions. For more information, see Configure Unix permissions and change the ownership mode for NFS and dual-protocol volumes. For SMB or dual-protocol volumes, use ACLs and SMB to control mounting and access via SMB to the shares. For more information, see Modify share-level ACLs with Microsoft Management Console and Manage SMB share ACLs. Use the New Technology File System (NTFS) security style with SMB shares. For more information, see Access control overview.	ACLs provide granular file security in NFSv4.1 and SMB. With NTFS, you can use NTFS ACLs. Use ACLs to limit access to necessary users.
Set appropriate share-access permissions, ownership, and the ownership mode for NFS shares. NFS uses an export policy to control volume access permissions via IP addresses. For more information, see the following resources: - Understand NAS share permissions - Configure an export policy for NFS or dual-protocol volumes - Configure Unix permissions and change the	Share-access permissions limit who can mount an NFS volume, the volume permissions, and who can change the volume owner. The file permissions control access to specific files and folders in a file system and are more granular than share permissions.

Recommendation	Benefit
ownership mode The owner and ownership mode dictates who owns and who can change the ownership of the volume. Set appropriate file access permissions for files and folders within NFS shares. For more information, see Understand NAS file permissions and Understand mode bits.	
Use a built-in or custom storage policy to restrict insecure volumes or snapshots. For more information, see Azure Policy definitions.	Restrict insecure volumes and use a policy to create snapshots to maintain the reliability and integrity of your applications.
Configure standard network features and an NSG on the Azure NetApp Files delegated subnet to filter network traffic to or from the Azure NetApp Files endpoint. Consider limiting the routes and deploying user-defined routes on the Azure NetApp Files delegated subnet or network to allow only necessary traffic.	An NSG and user-defined routes prevent unwanted or potentially malicious traffic from accessing Azure NetApp Files. User-defined routes help direct traffic only where you need it and to drop unnecessary traffic.
Adopt an effective strategy for your account keys. To mitigate risk, rotate the account keys periodically. To manage your own keys for volumes, use customer-managed keys.	Rotate account keys to help deter access from malicious actors. Use customer-managed keys if you have a security requirement or regulation for self-managed keys.
Reduce the visibility of resources. If you configure a snapshot policy or take snapshots, hide the snapshot path for NFSv3 and SMB volumes. By default, the snapshot directory path is hidden from NFSv4.1 clients. For NFSv4.1 and dual-protocol volumes, disable the showmount functionality. For SMB, enable access-based enumeration and nonbrowsable SMB shares.	Limit the visibility of resources to limit access. When you hide the snapshot path, you hide the directory path from NFSv3 and SMB clients, which adds an extra layer of protection. The directory remains accessible. The showmount functionality on NFS clients enables users to see exported file systems on an NFS server, but this feature can flag security probes as a vulnerability. You must enable the showmount functionality on some applications. In SMB deployments, access-based enumeration creates a perimeter on your data, which prevents unnecessary access to the share. Nonbrowsable SMB shares prevent the Windows client from browsing the SMB share. The share doesn't show up in the Windows file browser or in the list of shares when you run the `net view \\server /all` command.

Recommendation	Benefit
Decide which security style to implement if you have dual-protocol volumes. SMB and NFS use different permission models for user access and group access. With dual-protocol volumes, configure Azure NetApp Files to use either the UNIX or the NTFS security style. Only one security style can be active.	Understand the needs of your dual-protocol workloads to help optimize security.
Use double encryption for sensitive data at rest. By default, Azure NetApp Files encrypts data at rest. To configure double encryption, set the encryption type to *double* when you create a capacity pool.	Double encryption encrypts data at the hardware level (encrypted SSD drives) and at the software layer that's on the volume level.
Enable AES encryption if you establish a connection to a Windows Server Active Directory server. This option enables AES encryption authentication support for the admin account of the Windows Server Active Directory connection. For more information, see Create and manage Windows Server Active Directory connections.	AES encryption is a native and industry-standard platform encryption method that helps protect your data.
Enable LDAP over TLS if you establish a connection to a Windows Server Active Directory server and you use LDAP. For more information, including restrictions about when you can use LDAP, see the following resources: - LDAP encryption - Create and manage Windows Server Active Directory connections - Configure AD DS LDAP over TLS	LDAP over TLS secures communication between an Azure NetApp Files volume and the Windows Server Active Directory LDAP server.
Encrypt SMB connections to the domain controller. For more information, see Create and manage Windows Server Active Directory connections.	Encrypted domain controller connections only use the SMB3 protocol.
Determine and configure backup policy users, security privilege users, and administrators. If you establish a connection to a Windows Server Active Directory server, add the backup policy users, security privilege users, and administrators. For more information, see Create and manage Windows Server Active Directory connections.	This option grants extra security privileges to AD DS domain users or groups.
Enable NFSv4.1 Kerberos for NFSv4.1 and dual-protocol volumes.	Depending on your encryption level, Kerberos offers initial authentication, integrity checking, and privacy via a Kerberos ticket exchange. For more

Recommendation	Benefit
	information, see Understand data-in-transit encryption.
Enable SMB3 protocol encryption for in-flight SMB3 data. For more information, see the following resources: - SMB shares - Understand data-in-transit encryption - SMB encryption	SMB clients that don't use SMB3 encryption can't access a volume that has this setting enabled. Data at rest is encrypted regardless of this setting.

Cost optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget while meeting business requirements.

The Cost Optimization design principles provide a high-level design strategy for achieving those goals and making tradeoffs as necessary in the technical design related to Azure NetApp Files and its environment.

Design checklist

Start your design strategy based on the design review checklist for Cost Optimization for investments. Fine-tune the design so that the workload is aligned with the budget that's allocated for the workload. Your design should use the right Azure capabilities, monitor investments, and find opportunities to optimize over time.

- ✓ **Optimize component costs.** Determine the specifics of your storage account and design your Azure NetApp Files capacity pools accordingly to optimize costs and lower your TCO. To improve performance and prevent unnecessary costs, appropriately size your capacity pools to your needs and consolidate volumes that are in larger capacity pools with the appropriate QoS settings and performance tier. Use the Azure NetApp Files cool access feature to tier infrequently accessed data to lower-cost Azure storage.
- ✓ **Optimize data costs.** Use the Azure NetApp Files cool access feature to tier infrequently accessed data to lower-cost Azure storage. Azure NetApp Files offers a storage with cool access cost savings estimator to help estimate potential cost savings.
- ✓ **Align usage to billing increments.** Use Azure NetApp Files reservations to significantly reduce your capacity costs for storing data in your Azure NetApp Files volumes. Azure NetApp Files reservations offer a discount on capacity for storage costs when you commit to a reservation for one or three years.

- ✓ **Optimize environment costs.** Production workloads require proper data protection and disaster recovery. Azure NetApp Files offers several built-in options, including snapshots that provide space-optimized restore points, backups that efficiently protect data on lower-cost Azure storage tiers, and cross-region and cross-zone replication. To help optimize the cost of your deployment and ensure disaster preparedness, understand how each option suits your workload.
- ✓ **Understand and calculate pricing.** Understand your business needs so that you understand pricing. Azure NetApp Files offers several tools, including the Azure NetApp Files performance calculator, to help you accurately estimate pricing.
- ✓ **Optimize scalability costs.** Use Azure NetApp Files to meet your shifting business needs and respond to changes in workloads. For example, you might move volumes to improve performance and lower costs. You can use dynamic volume sizing to efficiently scale your Azure NetApp Files volumes to meet performance and capacity requirements on demand.
- ✓ **Use partner solutions to optimize costs.** Azure NetApp Files integrates with products like Azure VMware Solution and Microsoft SQL Server and is optimized for Oracle and SAP workloads. Understand Azure NetApp Files features and benefits to optimize your deployments and reduce the TCO. For example, you can use Azure NetApp Files data stores to increase storage capacity in Azure VMware Solution without using extra Azure VMware Solution nodes. You can also use the TCO estimator to calculate estimated costs for Azure VMware Solution and Azure NetApp Files.

Recommendations

Consider the following recommendations to optimize cost when you configure your Azure NetApp Files account.

Expand table

Recommendation	Benefit
Use the right performance tier, volume sizes, and QoS settings to improve performance and prevent unnecessary costs. For volumes that have manual QoS settings, you can define the throughput limit individually. To plan for performance in Azure NetApp Files, understand performance tiers and QoS settings.	The quota that's assigned to the volume and the selected service level determine the throughput limit for a volume that has automatic QoS settings. Ensure that you use the appropriate performance tier and QoS settings to help maintain performance.
Use Azure NetApp Files storage with cool access to transparently tier infrequently accessed data to lower-cost Azure storage. Configure a coolness period to have infrequently accessed data blocks be transparently moved from Azure NetApp Files to an Azure storage account and recalled upon access.	Cool storage uses a lower-cost storage tier, which reduces the cost of storage. The transition to the cool tier is seamless for users and applications.

Recommendation	Benefit
Use Azure NetApp Files reservations to save money on the storage cost for Azure NetApp Files with reservations.	Azure reservations can significantly reduce your capacity costs for storing data in your Azure NetApp Files volumes. How much you save depends on the total capacity you choose to reserve and your chosen service level.
Leverage the compounding benefits of Azure NetApp Files storage with cool access, space-efficient snapshots, and reserved capacity to reduce the effective price of Azure NetApp Files.	These three features can work together or independently to significantly reduce the effective price of Azure NetApp Files. Effective pricing reflects the actual cost you pay for storage after applying built-in efficiencies and discounts. These savings directly reduce the cost per GiB per month, enabling you to manage your storage expenses more effectively.
Size capacity pools appropriately without introducing unnecessary overhead or free space. You don't need to maintain free space in a capacity pool. Only increase capacity pool sizes when you need to increase a volume's size or create a new volume. Reduce capacity pool sizes when you delete volumes or reduce a volume size.	Appropriately size capacity pools to ensure that you only pay for the storage that you need.
Consolidate as many volumes as possible within a few large capacity pools to share resources. Take advantage of manual QoS to rightsize performance and capacity. Use Azure NetApp Files to share a single capacity pool between multiple volumes from disparate workloads. Set capacity pools to manual QoS so that you can configure each volume with a custom size and throughput. For more information, see Throughput limit examples of volumes in a manual QoS capacity pool.	Combine as many volumes as possible within the same capacity pool to optimize resources and costs.
Use native snapshots to create space-optimized restore points.	Use Azure NetApp Files snapshots to create space-efficient and cost-efficient restore points.
Use Azure NetApp Files backup to efficiently protect data on low-cost Azure storage. Azure NetApp Files supports a fully managed backup solution for long-term recovery, archives, and compliance.	Backups are incremental and fully managed to reduce the TCO.
Take advantage of cross-region or cross-zone replication as a cost-optimized disaster recovery solution.	Azure NetApp Files provides a fully managed and cost-optimized disaster recovery solution that has efficient replication technology. You can replicate

Recommendation	Benefit
	data between zones or regions, without needing to stand up infrastructure.
Implement dynamic volume shaping to provide on-demand performance and capacity only when needed. For a comparison of static volume shaping and dynamic volume shaping cost models, see Cost models. To determine whether Azure NetApp Files data stores can reduce your TCO when you use them with Azure VMware Solution, use the Azure NetApp Files for Azure VMware Solution TCO estimator.	Dynamic volume shaping provides nondisruptive volume resizing and service-level changes to meet changing performance requirements in real time. Ensure that you only pay for performance and capacity when you need it.
Use Azure NetApp Files data stores when you deploy Azure VMware Solution.	Use Azure VMware Solution so that you can add Azure NetApp Files volumes to the data store capacity, which expands the capacity beyond what the virtual storage area network (vSAN) provides. You can increase storage capacity without extra Azure VMware Solution nodes.
Use Azure NetApp Files to deploy small virtual machines (VMs) and take advantage of high network throughput limits when you deploy SQL Server. You can use Azure NetApp Files volumes as an SMB share to host SQL Server databases. For more information, see Benefits of using Azure NetApp Files for SQL Server deployments.	Azure VMs typically have a higher network throughput versus disk throughput. With Azure NetApp Files, you can use small Azure VM sizes, which reduce the TCO.
Use Azure NetApp Files to deploy small VMs and take advantage of high network throughput limits when you deploy SAP workloads. This approach reduces the TCO. For more information, see SAP on Azure NetApp Files sizing best practices.	Use Azure NetApp Files volumes with various SAP workloads. Azure VMs typically have a higher network throughput versus disk throughput. With Azure NetApp Files, you can use small Azure VM sizes, which reduce the TCO.
Use Azure NetApp Files to deploy small VMs and take advantage of higher network throughput limits when you deploy Oracle database workloads. This approach reduces the TCO. For more information, see Run demanding Oracle workloads in Azure.	Use Azure NetApp Files volumes with various Oracle database workloads. Azure VMs typically have a higher network throughput versus disk throughput. With Azure NetApp Files, you can use small Azure VM sizes, which reduce the TCO.
Use Azure NetApp Files when you deploy Siemens Teamcenter product lifecycle management (PLM) software. This approach reduces the TCO.	Deploy Azure NetApp Files as a storage solution for Siemens Teamcenter PLM. You can use several cost optimization features to lower the TCO.

Operational excellence

Operational Excellence primarily focuses on procedures for **development practices, observability, and release management**.

The Operational Excellence design principles provide a high-level design strategy for achieving those goals for the operational requirements of the workload.

Design checklist

Start your design strategy based on the design review checklist for Operational Excellence for defining processes for observability, testing, and deployment related to your file storage configuration.

- ✓ **Develop the proper architecture and design for your workloads** to optimize performance and security. Understand LDAP, application filtering, and the Windows Server Active Directory connector to successfully deploy Azure NetApp Files. To ensure that you protect your data, understand other features such as snapshots.
- ✓ **Deploy with confidence.** Understand and manage limits to help optimize your architecture for Azure NetApp Files. Solutions such as Azure Resource Manager templates (ARM templates) can help you automate your deployment. And you can use test environments that you clone from existing workloads to do dry runs that use real data and scenarios. Use Azure NetApp Files built-in tools to increase your confidence in your deployment.
- ✓ **Monitor your routine operations** to help optimize performance and better understand various workloads. Azure NetApp Files provides performance and capacity management tools, Azure-native monitoring features, and tools to manage your regional quotas and resource limits. Regularly test your production environment to adjust performance targets and optimize performance. Use snapshot-based cloning and other features in Azure NetApp Files to simulate your production workloads and environments and optimize your performance.

Recommendations

Consider the following recommendations to optimize operational excellence when you configure your Azure NetApp Files account.

Expand table

Recommendation	Benefit
Understand the Windows Server Active Directory connector, its options, and	Azure NetApp Files features such as SMB volumes, dual-protocol volumes, and NFSv4.1 Kerberos volumes integrate

Recommendation	Benefit
what the options do. Solution architectures that use Azure NetApp Files volumes require proper AD DS design and planning.	with AD DS. Understand AD DS deployment to help optimize your Azure NetApp Files experience.
Understand LDAP. LDAP provides a general-purpose, network-based directory service that you can use across diverse platforms to locate network objects.	LDAP is robust, scalable, and secure. You can use LDAP to contain millions of user objects and group objects. With Windows Server Active Directory, you can use multiple servers to replicate across multiple sites to improve performance and resiliency. The only LDAP server that Azure NetApp Files supports is Windows Server Active Directory. It supports both AD DS and Microsoft Entra Domain Services.
Use native monitoring features to monitor the performance and health of application volumes, file shares, and databases.	Use monitoring features to set up alerts and notifications for critical events such as file system capacity and performance problems. Use alerts to take corrective actions proactively.
Manage performance. Azure NetApp Files offers nondisruptive on-demand capacity scaling and service-level changes.	Use these features to scale up or scale down quickly when applications, file shares, and databases require adjustments. To help manage the performance and availability of your application, combine these capabilities with volume resizing or manual QoS settings.
Use Azure NetApp Files storage with cool access to transparently tier infrequently accessed data blocks to lower-cost Azure storage.	By configuring cool access and moving inactive data from Azure NetApp Files storage to an Azure storage account, you can optimize costs while maintaining accessibility and performance for active data. Azure NetApp Files storage with cool access offers significant benefits by seamlessly tiering infrequently accessed data blocks to lower-cost Azure storage. This feature is particularly advantageous for managing cold data, which is often unstructured and can account for over 60% of total storage capacity in many environments. By integrating cool access directly within the service, it eliminates the need for additional tools, making it easier to optimize costs. This approach is ideal for inactive data, large volumes of unstructured data, snapshot data, sequential read operations, and metadata-intensive workloads like chip design, version control systems, and home directories.
Use advanced snapshot data-protection and data-management capabilities. These features provide the foundation for data-protection solutions, including single-file restores, volume restores, clones, cross-region replication, and long-term retention.	Azure NetApp Files snapshot technology provides stability, scalability, and fast recoverability without affecting performance.

Recommendation	Benefit
Automate infrastructure deployments. Use ARM templates to define and deploy the resources that support your application, such as file shares, storage accounts, VMs, and databases. Also consider using infrastructure as code via Terraform to configure and deploy cloud infrastructure. For more information, see Deploy Azure NetApp Files by using Terraform and Terraform Azure Resource Manager and Azure NetApp Files account integration.	Use Azure NetApp Files to automate the deployment of your application infrastructure, file shares, and databases, which improves the deployment's efficiency and supports a lift-and-shift approach.
Test deployments. Use cloning via snapshot restore to quickly create a new volume for testing.	Azure NetApp Files provides a reliable and scalable platform to deploy your application code, files, and databases. Use cloning to ensure that your application remains up to date and stable and to test new releases or upgrades easily.
Test environments. You can use Azure NetApp Files to create and manage test environments quickly. Use snapshot restore to clone your production data to a new volume.	Test activities remain separate from production environments. Do testing to add guardrails. For example, you can ensure the effectiveness of a disaster recovery solution. Use test environments to test new releases or upgrades in a nondisruptive manner. Combine test environments with cross-zone or cross-region replication for further optimization. You can repurpose data that you typically replicate for disaster recovery only.
Manage your regional quotas and resource limits. Review those levels regularly as your workload evolves. To increase your regional quota, request a regional capacity quota increase. To increase resource limits, request a limit increase.	Quotas affect cost and performance. Plan the amount of resources that your workload requires. Review that level regularly as the workload evolves. Understand these requirements to maintain performance and capacity as needs change.

Performance efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance.

The Performance Efficiency design principles provide a high-level design strategy for achieving those capacity goals against the expected usage.

Design checklist

Start your design strategy based on the design review checklist for Performance Efficiency. Define a baseline that's based on key performance indicators for Azure NetApp Files.

- ✓ **Define performance targets.** Understand the demands of your workload and assign numerical values to your performance targets. Azure NetApp Files offers tools and resources to quantify these demands, including calculators and formulas to convert throughput to input/output operations per second (IOPS). You should also understand how Azure NetApp Files service levels and performance tiers affect your deployment and meet the needs of your workload.
- ✓ **Conduct capacity planning.** Understand the capacity requirements of your datasets so that you can plan for and optimize performance. Before you deploy your application, understand the nature of your workload and understand the resource limits of Azure NetApp Files. Ensure that Azure NetApp Files capabilities can handle your specific needs to effectively plan for your performance requirements. Make configuration choices that meet your performance and capacity needs.
- ✓ **Select the right service.** When you define the needs of your Azure NetApp Files deployment, understand the different performance, capacity, data protection, and disaster recovery requirements. Based on your requirements, calibrate Azure NetApp Files to meet your specific throughput and general performance needs. In some cases, you can reduce storage costs.
- ✓ **Continually optimize performance.** Monitor your volume performance to understand the shifting demands of your production workloads. Use these monitoring insights to optimize and tune your performance.

Recommendations

Expand table

Recommendation	Benefit
Review and understand the Azure NetApp Files service levels, which provide performance ceilings for Azure NetApp Files volumes.	When performance needs for a workload change, you can dynamically and seamlessly change the service levels of the volume to reduce cost.
Understand resource limits of Azure NetApp Files.	Understand the resource limits of Azure NetApp Files to reduce the risk of overprovisioning or overtaxing your volume.
Understand the nature of your workload. Convert throughput to IOPS so that you can get insights into performance. For more information,	Understand the IOPS, throughput, and latency requirements of your workload to determine what service level and volume capacity you require.

Recommendation	Benefit
see Performance benchmark test recommendations. Use the Azure NetApp Files performance calculator to understand how your needs match with Azure NetApp Files.	Use the Azure NetApp Files performance calculator to choose the correct capacity and service level so that you can size your volume to maximize your performance and cost efficiency.
Understand the capacity requirements of the dataset, and determine the appropriate QoS type.	Volume quotas and QoS settings affect performance and volume capacity. If you have a small dataset but high performance requirements, you can use manual QoS to overprovision the volume's performance.
Understand the difference between regular volumes and large volumes and the difference between regular volume performance and large volume performance.	Large volumes can provide more capacity and performance but come with minimum requirements that might not fit your workload. In the right context, large volumes can streamline your operation.
Configure your capacity pool for your workload's capacity, throughput, and QoS policy.	Azure NetApp Files volumes use automatic QoS policies by default and are tied to the volume capacity. You can use manual QoS to configure the capacity pool and provide high performance for volumes. Manual QoS uses the capacity pool size for its performance levels. Understand how capacity pools work so that you can tune Azure NetApp Files to deliver the optimal performance for your business needs.
Optimize and tune Azure NetApp Files performance. Review the performance FAQ for Azure NetApp Files.	Optimize performance through network and VM configurations.
Consider data protection and disaster recovery options.	Data protection and disaster recovery can promote consistent performance because they provide a secondary site that can properly accommodate your workload's performance needs if outages occur.
Monitor your Azure NetApp Files volume performance.	Monitor the performance of your Azure NetApp Files volumes to get insights into your workload's performance. Determine whether you need to adjust service levels or QoS policies for greater or lower performance.

Azure policies

Azure provides an extensive set of built-in policies related to Azure NetApp Files. Some of the preceding recommendations can be audited through Azure policies. Consider the following policies that are related to security:

- Azure NetApp Files SMB volumes should use SMB3 encryption
- Azure NetApp Files NFSv4.1 volumes should use Kerberos data encryption
- Azure NetApp Files NFSv4.1 volumes should use Kerberos data integrity or data privacy
- Azure NetApp Files volumes shouldn't use the NFSv3 protocol type

Azure Advisor recommendations

Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure NetApp Files deployments. Here are some recommendations that can help you improve the reliability, security, cost effectiveness, performance, and operational excellence of Azure NetApp Files.

Consider the following Advisor recommendation for cost effectiveness:

- Save on-demand costs with reserved capacity.

Consider the following Advisor recommendations for reliability:

- Implement disaster recovery strategies for your Azure NetApp Files resources.
- Enable continuous availability for SMB volumes.
- Review SAP configurations for time out values that you use with Azure NetApp Files.

Next steps

- Azure NetApp Files documentation
- Storage hierarchy of Azure NetApp Files
- Solution architectures that use Azure NetApp Files

Azure Well-Architected Framework perspective on Azure OpenAI Service

Article • 03/14/2024

Azure OpenAI Service provides REST API access to OpenAI large language models (LLMs), adding Azure networking and security capabilities. This article provides architectural recommendations to help you make informed decisions when you use Azure OpenAI as part of your workload's architecture. The guidance is based on the Azure Well-Architected Framework pillars.

> **Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern along with design strategies localized to the technology scope.
>
> Also included are *recommendations* on the technology capabilities that can help materialize those strategies. The recommendations don't represent an exhaustive list of all configurations available for Azure OpenAI and its dependencies. Instead, they list the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept or optimize your existing environments.
>
> Foundational architecture that demonstrates the key recommendations: Baseline OpenAI end-to-end chat reference architecture.

Technology scope

This review focuses solely on Azure OpenAI.

Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover quickly from failures**.

The Reliability design principles provide a high-level design strategy applied for individual components, system flows, and the system as a whole.

Design checklist

Start your design strategy based on the design review checklist for Reliability. Determine its relevance to your business requirements. Extend the strategy to include more approaches as needed.

- ✓ **Resiliency**: Choose the appropriate deployment option of either pay-as-you-go or provisioned throughput based on your use case. Because reserved capacity increases resiliency, choose provisioned throughput for production solutions. The pay-as-you-go approach is ideal for dev/test environments.
- ✓ **Redundancy**: Add the appropriate gateways in front of your Azure OpenAI deployments. The gateway must have the capability to withstand transient failures like throttling and also route to multiple Azure OpenAI instances. Consider routing to instances in different regions to build regional redundancy.
- ✓ **Resiliency**: If you're using provisioned throughput, consider also deploying a pay-as-you-go instance to handle overflow. You can route calls to the pay-as-you-go instance via your gateway when your provisioned throughput model is throttled. You can also use monitoring to predict when the model will be throttled and preemptively route calls to the pay-as-you-go instance.
- ✓ **Resiliency**: Monitor capacity usage to ensure you aren't exceeding throughput limits. Regularly review capacity usage to achieve more accurate forecasting and help prevent service interruptions due to capacity constraints.
- ✓ **Resiliency**: Follow the guidance for large data files and import the data from an Azure blob store. Large files, 100 MB or larger, can become unstable when uploaded through multipart forms because the requests are atomic and can't be retried or resumed.
- ✓ **Recovery**: Define a recovery strategy that includes a recovery plan for models that are fine-tuned and for training data uploaded to Azure OpenAI. Because Azure OpenAI doesn't have automatic failover, you must design a strategy that encompasses the entire service and all dependencies, such as storage that contains training data.

Recommendations

Expand table

Recommendation	Benefit
Monitor rate limits for pay-as-you-go: If you're using the pay-as-you-go approach, manage rate limits for your model deployments and monitor usage of tokens per minute (TPM) and requests per minute (RPM).	This important throughput information provides information required to ensure that you assign enough TPM from your quota to meet the demand for your deployments. Assigning enough quota prevents throttling of calls to your deployed models.
Monitor provision-managed utilization for provisioned throughput: If you're using the provisioned throughput payment model, monitor provision-managed utilization.	It's important to monitor provision-managed utilization to ensure it doesn't exceed 100%, to prevent throttling of calls to your deployed models.
Enable the dynamic quota feature: If your workload budget supports it, perform overprovisioning by enabling dynamic quota on model deployments.	Dynamic quota allows your deployment to consume more capacity than your quota normally does, as long as there's available capacity from an Azure perspective. Extra quota capacity can potentially prevent undesired throttling.
Tune content filters: Tune content filters to minimize false positives from overly aggressive filters.	Content filters block prompts or completions based on an opaque risk analysis. Ensure content filters are tuned to allow expected usage for your workload.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload.

The **Security design principles** provide a high-level design strategy for achieving those goals by applying approaches to the technical design around Azure OpenAI.

Design checklist

Start your design strategy based on the design review checklist for Security and identify vulnerabilities and controls to improve the security posture. Then, review the Azure security baseline for Azure OpenAI. Finally, extend the strategy to include more approaches as needed.

- ✓ **Protect confidentiality**: If you upload training data to Azure OpenAI, use customer-managed keys for data encryption, implement a key-rotation strategy, and delete training, validation, and training results data. If you use an external data store for

training data, follow security best practices for that store. For example, for Azure Blob Storage, use customer-managed keys for encryption and implement a key-rotation strategy. Use managed identity-based access, implement a network perimeter by using private endpoints, and enable access logs.

- ✓ **Protect confidentiality**: Guard against data exfiltration by limiting the outbound URLs that Azure OpenAI resources can access.
- ✓ **Protect integrity**: Implement access controls to authenticate and authorize user access to the system by using the least-privilege principle and by using individual identities instead of keys.
- ✓ **Protect integrity**: Implement jailbreak risk detection to safeguard your language model deployments against prompt injection attacks.
- ✓ **Protect availability**: Use security controls to prevent attacks that might exhaust model usage quotas. You might configure controls to isolate the service on a network. If the service must be accessible from the internet, consider using a gateway to block suspected abuse by using routing or throttling.

Recommendations

Expand table

Recommendation	Benefit
Secure keys: If your architecture requires Azure OpenAI key-based authentication, store those keys in Azure Key Vault, not in application code.	Separating secrets from code by storing them in Key Vault reduces the chance of leaking secrets. Separation also facilitates central management of secrets, easing responsibilities like key rotation.
Restrict access: Disable public access to Azure OpenAI unless your workload requires it. Create private endpoints if you're connecting from consumers in an Azure virtual network.	Controlling access to Azure OpenAI helps prevent attacks from unauthorized users. Using private endpoints ensures network traffic remains private between the application and the platform.
Microsoft Entra ID: Use Microsoft Entra ID for authentication and to authorize access to Azure OpenAI by using role-based access control (RBAC). Disable local authentication in Azure AI Services and set `disableLocalAuth` to `true`. Grant identities that perform completions or image generation the	Using Microsoft Entra ID centralizes the identity-management component and eliminates the use of API keys. Using RBAC with Microsoft Entra ID ensures that users or groups have exactly the permissions they need to do their job.

Recommendation	Benefit
Cognitive Services OpenAI User role. Grant model automation pipelines and ad-hoc data-science access a role like Cognitive Services OpenAI Contributor.	This kind of fine-grained access control isn't possible with Azure OpenAI API keys.
Use customer-managed keys: Use customer-managed keys for fine-tuned models and training data that's uploaded to Azure OpenAI.	Using customer-managed keys gives you greater flexibility to create, rotate, disable, and revoke access controls.
Protect against jailbreak attacks: Use Azure AI Content Safety Studio to detect jailbreak risks.	Detect jailbreak attempts to identify and block prompts that try to bypass the safety mechanisms of your Azure OpenAI deployments.

Cost Optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget while meeting business requirements.

Read the Cost Optimization design principles to learn about approaches for achieving those goals and the tradeoffs necessary in technical design choices related to Azure OpenAI.

Design checklist

Start your design strategy based on the design review checklist for Cost Optimization for investments. Fine-tune the design so that the workload is aligned with its allocated budget. Your design should use the appropriate Azure capabilities, monitor investments, and find opportunities to optimize over time.

- ✓ **Cost management**: Develop your cost model, considering prompt sizes. Understanding prompt input and response sizes and how text translates into tokens helps you create a viable cost model.
- ✓ **Usage optimization**: Start with pay-as-you-go pricing for Azure OpenAI until your token usage is predictable.
- ✓ **Rate optimization**: When your token usage is sufficiently high and predictable over a period of time, use the provisioned throughput pricing model for better cost optimization.
- ✓ **Usage optimization**: Consider model pricing and capabilities when you choose models. Start with less-costly models for less-complex tasks like text generation or

completion tasks. For more complex tasks like language translation or content understanding, consider using more advanced models. Consider different model capabilities and maximum token usage limits when you choose a model that's appropriate for use cases like text embedding, image generation, or transcription scenarios. By carefully selecting the model that best fits your needs, you can optimize costs while still achieving the desired application performance.

- ✓ **Usage optimization**: Use the token-limiting constraints offered by the API calls, such as `max_tokens` and `n`, which indicate the number of completions to generate.
- ✓ **Usage optimization**: Maximize Azure OpenAI price breakpoints, for example, fine-tuning and model breakpoints like image generation. Because fine-tuning is charged per hour, use as much time as you have available per hour to improve fine-tuning results while avoiding slipping into the next billing period. Similarly, the cost for generating 100 images is the same as the cost for 1 image. Maximize price breakpoints to your advantage.
- ✓ **Usage optimization**: Remove unused fine-tuned models when they're no longer being consumed to avoid incurring an ongoing hosting fee.
- ✓ **Adjust usage**: Optimize prompt input and response length. Longer prompts raise costs by consuming more tokens. However, prompts that are missing sufficient context don't help the models yield good results. Create concise prompts that provide enough context for the model to generate a useful response. Also ensure that you optimize the limit of the response length.
- ✓ **Cost efficiency**: Batch requests where possible to minimize the per-call overhead, which can reduce overall costs. Ensure that you optimize batch size.
- ✓ **Cost efficiency**: Because models have different fine-tuning costs, consider these costs if your solution requires fine-tuning.
- ✓ **Monitor and optimize**: Set up a cost-tracking system that monitors model usage. Use that information to help inform model choices and prompt sizes.

Recommendations

Expand table

Recommendation	Benefit
Design client code to set limits: Your custom clients should use the limit features of the Azure OpenAI completions API, such as maximum limit on the	Using API features to restrict usage aligns service consumption with client needs. This saves money by ensuring the model

Recommendation	Benefit
number of tokens per model (`max_tokens`) or number of completions to generation (`n`). Setting limits ensures that the server doesn't produce more than the client needs.	doesn't generate an overly long response that consumes more tokens than necessary.
Monitor pay-as-you-go usage: If you use the pay-as-you-go approach, monitor usage of TPM and RPM. Use that information to inform architectural design decisions such as what models to use, and to optimize prompt sizes.	Continuously monitoring TPM and RPM gives you relevant metrics to optimize the cost of Azure OpenAI models. You can couple this monitoring with model features and model pricing to optimize model usage. You can also use this monitoring to optimize prompt sizes.
Monitor provisioned throughput usage: If you use provisioned throughput, monitor provision-managed utilization to ensure you're not underutilizing the provisioned throughput you purchased.	Continuously monitoring provision-managed utilization gives you the information you need to understand if you're underutilizing your provisioned throughput.
Cost management: Use cost management features with OpenAI to monitor costs, set budgets to manage costs, and create alerts to notify stakeholders of risks or anomalies.	Cost monitoring, setting budgets, and setting alerts provides governance with the appropriate accountability processes.

Operational Excellence

Operational Excellence primarily focuses on procedures for **development practices**, **observability**, and **release management**.

The Operational Excellence design principles provide a high-level design strategy for achieving those goals toward the workload's operational requirements.

Design checklist

Start your design strategy based on the design review checklist for Operational Excellence. This checklist defines processes for observability, testing, and deployment related to Azure OpenAI.

- ✓ **Azure DevOps culture**: Ensure deployment of Azure OpenAI instances across your various environments, such as development, test, and production. Ensure that you have environments to support continuous learning and experimentation throughout the development cycle.
- ✓ **Observability**: Monitor, aggregate, and visualize appropriate metrics.

- ✓ **Observability**: If Azure OpenAI diagnostics are insufficient for your needs, consider using a gateway like Azure API Management in front of Azure OpenAI to log both incoming prompts and outgoing responses where permitted. This information can help you understand the effectiveness of the model for incoming prompts.
- ✓ **Deploy with confidence**: Use infrastructure as code (IaC) to deploy Azure OpenAI, model deployments, and other infrastructure required for fine-tuning models.
- ✓ **Deploy with confidence**: Follow large language model operations (LLMOps) practices to operationalize the management of your Azure OpenAI LLMs, including deployment, fine-tuning, and prompt engineering.
- ✓ **Automate for efficiency**: If you use key-based authentication, implement an automated key-rotation strategy.

Recommendations

Expand table

Recommendation	Benefit
Enable and configure Azure Diagnostics: Enable and configure Diagnostics for the Azure OpenAI Service.	Diagnostics collects and analyzes metrics and logs, helping you monitor the availability, performance, and operation of Azure OpenAI.

Performance Efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance.

The Performance Efficiency design principles provide a high-level design strategy for achieving those capacity goals against the expected usage.

Design checklist

Start your design strategy based on the design review checklist for Performance Efficiency for defining a baseline based on key performance indicators for Azure OpenAI workloads.

- ✓ **Capacity**: Estimate consumers' elasticity demands. Identify high-priority traffic that requires synchronous responses and low-priority traffic that can be asynchronous

and batched.

- ✓ **Capacity**: Benchmark token consumption requirements based on estimated demands from consumers. Consider using the Azure OpenAI benchmarking tool to help you validate the throughput if you're using provisioned throughput unit (PTU) deployments.

- ✓ **Capacity**: Use provisioned throughput for production workloads. Provisioned throughput offers dedicated memory and compute, reserved capacity, and consistent maximum latency for the specified model version. The pay-as-you-go offering can suffer from *noisy neighbor* problems like increased latency and throttling in regions under heavy use. Also, the pay-as-you-go approach doesn't offer guaranteed capacity.

- ✓ **Capacity**: Add the appropriate gateways in front of your Azure OpenAI deployments. Ensure that the gateway can route to multiple instances in the same or different regions.

- ✓ **Capacity**: Allocate PTUs to cover your predicted usage, and complement these PTUs with a TPM deployment to handle elasticity above that limit. This approach combines base throughput with elastic throughput for efficiency. Like other considerations, this approach requires a custom gateway implementation to route requests to the TPM deployment when the PTU limits are reached.

- ✓ **Capacity**: Send high-priority requests synchronously. Queue low-priority requests and send them through in batches when demand is low.

- ✓ **Capacity**: Select a model that aligns with your performance requirements, considering the tradeoff between speed and output complexity. Model performance can vary significantly based on the chosen model type. Models designed for speed offer faster response times, which can be beneficial for applications that require quick interactions. Conversely, more sophisticated models might deliver higher-quality outputs at the expense of increased response time.

- ✓ **Achieve performance**: For applications like chatbots or conversational interfaces, consider implementing streaming. Streaming can enhance the perceived performance of Azure OpenAI applications by delivering responses to users in an incremental manner, improving the user experience.

- ✓ **Achieve performance**: Determine when to use fine-tuning before you commit to fine-tuning. Although there are good use cases for fine-tuning, such as when the information needed to steer the model is too long or complex to fit into the

prompt, make sure that prompt engineering and retrieval-augmented generation (RAG) approaches don't work or are demonstrably more expensive.

- ✓ **Achieve performance**: Consider using dedicated model deployments per consumer group to provide per-model usage isolation that can help prevent noisy neighbors between your consumer groups.

Recommendations

There are no recommended configurations for Performance Efficiency for Azure OpenAI.

Azure Policy

Azure provides an extensive set of built-in policies related to Azure OpenAI and its dependencies. Some of the preceding recommendations can be audited through Azure Policy. Consider the following policy definitions:

- Disable key access
- Restrict network access
- Disable public network access
- Use Azure Private Link
- Enable data encryption with customer-managed keys

These Azure Policy definitions are also Azure Advisor security best-practice recommendations for Azure OpenAI.

Next steps

Consider the following articles as resources that demonstrate the recommendations highlighted in this article.

- Use this reference architecture as an example of how you can apply this article's guidance to a workload: Baseline OpenAI end-to-end chat reference architecture.
- Build implementation expertise by using Azure Machine Learning product documentation.

Feedback

Was this page helpful? Yes No

Architecture best practices for Azure Service Fabric

Article • 02/19/2025

Azure Service Fabric is a distributed systems platform that makes it easy to package, deploy, and manage scalable and reliable microservices and containers. These resources are deployed onto a network-connected set of virtual or physical machines, known as a *cluster*.

This article assumes that as an architect, you've reviewed the compute decision tree and chose Service Fabric as the compute platform for your workload. The guidance in this article provides architectural recommendations that are mapped to the principles of the Well-Architected Framework pillars.

> **ⓘ Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern along with design strategies localized to the technology scope.
>
> Also included are recommendations for the technology capabilities that can help materialize those strategies. The recommendations don't represent an exhaustive list of all configurations that are available for Service Fabric and its dependencies. Instead, they list the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept or to optimize your existing environments.
>
> Foundational architecture that demonstrates the key recommendations: Container orchestration for microservices.

Technology scope

This review focuses on the interrelated decisions for the following Azure resources:

- Service Fabric

> **ⓘ Note**
>
> This service guide builds on guidance in the Azure Virtual Machines and scale sets service guide. Virtual machine (VM) scale sets support Service Fabric nodes. Refer

to that service guide for recommendations about operating the compute back end for your Service Fabric nodes.

It's important to distinguish between *cluster* and *workload* in Service Fabric architectural and configuration guidance. The Service Fabric cluster admin and their resource provider share responsibility for cluster configuration. Developers handle workload configuration. This guide provides considerations and recommendations for both roles.

The following design checklists and recommendations indicate whether the guidance is applicable to cluster architecture, workload architecture, or both.

Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover fast from failures**.

Reliability design principles provide a high-level design strategy applied for individual components, system flows, and the system as a whole.

Design checklist

Start your design strategy based on the design review checklist for Reliability. Determine its relevance to your business requirements while keeping in mind the reliability of Service Fabric. Extend the strategy to include more approaches as needed.

- ✓ (Cluster) Determine the appropriate reliability level for your cluster based on the overall reliability target metric for the workload. The reliability level of the cluster that you identify dictates the minimum number of nodes to deploy for your primary node type. For more information, see Reliability levels.
- ✓ (Cluster) For critical workloads, consider using availability zones for your Service Fabric clusters.
- ✓ (Cluster) For production scenarios, use the Standard managed cluster SKU with a Silver durability tier, which requires at least five VMs. The Standard SKU provides greater reliability than the Basic SKU, which is suitable for nonproduction scenarios.
- ✓ (Cluster) Create extra secondary node types for your workloads to isolate different workload types, such as front-end services and back-end services. Then you can manage and scale those services independently. Each node type is backed by its own scale set.

Recommendations

Expand table

Recommendation	Benefit
(Cluster) Azure API Management can integrate with Service Fabric directly. Consider using API Management to expose and offload the cross-cutting functionality for cluster-hosted APIs.	API Management is a feature-rich application gateway that helps you securely publish, manage, and monitor APIs that are deployed to the Service Fabric cluster.
(Workload) For stateful workload scenarios, consider using the Reliable Services model.	The Reliable Services model keeps your services active when you experience system failures or network problems, or if the services themselves encounter malfunctions. For stateful services, your state is preserved when malfunctions occur.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload.

The Security design principles provide a high-level design strategy for achieving those goals by applying approaches to the technical design of Service Fabric.

Design checklist

Start your design strategy based on the design review checklist for Security and identify vulnerabilities and controls to improve the security posture. Extend the strategy to include more approaches as needed.

- ✓ (Cluster and workload) Familiarize yourself with Service Fabric product security guidance. For more information, see the following resources:
  - Security best practices
  - Cluster security scenarios
  - Application and service security
- ✓ (Cluster) Apply network segmentation and controls. Configure network security groups (NSGs) to restrict traffic flow between subnets and node types.
- ✓ (Cluster) Use native tools to securely manage application secrets and client certificates. To manage application secrets, use the Service Fabric Secret Store. To manage certificates, use Azure Key Vault.

- ✓ (Cluster) Consider using your existing load balancer so that you can use an internal load balancer and define different load balancers and NSGs for each node type.
- ✓ (Cluster) Enable Microsoft Entra integration to help securely control access to the cluster. Users can authenticate with their Entra credentials. Alternatively, you can use cluster client and admin certificates. Don't distribute the cluster client certificates among Service Fabric Explorer users.
- ✓ (Cluster and workload) Create a process to monitor the expiration date of client certificates.
- ✓ (Cluster and workload) Maintain separate clusters for development, staging, and production. Production environments typically require stricter security controls than nonproduction environments. And isolating environments from each other adds a layer of secuirty if one environment is compromised.

Recommendations

Expand table

Recommendation	Benefit
(Cluster) Ensure that you open the correct ports for application deployment and workloads.	This configuration helps ensure secure communication between the Service Fabric resources and the rest of the workload.
(Cluster) Use a separate data encipherment certificate to encrypt values when you use the Service Fabric Secret Store to distribute secrets.	A separate encipherment certificate isolates certificates, which reduces the risk of a single point of failure and provides more granular access control.
(Cluster) Apply an access control list (ACL) to your client certificate for your Service Fabric cluster.	An ACL provides an extra level of authentication so that you can better control access to certificates.
(Cluster) Use resource requests and limits to govern resource usage across the nodes in your cluster.	Resource limits help ensure that one service doesn't consume too many resources and deprive other services.
(Workload) Include client certificates in Service Fabric applications.	Client certificates for authentication provide enhanced security at both the cluster and workload level.
(Workload) Use a managed identity to authenticate Service Fabric applications to Azure resources.	Managed identities help you securely manage the credentials in your code for authenticating to various services, without saving them locally on a developer's workstation or in source control.

Recommendation	Benefit
(Cluster and workload) To host untrusted applications: - Use the strongest sandboxing technology available. - Remove access to Service Fabric runtime. - Follow other Service Fabric best practices.	These practices help ensure that only trusted and verified applications can interact with critical components. They also reduce the impact of untrusted applications that have vulnerabilities or malicious code on normal cluster operations.

Cost Optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget while meeting business requirements.

The Cost Optimization design principles provide a high-level design strategy for achieving those goals and making tradeoffs as necessary in the technical design related to Service Fabric and its environment.

Design checklist

Start your design strategy based on the design review checklist for Cost Optimization for investments. Fine-tune the design so that the workload is aligned with the budget that's allocated for the workload. Your design should use the right Azure capabilities, monitor investments, and find opportunities to optimize over time.

- ✓ (Workload and cluster) Use the Azure pricing calculator to estimate the intial cost. The compute instances, storage, networking resources, and IP addresses that you choose when you create a Service Fabric cluster incur charges. Service Fabric itself doesn't incur charges. For more information about cost modeling, see the example cost calculation process for application planning.
- ✓ (Cluster) Select appropriate VM SKUs. Choose VMs based on workload characteristics. For example, determine whether the workload is CPU intensive or runs interruptible processes.
- ✓ (Cluster) Select appropriate cluster SKUs. Use the Standard SKU for production environments and the Basic SKU for nonproduction environments, unless you have a specific reason to do otherwise. Use appropriate node types and sizes in each environment.

- ✔ (Cluster and workload) Select appropriate managed disk tiers and sizes. Review the Well-Architected Framework service guide for disk storage. To prevent unnecessary resource expenses, don't use VM SKUs that include temporary disk offerings.

Recommendations

Expand table

Recommendation	Benefit
(Cluster) Consider a VM SKU that includes temporary disk support if you don't need to maintain statefulness.	Optimize the resources that you pay for. A temporary disk instead of a managed disk can reduce costs for stateless workloads.
(Cluster and workload) Align your VM SKU selection with workload requirements. Ensure that you identify the correct node types, which are hosted on scale sets, to meet your requirements.	Matching your selection to your workload demands helps you avoid paying for unnecessary and expensive VM SKUs.
(Cluster and workload) Align your disk type selection with workload requirements.	An appropriate managed disk type helps you avoid paying for expensive and unnecessary disk types.

Operational Excellence

Operational Excellence primarily focuses on procedures for **development practices, observability, and release management**.

The Operational Excellence design principles provide a high-level design strategy for achieving those goals for the operational requirements of the workload.

Design checklist

Start your design strategy based on the design review checklist for Operational Excellence for defining processes for observability, testing, and deployment related to Service Fabric.

- ✔ (Cluster and workload) Integrate your Service Fabric components, including clusters, related infrastructure, and the application itself, into your monitoring and alerting platform. For more information, see Monitoring best practices.
- ✔ (Cluster and workload) Use the Service Fabric health model to continuously monitor the health of your solution. This tool should complement your overall workload

health model.

- ✓ (Cluster and workload) Create a process to monitor the expiration date of client certificates. For example, a feature of Key Vault sends an email when a specific percentage of the certificate's lifespan elapses.
- ✓ (Cluster and workload) Use continuous integration and continuous deployment (CI/CD) practices to manage your cluster deployments. Use a purposely built tool like Azure Pipelines or Github Actions to manage your CI/CD pipelines. These tools help you centrally manage all workload deployments in all environments with proper source control strategies.

Recommendations

Expand table

Recommendation	Benefit
(Workload) Use Application Insights to monitor your workloads.	Application Insights provides comprehensive application performance monitoring for live web applications, which enables you to collect and analyze application telemetry. This practice improves your application health and performance monitoring.
(Cluster and workload) Use Azure Monitor to monitor cluster and container infrastructure events.	Azure Monitor provides comprehensive monitoring and diagnostic capabilities that you can use to collect and analyze logs and metrics from your applications and Azure infrastructure. Azure Monitor integrates well with the Azure platform, including Service Fabric.
(Cluster) Implement appropriate cluster health policies as part of your health modeling.	These policies enable you to customize how failures are assessed in terms of cluster health. For example, you can set the maximum percentage of unhealthy nodes that are tolerated before the cluster is considered to be in error.
(Cluster) Implement appropriate application and service type health policies as part of your health modeling.	The application health policy describes how events are evaluated and how the health states for applications and their child components are aggregated. Service Fabric considers an entity unhealthy if it has a health report or if any of its child components are in a warning or error state.
(Cluster and workload) Use Azure Chaos Studio to inject malfunctions into your solution as part of your testing strategy.	Purposely introducing malfunctions into your solution helps you identify potential failure points and practice incident response measures.

Recommendation	Benefit
(Cluster and workload) Use Azure Pipelines as your CI/CD solution.	A CI/CD solution like Azure Pipelines helps you manage your deployments efficiently, consistently, and safely. Azure Pipelines has native support for Service Fabric deployments.

Performance Efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance.

The Performance Efficiency design principles provide a high-level design strategy for achieving those capacity goals against the expected usage.

Design checklist

Start your design strategy based on the design review checklist for Performance Efficiency for defining a baseline based on key performance indicators for Service Fabric.

- ✓ (Cluster) Take advantage of performance optimization and enhancing features that your workload requires. For recommendations about the underlying compute platform, see the Virtual Machines service guide.
- ✓ (Cluster) Deploy VMs and disk sizes that meet your perfomance requirements without incurring unnecessary expenses for unused capacity. Ensure that you can easily add capacity to meet your future growth plans.
- ✓ (Workload) Understand the programming models that Service Fabric supports. Choose the best model for your workload requirements. Each programming model has unique advantages and disadvantages, and your particular workload requirements might align with one model better than others.
- ✓ (Workload) Use established cloud architecture patterns to design your workload. Patterns that complement Service Fabric application designs include microservice, event-driven, and background-processing architecture patterns.

Recommendation

Expand table

Recommendation	Benefit
(Cluster) Exclude the Service Fabric processes that run on your Windows VMs from Microsoft Defender Antivirus if your security policies allow you to exclude processes and paths for open-source software.	This configuration reduces the performance impact and resource consumption overhead that Microsoft Defender Antivirus incurs.
(Cluster) Enable autoscaling for your cluster to add or reduce nodes on demand for a secondary node type.	Autoscaling reduces management overhead and potential business impact because you can monitor and optimize the number of nodes that service your workload.
(Cluster) Consider using accelerated networking.	Accelerated networking enables a high-performance path that bypasses the host from the data path, which reduces latency, jitter, and CPU usage for the most demanding network workloads.
(Cluster) Consider using encryption at host instead of Azure Disk Encryption.	Encryption at host supports all OS types and images, including custom images, by encrypting your VM data in Azure Storage. Azure Disk Encryption doesn't support these features.
(Workload) Implement the Service Fabric programming models that best suit your workload.	An appropriate programming model takes advantage of built-in features that support workload requirements, like state management, concurrency, and reuse of your existing codebase. To maintain your deployment standards, choose a programming model that aligns with those standards.
(Cluster and workload) Implement scaling to meet your business requirements. Find the right scaling mechanism for your workload.	Scaling enables maximum resource usage for your solution.

Azure policies

Azure provides an extensive set of built-in policies related to Service Fabric and its dependencies. Some of the preceding recommendations can be audited through Azure Policy. For example, you can check whether:

- Service Fabric clusters are configured to be zone redundant.
- Service Fabric clusters have the `ClusterProtectionLevel` property set to `EncryptAndSign`.

- Service Fabric clusters are configured to only use Microsoft Entra ID for client authentication.

For comprehensive governance, review the Azure Policy built-in definitions for Service Fabric and other policies that might affect the security of clusters and applications.

Azure Advisor recommendations

Azure Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments.

For more information, see Azure Advisor.

Related content

- Service Fabric managed cluster configuration options
- Azure application architecture fundamentals
- Deploy a Service Fabric managed cluster with an Azure Resource Manager template
- Deploy a Service Fabric managed cluster by using the Azure portal

---

Feedback

Was this page helpful? Yes No

Architecture best practices for Azure SQL Database

Article • 04/11/2025

Azure SQL Database is a fully managed platform as a service database engine that handles most database management functions without user involvement. Management functions include upgrades, patches, backups, and monitoring.

The single database resource type creates a database in SQL Database. The database has its own set of resources and is managed via a logical server. You can create multiple databases in a single resource pool by using elastic pools.

This article assumes that as an architect, you've reviewed the data store options and chose SQL Database as the database engine for your workload. The guidance in this article provides architectural recommendations that are mapped to the principles of the Well-Architected Framework pillars.

This article also assumes that you're familiar with the SQL Database core concepts. For more information, see Core concepts of SQL Database and What's new in SQL Database?.

> **ⓘ Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern along with design strategies localized to the technology scope.
>
> Also included are recommendations for the technology capabilities that can help materialize those strategies. The recommendations don't represent an exhaustive list of all configurations that are available for Azure SQL Database and its dependencies. Instead, they list the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept or to optimize your existing environments.
>
> Foundational architecture that demonstrates the key recommendations: Baseline highly available zone-redundant web application.

Technology scope

This review focuses on the interrelated decisions for the following Azure resources:

- SQL Database

Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover fast from failures**.

Reliability design principles provide a high-level design strategy applied for individual components, system flows, and the system as a whole.

Design checklist

Start your design strategy based on the design review checklist for Reliability. Determine its relevance to your business requirements while keeping in mind the reliability of SQL Database. Extend the strategy to include more approaches as needed.

- ✓ **Familiarize yourself with SQL Database product reliability guidance:**
  For more information, see the following resources:
  - Business continuity overview
  - High availability
  - High availability and disaster recovery checklist
  - Automated backups

- ✓ **Choose appropriate SKU configurations:** Use the Business Critical tier for critical workloads because it offers the highest reliability guarantees.

  Consider the SQL Database Hyperscale tier to meet strict recovery time objective and recovery point objective targets when the Business Critical tier isn't practical. The Hyperscale tier uses storage snapshots rather than traditional database backup mechanisms, which provide zero downtime and rapid recovery.

- ✓ **Build redundancy to improve resiliency:** Enhance the availability of your databases by using active geo-replication, failover groups, and zone-redundancy.

- ✓ **Use native disaster recovery and backup features:** Use geo-restore to recover from a service outage. You can restore a database on any SQL Database server or a managed instance in any Azure region. Restoration uses the most recent geo-replicated backups.

  Use point-in-time restore to recover from human error. Point-in-time restore returns your database to an earlier point in time to recover data from inadvertent changes.

- ✓ **Monitor reliability and overall health indicators of SQL Database:** Monitor SQL Database in near real-time to detect reliability incidents.

- ✓ **Implement retry logic and backoff logic:** Use these features to handle transient faults in your application.

- ✓ **Back up your TDE encryption keys:** When you use customer-managed keys for Transparent Data Encryption (TDE), back up the keys to Azure Key Vault.

Recommendations

Expand table

Recommendation	Benefit
Use active geo-replication to create a readable secondary database in a different region. If your application supports read-only connection strings, use the secondary database to handle read-only database actions. This method reduces burden on the primary instance.	If your primary database fails, you can perform a manual failover to the replicated database to ensure that your application continues to operate with minimal downtime. A secondary database that has a read-only connection improves overall performance and ensures higher availability for read operations.
Use failover groups to automate failover from primary to secondary database instances. Failover groups provide read-write and read-only listener endpoints that remain unchanged during geo-failovers. A geo-failover switches all secondary databases in the group to the primary role. After geo-failover finishes, the Domain Name System (DNS) record is automatically updated to redirect the endpoints to the new region.	Geo-failover groups simplify the deployment and management of geo-replicated databases, which ensures continuous availability without manual intervention. You don't have to change the connection string for your application after a geo-failover because connections are automatically routed to the new primary database.
Configure your database for zone redundancy. Zone-redundant availability spreads your data across three Azure availability zones in the primary region. Each availability zone is a separate physical location that has independent power, cooling, and networking.	If one of the availability zones experiences an outage, the SQL Database instance remains operational and automatically fails over to an operational zone without losing committed data.
Implement retry logic in your applications. SQL Database is resilient to transitive infrastructure failures, but these failures might affect your connectivity. Implement backoff logic into your retry logic.	Retry logic improves an application's resilience to temporary failures and helps the application recover without manual intervention. When the application encounters a transient error while interacting with SQL Database, make sure that your code can retry the call. Backoff logic adds delays between retries, which helps prevent network congestion and server overload.

Recommendation	Benefit
Back up your encryption keys when you use your own keys. When you use your own encryption keys, you get full and granular control of the keys, but you have to manage the keys and key rotation. Key Vault provides a centralized location to manage encryption keys.	Key Vault centrally manages your keys in one tool and helps prevent accidental key loss.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload.

The Security design principles provide a high-level design strategy for achieving those goals by applying approaches to the technical design of SQL Database.

Design checklist

Start your design strategy based on the design review checklist for Security and identify vulnerabilities and controls to improve the security posture. Extend the strategy to include more approaches as needed.

- ✓ **Review security baselines:** To enhance the security posture of your workload, review the Azure security baseline for SQL Database.

  For more information about features that can help workloads meet security and compliance requirements, see Built-in security and compliance features.

- ✓ **Implement strict, conditional, and auditable identity and access management:** Use Microsoft Entra ID for your workload's authentication and authorization needs. Microsoft Entra ID provides centralized authorization and access management.

  Centralize the administration of sign-ins, firewall rules, auditing rules, and threat detection policies for collections of databases at the logical server level.

- ✓ **Encrypt data:** Enable data encryption to safeguard confidentiality and integrity. Use features like `Always Encrypted` and `Always Encrypted with secure enclaves` to protect highly sensitive information, like credit card numbers and social security numbers. These features help prevent encryption keys from being exposed to the database engine.

- ✓ **Apply network segmentation and security controls:** Create intentional segmentation and perimeters in your network design and apply defense-in-depth principles by using localized network controls at all network boundaries.

Use virtual network rules to control communication from specific subnets in virtual networks.

Configure SQL Database firewall rules at the database level and server level to restrict access to databases. If you use an Azure firewall, configure Azure Firewall application rules with SQL fully qualified domain names (FQDNs).

Review the SQL Database connectivity architecture. Use the `Redirect` or `Proxy` connection policy when practical.

Recommendations

Expand table

Recommendation	Benefit
Configure the minimum Transport Layer Security (TLS) version to at least version 1.2, and if practical version 1.3.	Azure is deprecating TLS versions 1.0 and 1.1 because they're insecure. Version 1.3 is the latest and most secure version.
Consider designing database tables based on the ledger.	The ledger functionality provides auditing, tamper-evidence, and trust for all data changes.
Configure Always Encrypted to protect sensitive data inside applications by delegating data access to encryption keys.	Always Encrypted helps prevent encryption keys from being exposed to the database engine. This feature provides a separation between people who own the data and should have access to it and people who manage the data but shouldn't have access. For example, on-premises database administrators, cloud database operators, or other high-privileged unauthorized users shouldn't have access.
Extend the capabilities of Always Encrypted by enabling secure enclaves.	Use secure enclaves to perform otherwise unsupported operations in Always Encrypted databases, inside a secure memory enclave on the server side.
Use Azure Private Link for SQL Database to enforce secure communication over private endpoints.	Private Link provides private connectivity between your database and virtual network so that you can disable public access.
Scan for vulnerabilities with the Microsoft Defender for SQL Database vulnerability assessment.	The SQL vulnerability assessment is a built-in service for SQL Database that identifies and helps remediate potential security vulnerabilities. It provides actionable steps and customized remediation scripts based on Microsoft best practices.
Detect anomalous activities by using advanced threat protection for SQL Database. These activities can indicate unusual and potentially harmful	Advanced threat protection provides security alerts for anomalous activities, which helps you detect and respond to potential threats when they occur. Alerts are integrated into Microsoft Defender for Cloud.

Recommendation	Benefit
attempts to access or exploit databases.	
Track database events by using SQL Database auditing.	Auditing helps you maintain regulatory compliance, understand database activity, and gain insight into discrepancies and anomalies that could indicate business concerns or suspected security violations.
Configure a user-assigned managed identity as the server identity.	Managed identities for Azure resources eliminate the need to manage credentials in code.
Disable SQL-based authentication and allow Microsoft Entra authentication only.	Microsoft Entra for authentication centralizes your identity, access, and authorization management and provides granular permissions to SQL Database resources.

Cost Optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget while meeting business requirements.

The Cost Optimization design principles provide a high-level design strategy for achieving those goals and making tradeoffs as necessary in the technical design related to SQL Database.

Design checklist

- ✓ **Familiarize yourself with SQL Database cost management resources:** Review the Plan and manage costs for SQL Database article. This resource contains cost-saving strategies, including recommendations about how to optimize cost-effective instances and resources and how to choose the right billing model for your workload.
- ✓ **Estimate the initial cost:** As part of your cost modeling exercise, use the Azure pricing calculator to evaluate the approximate costs associated with SQL Database in your workload.
- ✓ **Choose the right SQL Database service tier for your workload:** Evaluate the SQL Database Serverless and Hyperscale tiers to better align pricing with your use case.

  Consider elastic pools to manage and scale multiple databases.
- ✓ **Optimize component costs:** Consider reserved capacity for SQL Database for static workloads that you intend to operate for long periods of time.

Fine-tune backup storage consumption to avoid charges for excess usage.

- ✓ **Optimize application code costs:** Optimize queries and other operations to reduce resource consumption, minimize runtime, and improve overall performance.
- ✓ **Optimize scaling costs:** Incorporate cost optimization considerations into your database scaling strategies.

  To reduce costs, scale down databases during periods of low usage. Examples include seasonal workloads that have weeks or months of decreased load or workloads that are idle overnight.

Recommendations

Expand table

Recommendation	Benefit
Research the available SQL Database service tiers, and select the best model for each use case based on your capacity planning.	An appropriate service tier can help you avoid wasted costs from overprovisioning.
Optimize your queries, tables, and databases by using query performance insight and performance recommendations.	Optimized operations can help reduce resource consumption and help you determine the appropriate SQL Database SKUs to provision to meet your performance and budgetary requirements.
Use the recommended backup strategies to fine-tune your backup storage consumption. For vCore databases in SQL Database, the storage that each type of backup consumes is reported on the database monitoring pane as a separate metric. The backup types include full, differential, and log backups. Backup storage consumption up to the maximum data size for a database is included in the price of the database. Excess storage costs depend on the workload and maximum size of the individual databases.	Understand the factors that influence backup storage costs, such as database size, rate of change, and retention period. Properly configure back storage to optimize your backup strategy and minimize expenses.
Consider using the Serverless tier for single databases that have intermittent, unpredictable usage patterns and can afford a delay in compute warm-up after idle usage periods.	Serverless databases automatically pause and scale their compute resources based on workload demand. Billing is based on the amount of compute used per second. If your database has unpredictable or bursty usage patterns with periods of low or idle usage,

Recommendation	Benefit
	serverless databases can help optimize cost and performance.
Consider using the Hyperscale tier for all workload types.	The Hyperscale tier's cloud-native architecture provides independently scalable compute and storage to support the widest variety of traditional and modern applications. Compute and storage resources in the Hyperscale tier substantially exceed the resources available in the General Purpose and Business Critical tiers. Hyperscale databases don't require extra SQL licensing costs.
Save costs for resources by using reserved capacity. After you determine the total compute capacity and performance tier for Azure SQL databases in a region, use that information to reserve the capacity for one or three years.	Pre-purchase compute resources to reduce compute costs for SQL Database. One-year reservations can save you up to 40%, and three-year reservations can save you up to 60%.
Save on SQL licensing costs by exchanging your Software Assurance-enabled SQL Server licenses with Azure Hybrid Benefit.	Azure Hybrid Benefit can save you up to 30% on licensing costs in SQL Database. And you can take advantage of your existing Software Assurance agreement as you migrate databases to Azure.
Use elastic pools to help manage and scale multiple databases.	SQL Database elastic pools are a simple, cost-effective solution to manage and scale multiple databases that have varying and unpredictable usage demands. The databases in an elastic pool are on a single server and share a set number of resources at a set price.

Operational Excellence

The Operational Excellence design principles provide a high-level design strategy for achieving those goals for the operational requirements of the workload.

Design checklist

Start your design strategy based on the design review checklist for Operational Excellence for defining processes for observability, testing, and deployment related to SQL Database.

- ✓ **Familiarize yourself with SQL Database operational excellence resources:** Review the Monitoring and performance tuning in SQL Database article. This article contains detailed

monitoring guidance, including recommendations for monitoring query performance, configuring alerts, and using automatic tuning to enhance efficiency.

- ✓ **Use infrastructure as code (IaC) technologies:** Use IaC technologies like Bicep and Azure Resource Manager templates to deploy Azure SQL databases to achieve consistent results and take advantage of reusability.

  Use the latest version of the Resource Manager API to take advantage of the newest features, security updates, and performance improvements.

- ✓ **Monitor your SQL Database:** Use monitoring to detect reliability incidents and optimize performance. Start by monitoring the CPU and input/output resources that your workload uses. For help designing the reliability and health monitoring solution for your workload, see Health modeling for workloads.

- ✓ **Optimize management of business continuity and disaster recovery:** Use Azure Backup to protect SQL Database and regularly test your backup strategy.

- ✓ **Use native database administration features:** Adopt SQL Database to relieve database administrators of many of their traditional tasks, like infrastructure-related administration, backup management, and high availability and disaster recovery operations. Encourage their growth in cloud-native administration and integration with DevOps teams as they adopt a data-as-code mindset.

Recommendations

Expand table

Recommendation	Benefit
Monitor SQL Database by using tools, such as database watcher, to detect reliability incidents.	Quick detection of reliability incidents enables you to promptly identify and address performance problems, which minimizes disruptions to your workload.
Use continuous integration and continuous deployment workflows to streamline the deployment process of SQL projects.	A repeatable workflow to release changes minimizes the need for human intervention and reduces the possibility of errors. This approach ensures a more reliable and efficient deployment process.
Use Azure Backup to back up a SQL Database server.	Azure Backup enables centralized management of business continuity and disaster recovery so that you can efficiently oversee your backup operations. Azure Backup supports long-term retention of recovery points to ensure that your data is preserved for extended periods. It also performs cross-region and cross-subscription restores, which enhance the flexibility and resilience of your backup strategy.

Performance Efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance.

The Performance Efficiency design principles provide a high-level design strategy for achieving those capacity goals against the expected usage.

Design checklist

Start your design strategy based on the design review checklist for Performance Efficiency for defining a baseline based on key performance indicators for SQL Database.

- ✓ **Familiarize yourself with SQL Database performance efficiency resources:** Review the Performance Center for SQL Server Database Engine and SQL Database and Tune applications and databases for performance articles. These articles provide insights about how to improve the performance of your SQL Database, including recommendations to optimize server configuration and query performance.

- ✓ **Select the right tier, features, and billing model:** Microsoft recommends the latest vCore-based purchasing model.

  Review resource limits. Resource limits for singe databases in each pricing tier are also known as *service objectives*. For more information, see vCore-based single database resource limits. For elastic pool resource limits, see vCore-based elastic pool resource limits.

  Review the default max degree of parallelism (MAXDOP), and configure it as needed based on a migrated or expected workload.

- ✓ **Optimize workload design and application code for performance:** Consider offloading read-only actions to read-only replicas.

  Applications that connect to SQL Database should use the latest connection providers, for example the latest OLE DB driver or ODBC driver.

  When you use elastic pools, familiarize yourself with resource governance.

Recommendations

Expand table

Recommendation	Benefit
Apply performance recommendations to optimize your SQL Database or address workload problems.	Use these recommendations to improve database performance.
Configure automatic tuning to automatically optimize your database performance. Automatic tuning is a feature that monitors queries, adapts to changing workloads, and applies tuning recommendations automatically.	Continuous performance tuning based on AI and machine learning can optimize performance and stabilize workloads.
Use the query store to help optimize the performance of your queries. The query store helps you quickly find performance differences that query plan changes cause, which simplifies performance troubleshooting.	The query store continuously collects detailed information about all queries. This capability simplifies troubleshooting query performance because you can use this information to quickly diagnose and address query performance problems.
Use the query performance insight tool to identify the top resource-consuming and long-running queries in your workload. Query performance insight provides details about database queries by CPU, duration, and implementation count. And you can view the history of resource usage.	Query performance insight provides deep insight into database query metrics such as CPU, duration, and implementation count. This feature reduces the time that you spend troubleshooting database performance.
Use built-in tools to diagnose and troubleshoot high CPU usage. These tools can help identify the causes of high CPU usage and optimize Transact-SQL queries.	Properly tuned applications and databases lead to more efficient operations and potentially lower costs.
Understand how blocking and deadlocking problems affect your database performance.	Resolving blocking and deadlocking problems helps your database operate more efficiently, eliminates bottlenecks, and improves performance.
Review the Azure portal usage reporting, and scale single databases or elastic pools as appropriate.	Regularly review usage reports to ensure that you use your resources efficiently. This practice helps identify underused resources, which can reduce costs.
Use in-memory database objects to enhance the performance of your workloads in scenarios like high-performance, online transaction-processing applications and real-time analytics.	In-memory technologies enable you to improve the performance of your workload and potentially reduce database costs.

Azure policies

Azure provides an extensive set of built-in policies related to SQL Database. A set of Azure policies can audit some of the preceding recommendations. For example, you can check

whether:

- Microsoft Entra-only authentication is enabled by default during creation.
- Zone redundancy is enabled to enhance availability and resilience.
- Long-term, geo-redundant backup should be enabled for Azure SQL databases.
- Setting the minimal TLS version to 1.2 improves security by ensuring that SQL Database can only be accessed from clients that use TLS 1.2. Don't use earlier versions of TLS because they have well-documented security vulnerabilities.
- SQL servers should have auditing enabled to ensure that you capture the operations performed against your SQL assets.

For comprehensive governance, review the built-in definitions for SQL Database policies that are listed in Azure Policy built-in definitions.

Related resources

- Tutorial: Secure a database in SQL Database
- SQL Database features
- Resolve Transact-SQL differences during migration to SQL Database
- Azure Database Migration Guides
- Video: Data exposed

Azure SQL Managed Instance and reliability

Article • 11/14/2023

Azure SQL Managed Instance is the intelligent, scalable cloud database service that combines the broadest SQL Server database engine compatibility with all the benefits of a fully managed and evergreen platform as a service.

The goal of the high availability architecture in SQL Managed Instance is to guarantee that your database is up and running without worrying about the impact of maintenance operations and outages. This solution is designed to:

- Ensure that committed data is never lost because of failures.
- Ensure that maintenance failures don't affect your workload.
- Ensure that the database won't be a single point of failure in your software architecture.

For more information about how Azure SQL Managed Instance supports application and workload resilience, reference the following articles:

- High availability for Azure SQL Managed Instance
- Use autofailover groups to enable transparent and coordinated geo-failover of multiple databases

The following sections include design considerations, a configuration checklist, and recommended configuration options specific to Azure SQL Managed Instance, and reliability.

Design considerations

Azure SQL Managed Instance includes the following design considerations:

- Define an application performance SLA and monitor it with alerts. Detecting quickly when your application performance inadvertently degrades below an acceptable level is important to maintain high resiliency. Use a monitoring solution to set alerts on key query performance metrics so you can take action when the performance breaks the SLA.
- Use point-in-time restore to recover from human error. Point-in-time restore returns your database to an earlier point in time to recover data from changes done inadvertently. For more information, read the Point-in-time-restore (PITR) documentation for managed instance.

- Use geo-restore to recover from a service outage. Geo-restore restores a database from a geo-redundant backup into a managed instance in a different region. For more information, reference Recover a database using Geo-restore documentation.
- Consider the time required for certain operations. Make sure you separate time to thoroughly test the amount of time required to scale up and down your existing managed instance, and to create a new managed instance. This timing practice ensures that you understand completely how time consuming operations will affect your RTO and RPO.

Checklist

Have you configured Azure SQL Managed Instance with reliability in mind?

- ✓ Use the Business Critical Tier.
- ✓ Configure a secondary instance and an Autofailover group to enable failover to another region.
- ✓ Implement Retry Logic.
- ✓ Monitor your SQL MI instance in near-real time to detect reliability incidents.

Configuration recommendations

Explore the following table of recommendations to optimize your Azure SQL Managed Instance configuration for reliability:

Recommendation	Description
Use the Business Critical Tier.	This tier provides higher resiliency to failures and faster failover times because of the underlying HA architecture, among other benefits. For more information, reference SQL Managed Instance High availability.
Configure a secondary instance and an Autofailover group to enable failover to another region.	If an outage impacts one or more of the databases in the managed instance, you can manually or automatically failover all the databases inside the instance to a secondary region. For more information, read the Autofailover groups documentation for managed instance.
Implement Retry Logic.	Although Azure SQL MI is resilient to transitive infrastructure failures, these failures might affect your connectivity. When a transient error occurs while working with SQL MI, make sure your code can retry the call. For more information, reference how to implement retry logic.
Monitor your SQL MI instance in near-real time to detect	Use one of the available solutions to monitor your SQL MI to detect potential reliability incidents early and make your

Recommendation	Description
reliability incidents.	databases more reliable. Choose a near real-time monitoring solution to quickly react to incidents. For more information, check out the Azure SQL Managed Instance monitoring options .

Next step

Azure SQL Managed Instance and operational excellence

Azure SQL Managed Instance and operational excellence

Article • 04/02/2025

Azure SQL Managed Instance is the intelligent, scalable cloud database service that combines the broadest SQL Server database engine compatibility with all the benefits of a fully managed and evergreen platform as a service.

The goal of the high availability architecture in SQL Managed Instance is to guarantee that your database is up and running without worrying about the impact of maintenance operations and outages. This solution is designed to:

- Ensure that committed data is never lost because of failures.
- Ensure that maintenance failures don't affect your workload.
- Ensure that the database won't be a single point of failure in your software architecture.

For more information about how Azure SQL Managed Instance supports operational excellence for your application workloads, reference the following articles:

- Overview of Azure SQL Managed Instance management operations
- Monitoring Azure SQL Managed Instance management operations

The following sections include design considerations, a configuration checklist, and recommended configuration options specific to Azure SQL Managed Instance, and operational excellence.

Design considerations

Azure SQL Managed Instance includes the following design considerations:

- Define an application performance SLA and monitor it with alerts. Detecting quickly when your application performance inadvertently degrades below an acceptable level is important to maintain high resiliency. Use a monitoring solution to set alerts on key query performance metrics so you can take action when the performance breaks the SLA.
- Use point-in-time restore to recover from human error. Point-in-time restore returns your database to an earlier point in time to recover data from changes done inadvertently. For more information, read the Point-in-time-restore (PITR) documentation for managed instance.

- Use geo-restore to recover from a service outage. Geo-restore restores a database from a geo-redundant backup into a managed instance in a different region. For more information, reference Recover a database using Geo-restore documentation.
- Consider the time required for certain operations. Make sure you separate time to thoroughly test the amount of time required to scale up and down your existing managed instance, and to create a new managed instance. This timing practice ensures that you understand completely how time consuming operations will affect your RTO and RPO.
- Use Azure Backup to protect your SQL database on an Azure Virtual Machine, ensuring recovery during outages. It stores backups in a Recovery Services vault based on your configured policy. Set the vault to Geo-redundant for restoration in a secondary region or subscription. Learn how to restore the database using Azure Backup.

Checklist

Have you configured Azure SQL Managed Instance with operational excellence in mind?

- ✓ Use the Business Critical Tier.
- ✓ Configure a secondary instance and an Autofailover group to enable failover to another region.
- ✓ Implement Retry Logic.
- ✓ Monitor your SQL MI instance in near-real time to detect reliability incidents.
- ✓ Restore the SQL database using the Azure Backup service.

Configuration recommendations

Explore the following table of recommendations to optimize your Azure SQL Managed Instance configuration for operational excellence:

Expand table

Recommendation	Description
Use the Business Critical Tier.	This tier provides higher resiliency to failures and faster failover times because of the underlying HA architecture, among other benefits. For more information, reference SQL Managed Instance High availability.
Configure a secondary instance and an Autofailover group to	If an outage impacts one or more of the databases in the managed instance, you can manually or automatically failover

Recommendation	Description
enable failover to another region.	all the databases inside the instance to a secondary region. For more information, read the Autofailover groups documentation for managed instance.
Implement Retry Logic.	Although Azure SQL MI is resilient to transitive infrastructure failures, these failures might affect your connectivity. When a transient error occurs while working with SQL MI, make sure your code can retry the call. For more information, reference how to implement retry logic.
Monitor your SQL MI instance in near-real time to detect reliability incidents.	Use one of the available solutions to monitor your SQL MI to detect potential reliability incidents early and make your databases more reliable. Choose a near real-time monitoring solution to quickly react to incidents. For more information, check out the Azure SQL Managed Instance monitoring options.
Restore the SQL database using the Azure Backup service.	Use Azure Backup to detect SQL databases, configure backups in the Recovery Services vault, and restore the database fully or to a specific point in time. It also supports disaster recovery drills, monitoring, and other management tasks.

Next step

Azure Cosmos DB for NoSQL

Feedback

Was this page helpful? Yes No

Architecture best practices for Azure Local

Article • 11/27/2024

Azure Local extends Azure to customer-owned infrastructure, enabling local execution of modern and traditional applications across distributed locations. This solution offers a unified management experience on a single control plane and supports a wide range of validated hardware from trusted Microsoft partners. You can use Azure Local and Azure Arc capabilities to keep business systems and application data on-premises to address data sovereignty, regulation and compliance, and latency requirements.

This article assumes you have an understanding of hybrid systems and have working knowledge of Azure Local. The guidance in this article provides architectural recommendations that are mapped to the principles of the Azure Well-Architected Framework pillars.

> **Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern along with design strategies localized to the technology scope.
>
> Also included are *recommendations* on the technology capabilities that can help materialize those strategies. The recommendations don't represent an exhaustive list of all configurations available for Azure Local and its dependencies. Instead, they list the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept or optimize your existing environments.
>
> Foundational architecture that demonstrates the key recommendations: Azure Local baseline reference architecture.

Technology scope

This review focuses on the interrelated decisions for the following Azure resources:

- Azure Local (*platform*), version 23H2 and later
- Azure Arc VMs (*workload*)

> **Note**
>
> This article covers the preceding scope and provides checklists and recommendations that are organized by **platform architecture** and **workload architecture**. Platform concerns are the responsibility of the platform administrators. Workload concerns are the responsibility of the workload operator and application developers. These roles and responsibilities are distinct and can be owned by separate teams or individuals. Keep that distinction in mind when you apply the guidance.

This guidance doesn't focus on specific resource types that you can deploy on Azure Local, such as Azure Arc VMs, Azure Kubernetes Service (AKS), and Azure Virtual Desktop. When you deploy these resource types on Azure Local, refer to the respective workload guidance to design solutions that meet your business requirements.

Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover fast from failures**.

The **Reliability design principles** provide a high-level design strategy applied for individual components, system flows, and the system as a whole.

In hybrid cloud deployments, the goal is to reduce the effects of one component failure. Use these design checklists and configuration suggestions to lessen the impact of a component failure for workloads that you deploy on Azure Local.

It's important to distinguish between *platform reliability* and *workload reliability*. Workload reliability has a dependency on the platform. Application owners or developers must design applications that can deliver the defined reliability targets.

Design checklist

Start your design strategy based on the design review checklist for Reliability. Determine its relevance to your business requirements while keeping in mind the performance of Azure Local. Extend the strategy to include more approaches as needed.

- ✓ (Azure Local platform architecture and workload architecture) **Define workload reliability targets**.

- Set your service-level objectives (SLOs) so that you can evaluate availability targets. Calculate SLOs as a percentage, such as 99.9%, 99.95%, or 99.995%, that reflects workload uptime. Keep in mind that this calculation isn't based only on the platform metrics that the Azure Local instance or workload emits. To get a comprehensive target measurement, factor in nuanced factors that are quantified, such as expected downtime during releases, routine operations, supportability, or other workload-specific or organizational-specific factors.

- Microsoft-provided service-level agreements (SLAs) often influence SLO calculations. But Microsoft doesn't provide an SLA for the uptime and connectivity of Azure Local instances or the deployed workload, because Microsoft doesn't control the customer datacenter reliability (*such as power and cooling*) or the people and processes that administer the platform.

✓ (Azure Local platform architecture) **Consider how performance and operations affect reliability**.

Degraded performance of the instance or its dependencies can make the Azure Local platform unavailable. For example:

- Without proper workload capacity planning, it's challenging to *rightsize Azure Local instances* in the design phase, which is a requirement so that the workload can meet the desired reliability targets. Use the Azure Local sizer tool during instance design. Consider the "*N+1 minimum requirement for number of machines*" if you require highly available VMs. For business-critical or mission-critical workloads, consider using a "*N+2 number of machines*" for the instance size if resiliency is paramount.

- The reliability of the platform depends on how well the critical platform dependencies, such as physical disk types, perform. *You must choose the right disk types for your requirements*. For workloads that need low-latency and high-throughput storage, we recommend an all-flash (*NVMe/SSD only*) storage configuration. For general purpose compute, a hybrid storage (*NVMe or SSDs for cache and HDDs for capacity*) configuration might provide more storage space. But the tradeoff is that spinning disks have significantly lower performance if your workload exceeds the cache working set, and HDDs have a much lower *mean time between failure* value compared to NVMe/SSDs.

  Performance Efficiency describes these examples in more detail.

Improper Azure Local operations can affect patching and upgrades, testing, and consistency of deployments. Here are some examples:

- If the Azure Local platform doesn't *evolve with the latest hardware original equipment manufacturer (OEM) firmware, drivers, and innovations*, the platform might not take advantage of the latest resiliency features. Apply hardware OEM driver and firmware updates regularly. For more information, see Azure Local solution catalog.

- You must test the target environment for connectivity, hardware, and identity and access management before your deployment. Otherwise, you might deploy the Azure Local solution to an unstable environment, which can create reliability problems. You can use the environmental checker tool in standalone mode to detect problems, even before the instance hardware is available.

  For operational guidance, see Operational Excellence.

✓ (Azure Local platform architecture) **Provide fault tolerance to the instance and its infrastructure dependencies**.

- **Storage design choices**. For most deployments, the default option to "*automatically create workload and infrastructure volumes*" is sufficient. If you select the *advanced option: "create required infrastructure volumes only*", configure the appropriate *volume fault tolerance within Storage Spaces Direct* based on your workload requirements. These decisions influence the performance, capacity, and resiliency capabilities of the volumes. For example, a three-way mirror increases reliability and performance for instances with three or more machines. For more information, see Fault tolerance for storage efficiency and Create Storage Spaces Direct virtual disks and volumes.

- **Network architecture**. Use a *validated network topology* to deploy Azure Local. Multi-machine instances, *with four or more physical machines*, require the "*storage switched*" design. Instances with two or three machines can optionally use the "*storage switchless*" design. Regardless of the instance size, we recommend that you use dual top of rack (ToR) switches for the management and compute intents (*north and south uplinks*) to provide increased fault tolerance. The dual ToR topology also provides resiliency during switch servicing (*firmware update*) operations. For more information, see Validated network topologies.

✓ (Workload architecture) **Build redundancy to provide resiliency**.

- Consider a workload that you deploy on a single Azure Local instance as a *locally redundant deployment*. The instance provides high availability at the platform level, but you must remember that you deploy the instance "*in a single rack*". Therefore, for business-critical or mission-critical use cases, we recommend that

you deploy multiple instances of a workload or service across two or more separate Azure Local instances, ideally in separate physical locations.

- Use industry-standard high-availability patterns for workloads, for example a design that provides active/passive synchronous or asynchronous data replication (*such as SQL Server Always On*). Another example is an external network load balancing (*NLB*) technology that can route user requests across the multiple workload instances that run on Azure Local instances that you deploy in separate physical locations. Consider using a partner external NLB device. Or evaluate the load balancing options that support traffic routing for hybrid and on-premises services, such as an Azure Application Gateway instance that uses Azure ExpressRoute or a VPN tunnel to connect to an on-premises service.

  For more information, see Deploy workloads instances across multiple Azure Local instances.

✓ (Workload architecture) **Plan and test recoverability** based on your workload recovery point objective (RPO) and recovery time objective (RTO) targets.

Have *a well-documented disaster recovery plan*. Test the recovery steps regularly to ensure that your business continuity plans and processes are valid. Determine whether Azure Site Recovery is a viable choice for protecting VMs that run on Azure Local. For more information, see Protect VM workloads with Azure Site Recovery on Azure Local (preview).

✓ (Workload architecture) **Configure and regularly test workload backup and restore procedures**.

Business requirements for *data recovery and retention* drive the strategy for workload backups. A comprehensive strategy includes considerations for *workload operating system (OS) and application persistent data*, with the ability to restore individual (*point-in-time*) file-level and folder-level data. Configure the backup retention policies based on your data recovery and compliance requirements, which determine the number and age of available data recovery points. Explore Azure Backup as an option to enable host-level or VM guest-level backups for Azure Local. Review data protection solutions from Backup independent software vendor partners where relevant. For more information, see Azure Backup guidance and best practices and Azure Backup for Azure Local.

Recommendations

Expand table

Recommendation	Benefit
Reserve the equivalent of **one capacity disk worth of space per machine** within the Storage Spaces Direct storage pool.	If you choose to create workload volumes after you deploy an Azure Local instance (*Advanced option: "create required infrastructure volumes only"*), we recommend that you **leave 5% to 10% of the total pool capacity unallocated in the storage pool**. This reserved and unused, or free, capacity enables Storage Spaces Direct to repair "in-place" when a physical disk fails, which improves data resiliency and performance if a physical disk failure occurs.
Ensure that all physical machines have network access to the list of **required outbound HTTPS endpoints** for Azure Local and Azure Arc.	To reliably manage, monitor, and operate Azure Local instances or workload resources, the required outbound network endpoints must have access, either directly or through a proxy server. A temporary interruption doesn't affect the running status of the workload but might affect manageability.
If you opt to create workload volumes (*virtual disks*) manually, use the most appropriate **resiliency type** to maximize workload resiliency and performance. For any user volumes that you create manually after you deploy the instance, create a storage path for the volumes in Azure. The volume can store workload VM configuration files, VM virtual hard disks (VHDs), and VM images via the storage path.	For Azure Local instances with three or more machines, consider using a three-way mirror to provide the highest resiliency and performance capabilities. We recommend that you use mirrored volumes for business-critical or mission-critical workloads.
Consider implementing **workload anti-affinity rules** to ensure that the VMs that host multiple instances of the same service run on separate physical hosts. This concept is similar to "*availability sets*" in Azure.	Make all components redundant. For business-critical or mission-critical workloads, use multiple Azure Arc VMs or Kubernetes replica sets or pods to deploy multiple instances of your applications or services. This approach increases resiliency if an unplanned outage of a single physical machine occurs.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload.

The **Security design principles** provide a high-level design strategy for achieving those goals by applying approaches to the technical design of Azure Local.

Azure Local is a secure-by-default product that has more than 300 security settings enabled during the cloud deployment process. Default security settings provide a consistent security baseline to ensure that devices start in a known good state. And you can use *drift protection controls* to provide at-scale management.

Default security features in Azure Local include hardened OS security settings, Windows Defender Application Control, volume encryption via BitLocker, secret rotation, local built-in user accounts, and Microsoft Defender for Cloud. For more information, see Review security features.

Design checklist

Start your design strategy based on the design review checklist for Security. Identify vulnerabilities and controls to improve the security posture. Extend the strategy to include more approaches as needed.

- ✔ (Azure Local platform architecture) **Review the security baselines**. Azure Local and security standards provide baseline guidance to strengthen the security posture of the platform and hosted workloads. If your workload needs to comply with specific regulatory compliance regulations, factor in the regulatory security standards, such as Payment Card Industry Data Security Standards and Federal Information Processing Standard 140.

  Azure Local platform-provided default settings enable security features, including identity controls, network filtering, and encryption. These settings form a good security baseline for a newly provisioned Azure Local instance. You can customize each setting based on your organizational security requirements.

  Make sure that you detect and protect against undesired security configuration drift.

- ✔ (Azure Local platform architecture) **Detect, prevent, and respond to threats**. Continuously monitor the Azure Local environment and protect against existing and evolving threats.

  We recommend that you enable Defender for Cloud on Azure Local. Enable the basic Defender for Cloud plan (*free tier*) by using Defender Cloud Security Posture Management to monitor and identify steps that you can take to secure your Azure Local platform, along with other Azure and Azure Arc resources.

  To benefit from the enhanced security features, including security alerts for individual servers and Azure Arc VMs, enable Microsoft Defender for Servers on your Azure Local instance machines and Azure Arc VMs.

- Use Defender for Cloud to measure the security posture of Azure Local machines and workloads. Defender for Cloud provides a single pane of glass experience to help manage security compliance.
- Use Defender for Servers to monitor the hosted VMs for threats and misconfigurations. You can also enable endpoint detection and response capabilities on Azure Local machines.
- Consider aggregating security and threat intelligence data from all sources into a centralized security information and event management (SIEM) solution, such as Microsoft Sentinel.

✓ (Azure Local platform architecture and workload architecture) **Create segmentation to contain the blast radius**. There are several strategies to attain segmentation.

- **Identity**. Keep roles and responsibilities for the platform and workload separate. Allow only authorized identities to carry out the specific operations that align with their designated roles. Azure Local platform administrators use both Azure and local domain credentials to do platform duties. Workload operators and application developers manage workload security. To simplify delegating permissions, use Azure Local built-in role-based access control (RBAC) roles, such as 'Azure Local Administrator' for platform administrators and 'Azure Local VM Contributor' or 'Azure Local VM Reader' for workload operators. For more information about specific built-in role *actions*, see Azure RBAC documentation for hybrid and multicloud roles.
- **Network**. Isolate networks if needed. For example, you can provision multiple logical networks that use separate virtual local area networks (vLANs) and network address ranges. When you use this approach, ensure that the management network can reach each logical network and vLAN so that Azure Local machines can communicate with the vLAN networks through the ToR switches or gateways. This configuration is required for availability management of the workload, such as allowing infrastructure management agents to communicate with the workload guest OS.
- Review Recommendations for building a segmentation strategy for additional information.

✓ (Azure Local platform architecture and workload architecture) **Use a trusted identity provider to control access**. We recommend Microsoft Entra ID for all authentication and authorization purposes. You can join a workload to an on-premises Windows Server Active Directory domain if required. Take advantage of

features that support strong passwords, multifactor authentication, RBAC, and controls for the management of secrets.

- ✓ (Azure Local platform architecture and workload architecture) **Isolate, filter, and block network traffic.** You might have a workload use case that requires virtual networks, microsegmentation via network security groups, network quality of service policies, or virtual appliance chaining so that you can bring in partner appliances for filtering. If you have such a workload, see software-defined network considerations for network reference patterns for a list of the supported features and capabilities that Network Controller provides.

- ✓ (Workload architecture) **Encrypt data to protect against tampering**. Encrypt data in transit, data at rest, and data in use.

  - Data-at-rest encryption is enabled on data volumes that you create during deployment. These data volumes include both infrastructure volumes and workload volumes. For more information, see Manage BitLocker encryption.

  - Use trusted launch for Azure Arc VMs to improve security of Gen 2 VMs by using OS features of modern operating systems, such as Secure Boot, which can use a virtual Trusted Platform Module.

- ✓ **Operationalize secret management**. Based on your organizational requirements, change the credentials that are associated with the deployment user identity for Azure Local. For more information, see Manage secrets rotation.

- ✓ (Azure Local platform architecture) **Enforce security controls**. Use Azure Policy to audit and enforce built-in policies, such as "*Application control policies should be consistently enforced*" or "*Encrypted volumes should be implemented*". You can use these Azure policies to audit security settings and assess the compliance status of Azure Local. For examples of the available policies, see Azure policies.

- ✓ (Workload architecture) **Improve workload security posture with built-in policies**. To assess Azure Arc VMs that run on Azure Local, you can apply built-in policies via the security benchmark, Azure Update Manager, or the Azure Policy guest configuration extension. You can use various policies to check the following conditions:
  - Log Analytics agent installation
  - Out-of-date system updates that need to be up to date with the latest security patches
  - Vulnerability assessment and potential mitigations
  - Use of secure communication protocols

Recommendations

Expand table

Recommendation	Benefit
Use the **security baseline and drift controls** settings to apply and maintain security settings on instance machines.	These configurations help to protect against unwanted changes and drift because they automatically refresh security settings every 90 minutes to enforce the intended security posture of Azure Local.
Use **Windows Defender Application Control** in Azure Local.	Windows Defender Application Control reduces the attack surface of Azure Local. Use the Azure portal or PowerShell to view policy settings and control policy modes. Windows Defender Application Control policies help to control which drivers and apps are allowed to run on your system.
Enable volume encryption via BitLocker for data encryption-at-rest protection.	BitLocker protects OS and data volumes by encrypting the instance shared volumes that are created on the Azure Local. BitLocker uses XTS-AES 256-bit encryption. We recommended that you keep the volume encryption default setting enabled during Azure Local cloud deployment for all data volumes.
Export **BitLocker recovery keys** to store them in a secure location that's external from the Azure Local instance.	You might need BitLocker keys during specific troubleshooting or recovery actions. We recommend that you export, save, and back up encrypt keys for OS and data volumes from each Azure Local instance via the 'Get-AsRecoveryKeyInfo' PowerShell cmdlet. Save the keys in a secure external location, such as Azure Key Vault.
Use a SIEM solution to increase security monitoring and alerting capabilities. To do so, you can **onboard Azure Arc-enabled servers (Azure Local platform machines) to Microsoft Sentinel**. Alternatively, if you use a different SIEM solution, configure **syslog forwarding of security events** to the chosen solution.	Forward security event data by using Microsoft Sentinel or syslog forwarding to provide alerting and reporting capabilities through integration with a customer-managed SIEM solution.
Use **Server Message Block (SMB) signing** to enhance data-in-transit protection, which is enabled in the "default security settings."	SMB signing allows you to digitally sign SMB traffic between an Azure Local platform and systems external to the platform (north or south). Configure signing for external SMB traffic

Recommendation	Benefit
	between the Azure Local platform and other systems to help prevent relay attacks.
Use the **SMB encryption** setting to enhance data-in-transit protection, which is enabled in the "default security settings."	The SMB encryption for in-instance traffic setting controls the encryption of traffic between physical machines in the Azure Local instance (east or west) on your storage network.

Cost Optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget while meeting business requirements.

The Cost Optimization design principles provide a high-level design strategy for achieving those goals and making tradeoffs as necessary in the technical design related to Azure Local and its environment.

Design checklist

Start your design strategy based on the design review checklist for Cost Optimization for investments. Fine-tune the design so that the workload is aligned with the budget that's allocated for the workload. Your design should use the right Azure capabilities, monitor investments, and find opportunities to optimize over time.

Azure Local incurs costs for hardware, software licensing, workloads, guest VMs (Windows Server or Linux) licensing, and other integrated cloud services, such as Azure Monitor and Defender for Cloud.

- ✓ (Azure Local platform architecture and workload architecture) **Estimate realistic costs as part of cost modeling**. Use the Azure pricing calculator to select and configure services like Azure Local, Azure Arc, and AKS on Azure Local. Experiment with various configurations and payment options to model costs.
- ✓ (Azure Local platform architecture and workload architecture) **Optimize the cost of Azure Local hardware**. Choose a hardware OEM partner that aligns with your business and commercial requirements. To explore the certified list of validated machines, integrated systems, and premier solutions, see Azure Local solutions catalog. Communicate your workload characteristics, size, quantity, and performance with your hardware partner so that you can rightsize a cost-effective hardware solution for the Azure Local machine and instance size.

- ✓ (Azure Local platform architecture) **Optimize your licensing costs**. Azure Local software is licensed and billed on a "*per physical CPU core*" basis. Use existing on-premises core licenses with Azure Hybrid Benefit to reduce licensing costs for Azure Local workloads, such as Azure Arc VMs that run Windows Server, SQL Server, or AKS and Azure Arc-enabled Azure SQL Managed Instance. For more information, see Azure Hybrid Benefit cost calculator.

- ✓ (Azure Local platform architecture) **Save on environment costs**. Evaluate whether the following options can help optimize your resource usage.

  - **Take advantage of discount programs that Microsoft offers**. Consider using Azure Hybrid Benefit to reduce the cost to run Azure Local and Windows Server workloads. For more information, see Azure Hybrid Benefit for Azure Local.

  - **Explore promotional offers**. Take advantage of the Azure Local 60-day free trial after registration for initial proof of concepts or validations.

- ✓ (Azure Local platform architecture) **Save on operational costs**.

  - **Evaluate technology options for patching, updating, and other operations**. Update Manager is free for Azure Local instances that have Azure Hybrid Benefit and Azure Arc VM management enabled. For more information, see Update Manager FAQ and Update Manager pricing.

  - **Evaluate costs related to observability**. Set up alert rules and data collection rules (DCRs) to meet your monitoring and auditing needs. The amount of data that your workload ingests, processes, and retains directly influences costs. Optimize by using smart retention policies, limiting the number and frequency of alerts, and choosing the right storage tier for storing logs. For more information, see Cost Optimization guidance for Log Analytics.

- ✓ (Workload architecture) **Evaluate density over isolation**. Use AKS on Azure Local to improve density and simplify workload management so that you can enable containerized applications to scale across multiple datacenter or edge locations. For more information, see AKS on Azure Local pricing.

Recommendations

Expand table

Recommendation	Benefit
Use **Azure Hybrid Benefit for Azure Local** if you have Windows Server Datacenter	With Azure Hybrid Benefit for Azure Local, you can maximize the value of your on-premises licenses

Recommendation	Benefit
licenses with Software Assurance.	and modernize your existing infrastructure to Azure Local at no additional cost.
Choose either the Windows Server subscription add-on or bring your own license to license and activate the Windows Server VMs and use them on Azure Local. For more information, see **License Windows Server VMs on Azure Local**.	While you can use any existing Windows Server licenses and activation methods available, optionally, you can enable "Windows Server subscription add-on" available for Azure Local only to subscribe Windows Server guest licenses through Azure which is charged for the total number of physical cores in the Azure Local instance.
Use the **Azure verification for VMs** benefit extended to Azure Local so that supported Azure-exclusive workloads can work outside of the cloud.	This benefit is enabled by default on Azure Local version 23H2 or later. Use this benefit so that the VMs can operate in other Azure environments and workloads can benefit from offers that are available only in Azure, such as Extended Security Updates enabled by Azure Arc.

Operational Excellence

Operational Excellence primarily focuses on procedures for **development practices, observability, and release management**.

The Operational Excellence design principles provide a high-level design strategy for achieving those goals for the operational requirements of the workload.

Monitoring and diagnostics are crucial. You can use metrics to measure performance statistics and to troubleshoot and remediate problems quickly. For more information about how to troubleshoot problems, see Operational Excellence design principles and Collect diagnostic logs for Azure Local.

Design checklist

Start your design strategy based on the design review checklist for Operational Excellence for defining processes for observability, testing, and deployment related to Azure Local.

- ✓ (Azure Local platform architecture) **Increase supportability of Azure Local**. Observability is enabled by default at the time of deployment. These capabilities enhance the supportability of the platform. Telemetry and diagnostic information is shared securely from the platform by using the *AzureEdgeTelemetryAndDiagnostics*

extension, which is installed on all Azure Local machines by default. For more information, see Azure Local observability.

- ✓ (Azure Local platform architecture) **Use Azure services to reduce operational complexity and increase management scale**. Azure Local is integrated with Azure to enable services such as Update Manager for patching the platform and Azure Monitor for monitoring and alerting. You can use Azure Arc and Azure Policy to manage security configuration and compliance auditing. Implement Defender for Cloud to help manage cyber threats and vulnerability. Use Azure as the control plane for these operational processes and procedures to help reduce complexity, improve efficiencies of scale, and improve management consistency.

- ✓ (Workload architecture) **Plan IP address network range requirements for workloads in advance**. Azure Local provides a platform to deploy and manage virtualized or containerized workloads. Also consider the IP address requirements for logical networks that your workload uses. Review these resources:

  - Azure Local (*platform*) network reference architecture and IP requirements
  - Workloads deployed on Azure Local require logical networks. For specific examples, see Network requirements for AKS clusters, Logical networks for Azure Arc VMs, and Virtual Desktop with Azure Local.

- ✓ (Workload configuration) **Enable monitoring and alerting for workloads that you deploy on Azure Local**. You can use Azure Monitor for virtual machines, or VM Insights for Arc VMs, or use Container Insights and managed Prometheus AKS clusters.

  Evaluate whether you should use a centralized Log Analytics workspace for your workload. For an example of a shared log sink (*data location*), see Workload management and monitoring recommendations.

- ✓ (Azure Local platform architecture) **Use proper validation techniques for a safe deployment**. Use the environmental checker tool in standalone mode to assess the readiness of the target environment before you deploy an Azure Local solution. This tool validates the proper configuration of required connectivity, hardware, Windows Server Active Directory, networks, and Azure Arc integration prerequisites.

- ✓ (Azure Local platform architecture) **Get current and stay current**. Use the Azure Local solution catalog to stay current with the latest hardware OEM innovations for Azure Local instance deployments. Consider using premium solutions to benefit from extra integration, turn-key deployment capabilities, and a simplified update experience.

Use Update Manager to update the platform and manage the OS, core agents, and services, including solution extensions. Stay current, and consider using the "Enable automatic upgrade" setting where possible for extensions.

Recommendations

Expand table

Recommendation	Benefit
Enable Monitor Insights on Azure Local instances to enhance monitoring and alerting by using native Azure capabilities. Insights can monitor key Azure Local features by using the instance performance counters and event log channels that are collected by the DCR. For certain hardware infrastructure, such as Dell APEX, you can visualize hardware events in real time. For more information, see Feature workbooks.	Azure manages Insights, so it's always up to date, it's scalable across multiple instances, and it's highly customizable. Insights provides access to default workbooks with basic metrics, along with specialized workbooks that are created for monitoring key features of Azure Local. This feature provides near real-time monitoring. You can create graphs and customized visualization by using aggregation and the filter functionality. You can also configure custom alert rules. The cost of Insights is based on the quantity of data ingested and the data retention settings of the Log Analytics workspace. When you enable Azure Local Insights, we **recommended** that you use the DCR created by the Insights creation experience. The prefix of the DCR name is `AzureStackHCI-`. It's configured to collect only the required data.
Set up alerts, and configure the alert processing rules based on your organizational requirements. Get notified of changes in health, metrics, logs, or other types of observability data. - Health alerts - Log alerts - Metric alerts For more information, see Recommended rules for metric alerts.	Integrate Monitor alerts with Azure Local to get several key benefits at no extra cost. Get near real-time monitoring and customize alerts to notify the right team or admin for remediation. You can collect a comprehensive list of metrics for compute, storage, and network resources in Azure Local. Perform advanced logic operations on your log data and evaluate metrics of your Azure Local instance at regular intervals.
Use the update feature to integrate and manage various aspects of the Azure Local solution in one place. For more	The update orchestrator is installed during the initial Azure Local instance deployment. This feature automates updates and management operations. To keep Azure Local in a supported state, make sure that

Recommendation	Benefit
information, see **About updates in Azure Local**.	you update your instances on a regular cadence to move to new baseline builds when they become available. This method provides new capabilities and improvements to the platform. For more information about *release trains*, the cadence of updates, and the support window of each baseline build, see Azure Local version 23H2 release information.
To help with hands-on skilling, labs, training events, product demos, or proof-of-concept projects, consider using Jumpstart HCIBox. Rapidly deploy Azure Local without the need for physical hardware by using a VM on Azure to deploy the solution.	HCIBox supports Azure Local version 23H2 to enable rapid testing and evaluation of the latest capabilities of Azure edge products, such as native Azure Arc and AKS integration in a self-contained sandbox. You can deploy this sandbox to an Azure subscription by using a VM that supports nested virtualization to emulate an Azure Local instance inside an Azure VM. Get Azure Local features like the new cloud deployment feature with minimal manual effort. For more information, see Microsoft Tech Community blog.

Performance Efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance.

The Performance Efficiency design principles provide a high-level design strategy for achieving those capacity goals against the expected usage.

Design checklist

Start your design strategy based on the design review checklist for Performance Efficiency. Define a baseline that's based on key indicators for Azure Local.

- ✓ (Azure Local platform architecture) **Use the Azure Local-validated hardware** or integrated systems from OEM partner offerings. Consider using the premium solution builders in the Azure Local catalog to optimize the performance of your Azure Local environment.

- ✓ (Azure Local platform storage architecture) Choose the right **physical disk types for the Azure Local machines** based on your workload performance and capacity requirements. For high-performance workloads that require low latency and high-throughput storage, consider using an all-flash (NVMe/SSD only) storage configuration. For general purpose compute or large storage capacity requirements, consider using hybrid storage (SSD or NVMe for cache tier and HDDs for capacity tier), which might provide increased storage capacity.

- ✓ (Azure Local platform architecture) **Use the Azure Local sizer tool during the instance design (*pre-deployment*) phase**. Azure Local instances should be sized appropriately by using the workload capacity, performance, and resiliency requirements as inputs. The size determines the maximum number of physical machines that can be offline simultaneously (*cluster quorum*), such as any planned (*maintenance*) or unplanned (*power or hardware failure*) events. For more information, see Cluster quorum overview.

- ✓ (Azure Local platform architecture) **Use all-flash (NVMe or SSD) based solutions for workloads that have high-performance or low-latency requirements**. These workloads include but are not limited to highly transactional database technologies, production AKS clusters, or any mission-critical or business-critical workloads with low-latency or high-throughput storage requirements. Use all-flash deployments to maximize storage performance. All-NVMe or all-SSD configurations (*especially at a very small scale*) improve storage efficiency and maximize performance because no drives are used as a cache tier. For more information, see All-flash-based storage.

- ✓ (Azure Local platform architecture) **Establish a performance baseline for Azure Local instance storage** before you deploy production workloads. Configure **Monitor Azure Local features with Insights** to monitor the performance of a single Azure Local instance or multiple instances simultaneously.

- ✓ (Azure Local platform architecture) **Consider using the Monitor for Resilient File System (ReFS) deduplication and compression feature** after you enable Insights for the Azure Local instance. Determine whether you should use this feature based on your workload storage usage and capacity requirements. This feature provides monitoring for ReFS deduplication and compression savings, performance impact, and jobs. For more information, see Monitor ReFS deduplication and compression.

  As a minimum requirement, plan to reserve `1 x physical machines (N+1)` worth of capacity across the instance to ensure that instance machines can be drained when they perform updates via Update Management. Consider reserving `2 physical machines (N+2)` machines work of capacity for business-critical or mission-critical use cases.

Recommendations

Expand table

Recommendation	Benefit
If you select the advanced option to "*create infrastructure volumes only*" during Azure Local instance deployment, we recommend that you **create the virtual disks by using mirroring** when you create workload volumes for performance-intensive workloads.	This recommendation benefits workloads that have strict latency requirements or that need high throughput with a mix of random read and write input/output operations per second (IOPs), such as SQL Server databases, Kubernetes clusters, or other performance-sensitive VMs. Deploy the workload VHDs on volumes that use mirroring to maximize performance and resiliency. Mirroring is faster than any other resiliency type.
Consider using **DiskSpd to test workload storage performance** capabilities of the Azure Local instance. You can also use VMFleet to generate load and measure the performance of a storage subsystem. Evaluate whether you should use VMFleet for measuring storage subsystem performance.	Establish a baseline for Azure Local instance performance before you deploy production workloads. DiskSpd allows administrators to test the storage performance of the instance by using various command line parameters. The main function of DiskSpd is to issue read and write operations and output performance metrics, such as latency, throughput, and IOPs.

Tradeoffs

There are design tradeoffs with the approaches described in the pillar checklists. Here are some examples of advantages and drawbacks.

Building redundancy increases costs

- Understand your workload's requirements up front, such as the workload RTO and RPO targets and storage performance requirements (IOPs and throughput), when you design and procure the hardware for an Azure Local solution. To deploy highly available workloads, we recommend a minimum of a three-machine instance, which enables three-way mirroring for workload volumes and data. For the compute resources, ensure that you deploy a minimum of "N+1 number of physical machines", which *reserves the capacity of a "single machine worth of space" in the instance at all times*. For business-critical or mission-critical workloads, consider reserving "*N+2 machines worth of capacity*" to provide increased resiliency. For example, if two machines in the instance are offline, the workload can remain online. This approach provides increased resiliency for a scenario such

as, if a machine running workload goes offline during a planned update procedure (*resulting in two machines being offline simultaneously*).

- For business-critical or mission-critical workloads, we recommend that you deploy two or more separate Azure Local instances and deploy multiple instances of your workload services across the separate instances. Use a workload design pattern that takes advantage of data replication and application load balancing technologies. For example, SQL Server always-on availability groups use synchronous or asynchronous database replication to achieve low RTO and RTO targets across separate instances in different datacenters.
- Consequently, an increase in workload resiliency and a decrease in RTO and RPO targets increases costs and requires well-architected applications and operational rigor.



Providing scalability without effective workload planning increases costs

- Incorrect instance sizing can lead to insufficient capacity or reduced return on investment (ROI) if the hardware is overprovisioned. Both scenarios affect costs.
- **Increased capacity equals higher costs**. During the Azure Local instance design phase, adequate planning is required to *rightsize the capabilities and number of instance machines* based on workload capacity requirements. Therefore, you must understand the workload requirements (*vCPUs, memory, storage, and X number of VMs*) and allow for some extra *headroom* in addition to projected growth. You can perform an add-machine gesture when you use a "*storage switched*" design. But it can take a long time to get more hardware after your deployment. And an add-note gesture is more complex than sizing the instance hardware and number of machines (*maximum 16 machines*) appropriately during the initial deployment.
- There are disadvantages if you overprovision the machine hardware specification and select the incorrect number of machines (*size of the instance*). For example, if the workload requirements are much smaller than the instance's overall capacity and the hardware is underused throughout the hardware warranty period, the ROI value might decrease.

Azure policies

Azure provides an extensive set of built-in policies related to Azure Local and its dependencies. Some of the preceding recommendations can be audited through Azure Policy. For example, you can check whether:

- Host and VM networking should be protected.
- Encrypted volumes should be implemented.
- Application control policies should be consistently enforced.
- Secured-core requirements should be met.

Review the Azure Local built-in policies. Defender for Cloud has new recommendations that show the compliance state for the built-in policies. For more information, see Built-in policies for Azure Security Center.

If your workload runs on Azure Arc VMs that you deploy on Azure Local, consider built-in policies, such as denying the creation or modification of Extended Security Updates licenses. For more information, see Built-in policy definitions for Azure Arc-enabled workloads.

Consider creating custom policies to provide extra governance for both the Azure Local resources and Azure Arc VMs that you deploy on an Azure Local instance. For example:

- Auditing Azure Local host registration with Azure
- Ensuring that hosts run the latest OS version
- Checking for required hardware components and network configurations
- Verifying the enablement of necessary Azure services and security settings
- Confirming the installation of required extensions
- Assessing the deployment of Kubernetes clusters and AKS integration

Azure Advisor recommendations

Azure Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments. Here are some recommendations that can help you improve the reliability, security, cost effectiveness, performance, and operational excellence of your VMs.

- Reliability
- Security
- Cost Optimization
- Performance
- Operational Excellence

Next steps

- Consider the following articles in Azure Architecture Center as resources that demonstrate the recommendations highlighted in this article.

    - Foundational architecture that demonstrates the key recommendations: Azure Local baseline reference architecture.
    - If your organization needs a hybrid approach, carefully select your design choices related to a hybrid network architecture. For more information, see Hybrid architecture design.

- Build implementation expertise by using the following Azure Local product documentation:
    - Azure Local overview
    - Azure Local network deployment considerations
    - About Azure Local version 23H2 deployment

- Review Cloud Adoption Framework guidance:

  The Cloud Adoption Framework Ready methodology guides customers when they prepare their environment for cloud adoption. The methodology includes technical accelerators like Azure landing zones, which are the building blocks of any Azure cloud adoption environment. Review the following articles for more details about how to prepare your environment for a hybrid cloud.
    - Prepare your environment for hybrid and multicloud scenarios
    - Introduction to Azure Arc landing zone accelerator for hybrid and multicloud

---

Feedback

Was this page helpful? Yes No

Architecture best practices for Azure Traffic Manager

Article • 02/03/2025

Azure Traffic Manager is a global load balancer that can distribute traffic across multiple Azure regions, zones within a region, or datacenters within those zones. It uses the Domain Name System (DNS) protocol to establish a communication path between a client and your workload's endpoints. After the connection is established, clients can connect directly to the endpoint without the help of Traffic Manager.

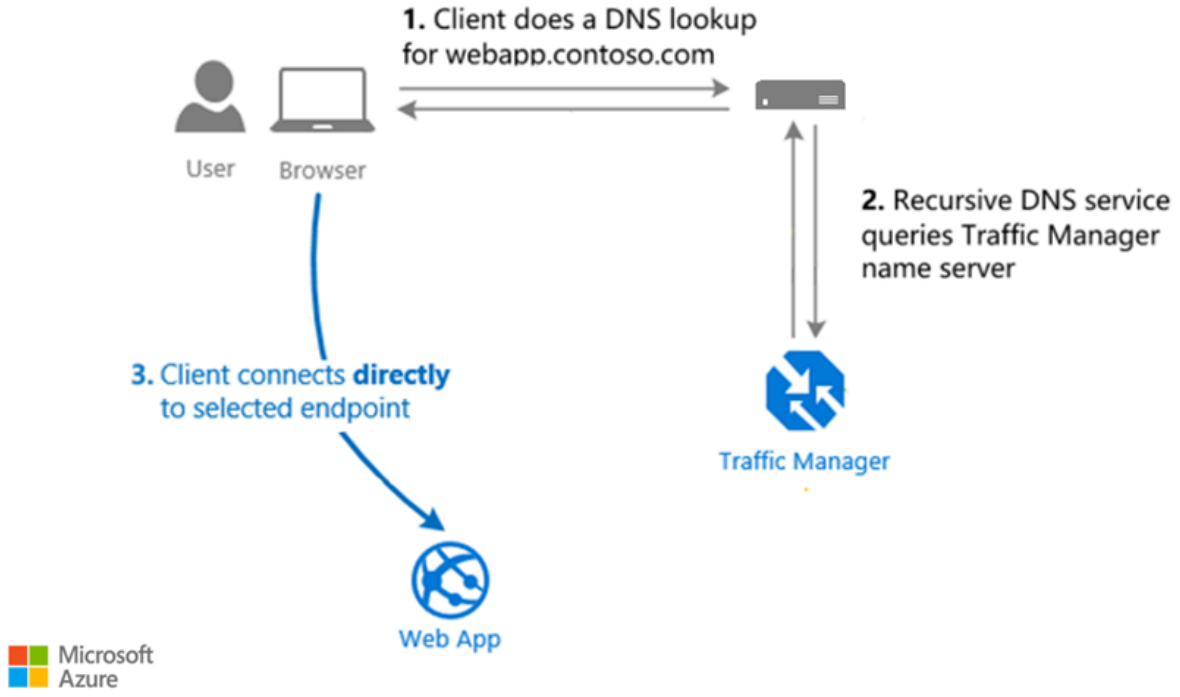
This article assumes that as an architect, you've reviewed the load-balancing options in Azure and chose Azure Traffic Manager for your workload, which is deployed across multiple regions in either an active-active or active-passive model. The guidance in this article provides architectural recommendations that are mapped to the principles of the Well-Architected Framework pillars.

> **ⓘ Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern along with design strategies localized to the technology scope.
>
> Also included are recommendations for the technology capabilities that can help materialize those strategies. The recommendations don't represent an exhaustive list of all configurations that are available for Traffic Manager and its dependencies. Instead, they list the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept or to optimize your existing environments.
>
> Foundational architecture that demonstrates the key recommendations: Multiregion load balancing with Traffic Manager, Azure Firewall, and Application Gateway.

Technology scope

This review focuses on the interrelated decisions for the following Azure resource:

- Traffic Manager



> ⓘ Note
>
> For workloads that host HTTP applications, Azure Front Door is a natural choice because of its features, such as a content delivery network, Transport Layer Security (TLS) termination, and an integrated firewall.
>
> Compared to Azure Front Door, Traffic Manager is easier to set up, configure, and maintain. Traffic Manager doesn't have an endpoint that you can control directly. Unlike Front Door, which handles client requests, Traffic Manager only connects clients to a workload's endpoint.
>
> But this simplicity comes with tradeoffs that can introduce complexities in an architecture. For instance, you might need to implement extra security measures to block Open Worldwide Application Security Project (OWASP) attack types. A Web Application Firewall (WAF) in Azure Front Door or Azure Application Gateway provides this capability. Or you might add a cache, which can speed up content delivery but adds complexity because you must manage a data store.
>
> For more information, see Well-Architected Framework perspective on Azure Front Door.

Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover fast from failures.**

Reliability design principles provide a high-level design strategy applied for individual components, system flows, and the system as a whole.

Design checklist

Start your design strategy based on the design review checklist for Reliability. Determine its relevance to your business requirements while keeping in mind the nature of your application and the criticality of its components. Extend the strategy to include more approaches as needed.

- ✓ **Account for potential failures.** Traffic Manager is designed to be resilient. But it can still be a single point of failure for your workload. To mitigate this risk, define a secondary path to an alternate service that becomes active if Traffic Manager is unavailable. To avoid routing problems, don't use Traffic Manager and the alternate service together.

- ✓ **Have a good understanding of service-level agreement (SLA) coverage.** When you evaluate Traffic Manager SLAs, understand the coverage related to the published percentile. For example, your DNS lookups might fail multiple times. Those failures aren't considered downtime until a full minute of continuous DNS lookup failures occur.

- ✓ **Incorporate redundancy in your workload architecture.** If your service is exposed through a public IP address, use Traffic Manager to implement redundancy across Azure regions, on-premises, and other clouds. For example, you might have an on-premises application that has a secondary instance in the cloud. If the on-premises system fails, the cloud instance can become active, which helps ensure continuity.

- ✓ **Use a reliable deployment architecture to support redundancy.** As a load balancer, Traffic Manager distributes traffic across workload endpoints based on how you configure its routing method. You define this configuration in a Traffic Manager profile. The profile is a core component of your deployment strategy. You can use the appropriate profile configuration to implement an active-active model or an active-passive model that has a warm spare.

  Each profile specifies a single traffic-routing method. Some scenarios require more sophisticated traffic routing. You can combine Traffic Manager profiles to take advantage of more than one traffic-routing method.

  For more information, see Traffic Manager routing methods.

- ✓ **Evaluate the caching duration of DNS responses.** The time-to-live (TTL) setting for DNS lookups in Traffic Manager determines how long downstream DNS resolvers cache DNS responses. The default TTL might result in longer caching times than necessary, potentially causing downtime if an endpoint fails. Reduce the TTL to increase the frequency of cache updates. This method can help mitigate downtime but increases the frequency of DNS lookups.

- ✓ **Avoid sending traffic to unhealthy or compromised instances.** Review the built-in health probing features in Traffic Manager.

  For HTTPS and HTTP applications, implement the Health Endpoint Monitoring pattern to provide a custom page within your application. Based on specific checks, the page returns an appropriate HTTPS status code. In addition to the availability of the endpoint, your health check should monitor all dependencies of your application.

  For other applications, Traffic Manager uses the Transmission Control Protocol (TCP) to determine the availability of the endpoint.

  For more information, see Health Endpoint Monitoring pattern and Understand Traffic Manager probes.

- ✓ **Determine your outage tolerance.** If a back end becomes unavailable, some time can pass before Traffic Manager recognizes the failure and stops directing traffic to the unavailable endpoint. There's a period of time when client requests can't be served. Use this tolerance to configure probe settings, which determine how quickly you want to start your business-continuity operations.

- ✓ **Include the endpoints as part of your resiliency testing.** Simulate unavailable endpoints to see how Traffic Manager handles failures. Suppose your workload uses a load balancer like Application Gateway in a private virtual network. You can use network security group (NSG) rules in Azure Chaos Studio to simulate failures of the endpoint. You can block access to the subnet where Application Gateway resides.

Recommendations

Expand table

Recommendation	Benefit
Deploy multiple endpoints in the Traffic Manager profiles, and enable them. If an endpoint isn't enabled, it's not probed for health checks or included in the	Redundant instances help ensure availability if one endpoint fails.

Recommendation	Benefit
traffic-routing rotation. Place these endpoints in different regions.	
Evaluate the various traffic-routing methods. Configure one or a combination of methods to align with your deployment strategy. Traffic Manager applies the chosen method to each DNS query and uses the method to determine which endpoint is returned in the DNS response. - The weighted method distributes traffic based on the configured weight coefficient. This method supports active-active models. - The priority-based method configures the primary region to receive traffic and send it to the secondary region as a backup. This method supports active-passive models. - The geographic method routes traffic based on the geographic origin of the DNS query. To cover all regions, configure at least one endpoint with the All (World) property.	An optimized routing method helps ensure that you distribute traffic efficiently across your endpoints. You can support your active-active or active-passive deployment model goals. An efficient routing method helps ensure that secondary regions can handle traffic or act as a backup. Geographical routing helps direct users to the nearest endpoint based on their location. It helps ensure that traffic is efficiently managed and not lost.
Set the DNS TTL interval duration to a low value, preferably less than 60 seconds. To optimize performance, adjust the health probe timing and the DNS record TTL.	A low TTL duration helps ensure more frequent downstream DNS resolver cache updates and quicker failover. It also minimizes downtime and improves the overall responsiveness of your application.
Set up health probes to monitor the endpoint. - Don't enable `AlwaysServe`, which disables endpoint monitoring and sends requests to the endpoint, regardless of its health state. - Set the `probing interval` value. Consider the tradeoff between how fast you can detect failures and the number of requests to the endpoint. The number of requests can be substantial because Traffic Manager is a global service that pings simultaneously from various locations. - Set the `probe timeout` value. Consider how long to wait before declaring the endpoint unhealthy. Include false positives in the number of failures.	Health probes make sure that only healthy instances serve user requests. They also help determine whether failures are nontransient and how quickly failover operations should take place.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload.

The Security design principles provide a high-level design strategy for achieving those goals by applying approaches to the technical design of Traffic Manager.

Design checklist

- ✓ **Review security baselines.** To enhance your security posture, review the security baseline for Traffic Manager.
- ✓ **Prevent unauthorized modification of traffic routing.** Treat Traffic Manager profiles as critical workload resources because they contain the configuration settings for routing traffic. Only authorized identities should have access to these profiles. Implement role-based access control (RBAC) on the control plane to restrict tasks like creating, deleting, or modifying resources. Only authorized identities should have the authority to enable or disable endpoints. Unauthorized access can result in configuration changes and potentially reroute traffic to malicious implementations.
- ✓ **Protect applications from threats at the network edge.** Traffic Manager doesn't provide built-in security features, like a WAF. To help secure HTTP applications, you should implement traffic inspection at the endpoint level.

  A typical architecture might include Traffic Manager and multiple endpoints. For each endpoint, an application gateway that handles TLS termination and other security functions provides protection. For a reference architecture that demonstrates that pattern, see Multiregion load balancing with Traffic Manager, Azure Firewall, and Application Gateway.
- ✓ **Harden DNS entries.** Traffic Manager can be susceptible to attacks that manipulate DNS data, which can redirect traffic to malicious sites and cause security problems. A common threat is a subdomain takeover, in which a DNS record points to a deprovisioned Azure resource. To prevent subdomain takeovers, use Azure DNS alias records and couple the lifecycle of a DNS record with an Azure resource.

  For more information, see Prevent dangling DNS entries.

Recommendations

Expand table

Recommendation	Benefit
Add application gateways to the workload endpoints. To implement security inspection with a firewall for HTTP applications, add application gateways to the workload endpoints.	You can inspect inbound HTTP traffic to help protect the application from common attacks.
Create an alias record in Azure DNS for your workload's apex domain name to reference a Traffic Manager profile.	Alias records tightly couple the lifecycle of a DNS record with an Azure resource. This configuration helps prevent dangling references if the workload is decommissioned. If the Traffic Manager profile is deleted, the DNS alias record becomes an empty record set. It no longer references the deleted resource.

Cost Optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget while meeting business requirements.

The Cost Optimization design principles provide a high-level design strategy for achieving those goals and making tradeoffs as necessary in the technical design related to Traffic Manager and its environment.

Design checklist

Start your design strategy based on the design review checklist for Cost Optimization for investments. Fine-tune the design so that the workload is aligned with the budget that's allocated for the workload. Your design should use the right Azure capabilities, monitor investments, and find opportunities to optimize over time.

- ✓ **Evaluate the cost of features.** The traffic view dashboard feature shows where your clients connect from and the associated latency. This information helps optimize performance and refine your design, which contributes to operational excellence and system efficiency. However, it incurs extra costs. For more information, see Traffic view billing.

  Also consider the cost associated with health probes. Traffic Manager pings your defined endpoints from various locations to check their availability. You can choose

slow pings and fast pings. Fast pings detect failures quicker but incur higher costs. They also add more load on the workload because health checks are more frequent.

- ✓ **Evaluate the cost of your routing strategy.** For example, if most clients access your endpoint from a high-latency region, you could create another endpoint closer to those users and adjust the routing method in Traffic Manager. This practice reduces latency so that you can process more requests with less capacity, which leads to cost savings.

Recommendations

Expand table

Recommendation	Benefit
Use the pricing calculator to estimate costs of Traffic Manager features.	You can have a more accurate cost model and set governance around resources, if necessary.
Enable the traffic view dashboard during your optimization efforts.	The feature helps you better understand usage patterns. Use this data for performance tuning to help meet your workload targets. Enable features at the right time, and apply the right tier to avoid underusing resources.
Choose between fast or slow pings for health probes, depending on your recovery metrics. Fast pings detect failures quicker but cost more. Slow pings are slower but cost less. Don't disable pings.	Ping less frequently to optimize cost and reduce the load on workload endpoints.

Operational Excellence

Operational Excellence primarily focuses on procedures for **development practices, observability, and release management.**

The Operational Excellence design principles provide a high-level design strategy for achieving those goals for the operational requirements of the workload.

Design checklist

- ✓ **Collect and analyze operational data as part of your workload monitoring.** Capture relevant Traffic Manager logs and metrics. Use this data to troubleshoot, understand traffic behaviors, and fine-tune routing logic.

- ✓ **View traffic data for profiles.** This data helps pinpoint areas for improvement, like expanding your Azure presence in high-latency regions. It also highlights traffic patterns across different regions to help you determine where to increase or decrease investment.

- ✓ **Implement disaster recovery operations**. Your disaster recovery implementation can be designed to reroute network/web traffic from the main site to a backup site. This disaster recovery method can be implemented using Azure DNS and Azure Traffic Manager (DNS). In the event of a disaster, if the primary endpoint becomes degraded, Traffic Manager redirects traffic to a healthy secondary endpoint. By default, Traffic Manager gives priority to the primary endpoint, but it can also be set up with extra failover endpoints or load balancers to distribute the traffic load. For more information, see Set up disaster recovery and outage detection.

- ✓ **Avoid automatic-failback operations.** When you do a failback, don't use automatic failback, which immediately switches back to the original endpoint in the primary when it's available. Instead, disable the original endpoint, and use the secondary endpoint until you want to switch. This approach provides time for stabilization. Immediate failback can create extra load and delays.

  For more information, see Multiregion load balancing reference architecture.

- ✓ **Test configuration settings.** Misconfigurations can affect all aspects of the workload, especially reliability. Test the configurations with multiple clients at various locations. For more information, see Verify Traffic Manager settings.

Recommendations

Expand table

Recommendation	Benefit
Enable diagnostic logs for a Traffic Manager profile. Use tools to replay logs and analyze the health check data.	Diagnostic logs provide insights about the behavior of the Traffic Manager profile. For example, you can identify the reasons for individual probe time-outs against an endpoint.
Enable the traffic view dashboard. Get insights about where your clients	This information helps optimize performance and cost because you can refine your design.

Recommendation	Benefit
connect from and the associated latency.	
Take advantage of the Heat Map REST API, which provides data about client locations and latency.	This approach offers flexibility, such as setting a specific time period. You can add data to custom tooling or dashboards. You can also use this API to integrate with external tools.
Disable the endpoints for operational activities. For example, you might disable failback after a failover to do maintenance or testing.	Disabling the endpoint from the load balancer is beneficial for operational tasks because you can't stop live traffic. During maintenance, instances don't receive traffic if you disable the endpoint. This approach prevents automatic failback.

Performance Efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance.

The Performance Efficiency design principles provide a high-level design strategy for achieving those capacity goals against the expected usage.

Design checklist

Start your design strategy based on the design review checklist for Performance Efficiency. Define a baseline that's based on key performance indicators for Traffic Manager.

- ✓ **Measure the performance impact of your configuration.** Traffic Manager doesn't interfere with the direct connection between the client and your endpoint. The main performance impact is the initial DNS lookup. The TTL setting affects how often this lookup occurs. A lower TTL means more frequent DNS resolutions, which can slightly affect performance. If you set the TTL to zero, every request requires a DNS lookup, which adds a small delay before accessing the endpoint.

  Because Traffic Manager operates at the DNS level, it doesn't support session affinity. Traffic Manager might direct users to different endpoints for each call. If you need session affinity, you must persist state to a separate data store.

  For more information, see Performance considerations for Traffic Manager.

✓ **Adjust traffic-routing behavior to optimize performance.** A single profile can have multiple endpoints, but you can apply only one routing method at a time.

For more complex scenarios, consider creating a hierarchy of profiles to combine different routing methods. For example, you might prioritize regions and then use performance-based routing within regions.

Recommendations

Expand table

Recommendation	Benefit
Use the performance routing method when you have endpoints in different geographic locations.	This method prioritizes sending traffic to the endpoint that has the lowest latency. This method helps ensure fast service for users.
Use specialized tools to optimize performance. Measure the performance of your DNS lookups. To analyze the performance of traffic, use the traffic view dashboard or the Heat Map REST API.	DNS latency measurement tools perform a full DNS lookup and provide performance data. You can use this data to set the TTL duration and optimize performance.
Combine traffic methods with nested profiles for optimal performance, as determined by your workload requirements.	An optimized routing method helps ensure that traffic is directed to the most responsive and closest endpoints. This approach helps improve application performance and user experience.
Use the real user measurements feature to view network latency measurements to Azure regions. This feature is available at no extra cost.	You can make data-driven routing decisions to direct queries to the Azure region that provides the lowest latency.
Set the DNS TTL interval to a higher value.	Clients can cache the results for a longer period of time, which reduces the need to resolve DNS each time.

Azure policies

Azure provides an extensive set of built-in policies related to Traffic Manager and its dependencies. Some of the preceding recommendations can be audited through Azure Policy. For example, you can check whether:

- Resource logs are enabled to track activities.
- Logs for Traffic Manager profiles are sent to Azure Event Hubs.

For comprehensive governance, review the Azure Policy built-in definitions for Azure networking services.

Azure Advisor recommendations

Azure Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments. Here are some recommendations that can help you improve the reliability, security, cost effectiveness, performance, and operational excellence of your web application instances.

- Reliability
- Security
- Cost Optimization
- Performance
- Operational Excellence

Tradeoffs

You might have to make design tradeoffs if you use the approaches in the pillar checklists. Here are some examples of advantages and drawbacks.



Reliability and Performance Efficiency

- **The TTL setting for DNS lookups.** The default TTL setting can result in longer caching times. Long caching times can cause downtime if an endpoint fails because the application continues to attempt a connection to the failed endpoint for the TTL duration.

  Reduce the TTL to help mitigate this problem. A lower TTL has more frequent updates and quicker failover, but it increases the frequency of DNS lookups. This approach can affect performance and increase the load on DNS servers.



Reliability and Cost Optimization

- **Health probes.** Traffic Manager uses health probes to ping your endpoints from various locations to check their availability. You can choose slow pings or fast pings. Fast pings detect failures quicker but add cost. Slow pings take longer to detect failures but cost less. Balance the speed of failure detection and recovery with the associated costs.

Next step

Consider the following resources that demonstrate the recommendations in this article.

- Use the following reference architecture as an example of how you can apply this article's guidance to a workload:
  - Multiregion load balancing with Traffic Manager, Azure Firewall, and Application Gateway
- Use the following product documentation to improve your implementation expertise:
  - Traffic Manager

Feedback

Was this page helpful? Yes No

Event Grid and reliability

Article • 11/14/2023

Azure Event Grid lets you easily build applications with event-based architectures. This solution has build-in support for events coming from Azure services, like storage blobs and resource groups. Event Grid also has support for your own events, using custom topics.

For more information about using Event Grid, reference Create and route custom events with Azure Event Grid.

To understand how using Event Grid creates a more reliable workload, reference Server-side geo disaster recovery in Azure Event Grid.

The following sections are specific to Azure Event Grid and reliability:

- Design considerations
- Configuration checklist
- Recommended configuration options
- Source artifacts

Design considerations

Azure Event Grid provides an uptime SLA. For more information, reference SLA for Event Grid.

Checklist

Have you configured Azure Event Grid with reliability in mind?

- ✓ Deploy an Event Grid instance per region, in case of a multi-region Azure solution.
- ✓ Monitor Event Grid for failed event delivery.
- ✓ Use batched events.
- ✓ Event batches can't exceed `1MB` in size.
- ✓ Configure and optimize batch-size selection during load testing.
- ✓ Ensure Event Grid messages are accepted with `HTTP` `200-204` responses only if delivering to an endpoint that holds custom code.
- ✓ Monitor Event Grid for failed event publishing.

Configuration recommendations

Consider the following recommendations to optimize reliability when configuring Azure Event Grid:

Recommendation	Description
Monitor Event Grid for failed event delivery.	The `Delivery Failed` metric will increase every time a message can't be delivered to an event handler (timeout or a non-`200-204 HTTP` status code). If an event can't be lost, set up a Dead-Letter-Queue (DLQ) storage account. A DLQ account is where events that can't be delivered after the maximum retry count will be placed. Optionally, implement a notification system on the DLQ storage account, for example, by handling a *new file* event through Event Grid.
Use batched events in high-throughput scenarios.	The service will deliver a `json` array with multiple events to the subscribers, instead of an array with one event. The consuming application must be able to process these arrays.
Event batches can't exceed `1MB` in size.	If the message payload is large, only one or a few messages will fit in the batch. The consuming service will need to process more event batches. If your event has a large payload, consider storing it elsewhere, such as in blob storage, and passing a reference in the event. When integrating with third-party services through the CloudEvents schema, it's not recommended to exceed `64KB` events.
Configure and optimize batch-size selection during load testing.	Batch size selection depends on the payload size and the message volume.
Monitor Event Grid for failed event publishing.	The `Unmatched` metric will show messages that are published, but not matched to any subscription. Depending on your application architecture, the latter may be intentional.

Source artifacts

To determine the **Input Schema** type for all available Event Grid topics, use the following query:

SQL

```
Resources
| where type == 'microsoft.eventgrid/topics'
| project name, resourceGroup, location, subscriptionId,
properties['inputSchema']
```

To retrieve the **Resource ID** of existing private endpoints for Event Grid domains, use the following query:

SQL

```
Resources
| where type == 'microsoft.eventgrid/domains' and
notnull(properties['privateEndpointConnections'])
| mvexpand properties['privateEndpointConnections']
| project-rename privateEndpointConnections =
properties_privateEndpointConnections
| project name, resourceGroup, location, subscriptionId,
privateEndpointConnections['properties']['privateEndpoint']['id']
```

To identify **Public Network Access** status for all available Event Grid domains, use the following query:

SQL

```
Resources
| where type == 'microsoft.eventgrid/domains'
| project name, resourceGroup, location, subscriptionId,
properties['publicNetworkAccess']
```

To identify **Firewall Rules** for all public Event Grid domains, use the following query:

SQL

```
Resources
| where type == 'microsoft.eventgrid/domains' and
properties['publicNetworkAccess'] == 'Enabled'
| project name, resourceGroup, location, subscriptionId,
properties['inboundIpRules']
```

To identify **Firewall Rules** for all public Event Grid topics, use the following query:

SQL

```
Resources
| where type == 'microsoft.eventgrid/topics' and
properties['publicNetworkAccess'] == 'Enabled'
| project name, resourceGroup, location, subscriptionId,
properties['inboundIpRules']
```

To retrieve the **Resource ID** of existing private endpoints for Event Grid topics, use the following query:

SQL

```
Resources
| where type == 'microsoft.eventgrid/topics' and
notnull(properties['privateEndpointConnections'])
| mvexpand properties['privateEndpointConnections']
| project-rename privateEndpointConnections =
properties_privateEndpointConnections
| project name, resourceGroup, location, subscriptionId,
privateEndpointConnections['properties']['privateEndpoint']['id']
```

To determine the **Input Schema** type for all available Event Grid domains, use the following schema:

SQL

```
Resources
| where type == 'microsoft.eventgrid/domains'
| project name, resourceGroup, location, subscriptionId,
properties['inputSchema']
```

To identify **Public Network Access** status for all available Event Grid topics, use the following query:

SQL

```
Resources
| where type == 'microsoft.eventgrid/topics'
| project name, resourceGroup, location, subscriptionId,
properties['publicNetworkAccess']
```

Next step

Event Grid and operational excellence

Event Grid and operational excellence

Article • 11/14/2023

Azure Event Grid lets you easily build applications with event-based architectures. This solution has build-in support for events coming from Azure services, like storage blobs and resource groups. Event Grid also has support for your own events, using custom topics.

For more information about using Event Grid, reference Create and route custom events with Azure Event Grid.

To understand how using Event Grid promotes operational excellence for your workload, reference Diagnostic logs for Event Grid topics and Event Grid domains.

The following sections are specific to Azure Event Grid and operational excellence:

- Design considerations
- Configuration checklist
- Recommended configuration options
- Source artifacts

Design considerations

Azure Event Grid provides an uptime SLA. For more information, reference SLA for Event Grid.

Checklist

Have you configured Azure Event Grid with operational excellence in mind?

- ✓ Monitor Event Grid for failed event delivery.
- ✓ Use batched events.
- ✓ Event batches can't exceed `1MB` in size.
- ✓ Configure and optimize batch-size selection during load testing.
- ✓ Ensure Event Grid messages are accepted with `HTTP` `200-204` responses only if delivering to an endpoint that holds custom code.
- ✓ Monitor Event Grid for failed event publishing.

Configuration recommendations

Consider the following recommendations to optimize operational excellence when configuring Azure Event Grid:

Recommendation	Description
Monitor Event Grid for failed event delivery.	The `Delivery Failed` metric will increase every time a message can't be delivered to an event handler (timeout or a non-`200-204 HTTP` status code). If an event can't be lost, set up a Dead-Letter-Queue (DLQ) storage account. A DLQ account is where events that can't be delivered after the maximum retry count will be placed. Optionally, implement a notification system on the DLQ storage account, for example, by handling a *new file* event through Event Grid.
Use batched events in high-throughput scenarios.	The service will deliver a `json` array with multiple events to the subscribers, instead of an array with one event. The consuming application must be able to process these arrays.
Event batches can't exceed `1MB` in size.	If the message payload is large, only one or a few messages will fit in the batch. The consuming service will need to process more event batches. If your event has a large payload, consider storing it elsewhere, such as in blob storage, and passing a reference in the event. When integrating with third-party services through the CloudEvents schema, it's not recommended to exceed `64KB` events.
Configure and optimize batch-size selection during load testing.	Batch size selection depends on the payload size and the message volume.
Monitor Event Grid for failed event publishing.	The `Unmatched` metric will show messages that are published, but not matched to any subscription. Depending on your application architecture, the latter may be intentional.

Source artifacts

To determine the **Input Schema** type for all available Event Grid topics, use the following query:

SQL

```
Resources
| where type == 'microsoft.eventgrid/topics'
| project name, resourceGroup, location, subscriptionId,
properties['inputSchema']
```

To retrieve the **Resource ID** of existing private endpoints for Event Grid domains, use the following query:

SQL

```
Resources
| where type == 'microsoft.eventgrid/domains' and
notnull(properties['privateEndpointConnections'])
| mvexpand properties['privateEndpointConnections']
| project-rename privateEndpointConnections =
properties_privateEndpointConnections
| project name, resourceGroup, location, subscriptionId,
privateEndpointConnections['properties']['privateEndpoint']['id']
```

To identify **Public Network Access** status for all available Event Grid domains, use the following query:

SQL

```
Resources
| where type == 'microsoft.eventgrid/domains'
| project name, resourceGroup, location, subscriptionId,
properties['publicNetworkAccess']
```

To identify **Firewall Rules** for all public Event Grid domains, use the following query:

SQL

```
Resources
| where type == 'microsoft.eventgrid/domains' and
properties['publicNetworkAccess'] == 'Enabled'
| project name, resourceGroup, location, subscriptionId,
properties['inboundIpRules']
```

To identify **Firewall Rules** for all public Event Grid topics, use the following query:

SQL

```
Resources
| where type == 'microsoft.eventgrid/topics' and
properties['publicNetworkAccess'] == 'Enabled'
| project name, resourceGroup, location, subscriptionId,
properties['inboundIpRules']
```

To retrieve the **Resource ID** of existing private endpoints for Event Grid topics, use the following query:

SQL

```
Resources
| where type == 'microsoft.eventgrid/topics' and
notnull(properties['privateEndpointConnections'])
| mvexpand properties['privateEndpointConnections']
| project-rename privateEndpointConnections =
properties_privateEndpointConnections
| project name, resourceGroup, location, subscriptionId,
privateEndpointConnections['properties']['privateEndpoint']['id']
```

To determine the **Input Schema** type for all available Event Grid domains, use the following schema:

SQL

```
Resources
| where type == 'microsoft.eventgrid/domains'
| project name, resourceGroup, location, subscriptionId,
properties['inputSchema']
```

To identify **Public Network Access** status for all available Event Grid topics, use the following query:

SQL

```
Resources
| where type == 'microsoft.eventgrid/topics'
| project name, resourceGroup, location, subscriptionId,
properties['publicNetworkAccess']
```

Next step

Event Hubs and reliability

Architecture best practices for Azure Event Hubs

Article • 08/14/2024

Azure Event Hubs is a scalable event processing service that ingests and processes large volumes of events and data, with low latency and high reliability. It can receive and process millions of events per second. Data sent to an event hub can be transformed and stored by using any real-time analytics provider or batching and storage adapters.

For more information about using Event Hubs, reference the Azure Event Hubs documentation to learn how to use Event Hubs to ingest millions of events per second from connected devices and applications.

To understand ways using Event Hubs helps you achieve operational excellence and reliability for your workload, reference the following articles:

- Monitor Azure Event Hubs
- Stream Azure Diagnostics data using Event Hubs
- Scaling with Event Hubs

The following sections are specific to Azure Event Hubs from a Well Architected Framework perspective:

- Design considerations
- Configuration checklist
- Recommended configuration options
- Source artifacts

Design considerations

Azure Event Hubs provides an uptime SLA. For more information, reference SLA for Event Hubs.

Checklist

Have you configured Azure Event Hubs with operational excellence in mind?

- ✓ Create SendOnly and ListenOnly policies for the event publisher and consumer, respectively.

- ✓ When using the SDK to send events to Event Hubs, ensure the exceptions thrown by the retry policy (`EventHubsException` or `OperationCancelledException`) are properly caught.
- ✓ In high-throughput scenarios, use batched events.
- ✓ Every consumer can read events from one to maximum partitions supported by the Event Hubs SKU
- ✓ When developing new applications, use `EventProcessorClient` (.NET and Java) or `EventHubConsumerClient` (Python and JavaScript) as the client SDK.
- ✓ As part of your solution-wide availability and disaster recovery strategy, consider enabling the Event Hubs geo disaster-recovery option.
- ✓ When a solution has a large number of independent event publishers, consider using Event Publishers for fine-grained access control.
- ✓ Don't publish events to a specific partition.
- ✓ When publishing events frequently, use the AMQP protocol when possible.
- ✓ The number of partitions reflect the degree of downstream parallelism you can achieve.
- ✓ Ensure each consuming application uses a separate consumer group and only one active receiver per consumer group is in place.
- ✓ When using the Capture feature, carefully consider the configuration of the time window and file size, especially with low event volumes.

Configuration recommendations

Consider the following recommendations to optimize reliability when configuring Azure Event Hubs:

Expand table

Recommendation	Description
When using the SDK to send events to Event Hubs, ensure the exceptions thrown by the retry policy (`EventHubsException` or `OperationCancelledException`) are properly caught.	When using `HTTPS`, ensure a proper retry pattern is implemented.
In high-throughput scenarios, use batched events.	The service will deliver a `json` array with multiple events to the subscribers, instead of an array with one event. The consuming application must process these arrays.

Recommendation	Description
Every consumer can read events from one to maximum partitions supported by the Event Hubs SKU.	To achieve maximum scale on the side of the consuming application, every consumer should read from a single partition.
When developing new applications, use `EventProcessorClient` (.NET and Java) or `EventHubConsumerClient` (Python and JavaScript) as the client SDK.	`EventProcessorHost` has been deprecated.
As part of your solution-wide availability and disaster recovery strategy, consider enabling the Event Hubs geo disaster-recovery option.	This option allows the creation of a secondary namespace in a different region. Only the active namespace receives messages at any time. Messages and events aren't replicated to the secondary region. The RTO for the regional failover is *up to 30 minutes*. Confirm this RTO aligns with the requirements of the customer and fits in the broader availability strategy. If a higher RTO is required, consider implementing a client-side failover pattern.
When a solution has a large number of independent event publishers, consider using Event Publishers for fine-grained access control.	Event Publishers automatically set the partition key to the publisher name, so this feature should only be used if the events originate from all publishers evenly.
Don't publish events to a specific partition.	If ordering events is essential, implement ordering downstream or use a different messaging service instead.
When publishing events frequently, use the AMQP protocol when possible.	AMQP has higher network costs when initializing the session, but `HTTPS` requires TLS overhead for every request. AMQP has higher performance for frequent publishers.
The number of partitions reflect the degree of downstream parallelism you can achieve.	For maximum throughput, use the maximum number of partitions supported by the SKU when creating the Event Hub. Increasing the number of partitions enables you to scale concurrent processing entities to match the partitions, ensuring optimal send and receive availability.
When using the Capture feature, carefully consider the configuration of the time window and file size, especially with low event volumes.	Data Lake gen2 will charge for minimal transaction size. If you set the time window so low that the file hasn't reached minimum size, you'll incur extra cost.

Source artifacts

To find Event Hubs namespaces with **Basic** SKU, use the following query:

SQL

```
Resources
| where type == 'microsoft.eventhub/namespaces'
| where sku.name == 'Basic'
| project resourceGroup, name, sku.name
```

Next step

Service Bus and reliability

Feedback

Was this page helpful? Yes No

Azure Well-Architected Framework perspective on Azure ExpressRoute

Article • 11/04/2024

Azure ExpressRoute enables you to create private connections between Azure data-centers and infrastructure that's on your premises or in a colocation facility. ExpressRoute connections don't go over the public internet, offering more security, faster speeds, and lower latencies than typical internet connections.

This article assumes that you have working knowledge of Azure ExpressRoute and are well versed with all of its features. The guidance in this article provides architectural recommendations that are mapped to the principles of the Azure Well-Architected Framework pillars.

> **ⓘ Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern along with design strategies localized to the technology scope.
>
> Also included are *recommendations* on the technology capabilities that can help materialize those strategies. The recommendations don't represent an exhaustive list of all configurations available for Azure ExpressRoute and its dependencies. Instead, they list the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept or optimize your existing environments.
>
> Foundational architecture that demonstrates the key recommendations: **Implement Cloud Adoption Framework enterprise-scale landing zones in Azure**.

Technology scope

This review focuses on the interrelated decisions for the following Azure resources:

- Azure ExpressRoute

Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover fast from failures**.

Highly reliable, resilient, and available network connections are fundamental to a well-structured system. Reliability consists of two principles: *resiliency* and *availability*. Resiliency aims to prevent failures and, if they occur, to restore applications to a fully operational state. Availability ensures consistent access to your applications or workloads. It's important to proactively plan for reliability based on your business needs and application requirements.

The **Reliability design principles** provide a high-level design strategy applied for individual components, system flows, and the system as a whole.

Design checklist

Start your design strategy based on the design review checklist for Reliability. Determine its relevance to your business requirements while keeping in mind the features and capabilities of Azure ExpressRoute. Extend the strategy to include more approaches as needed.

- ✓ **Build redundancy, strengthen resiliency:** Eliminate single points of failure as much as practical. Plan for redundancy in the network design by configuring multiple ExpressRoute circuits, diverse paths, and multiple peering locations closest to your on-premises locations. Define availability and recovery targets for the components, the flows, and the overall solution. Visualize the targets to negotiate, gain consensus, set expectations, and drive actions to achieve the ideal state.
- ✓ **Anticipate potential failures:** Plan mitigation strategies for potential failures. The following table shows examples of failure mode analysis.

Expand table

Failure	Mitigation
Circuit disruption, such as hardware failure at a peering location.	Have a secondary circuit for redundancy while service is restored. Monitor the health of the circuit to identify when there might be issues.
Gateway disruption in the Azure virtual networks.	Have your gateway deployed in multiple availability zones, and with Active/Active configuration so that the disruption of the gateway in a single zone does not prevent connectivity.

- ✓ **Plan for site resiliency:** Planning for site resiliency is crucial to ensure high availability. ExpressRoute offers three architectures of site resiliency: *Standard*, *High*,

and *Maximum*. Standard resiliency provides basic protection against link failures, but does not provide protection against site failures. High resiliency offers enhanced protection with additional failover mechanisms, and Maximum resiliency ensures the highest level of protection with multiple redundant systems and failover mechanisms.

- ✓ **Plan for regions and availability zones:** Plan for multiple region and availability zones closest to your on-premises locations to provide resiliency and high availability.
- ✓ **Plan for ExpressRoute circuit or ExpressRoute Direct:** During the initial planning phase, you want to decide whether you want to configure an ExpressRoute circuit or an ExpressRoute Direct connection.
- ✓ **Choose the right circuit SKU:** ExpressRoute circuit SKUs provide redundancy through the use of geographic expansion. ExpressRoute have three SKUs: Local, Standard, and Premium.

Expand table

SKU	Geographic Expansion	Redundancy	Use Case
Local	Access to Azure regions within or near the same metro area.	Localized redundancy	High-speed, low-latency connections within a metro area.
Standard	Access to all Azure regions within a specific geopolitical area.	Regional redundancy	Broader regional connectivity needs.
Premium	Global connectivity to any Azure region worldwide.	Global redundancy	Extensive geographic reach for global operations.

- ✓ **Plan for Active-Active connectivity:** ExpressRoute dedicated circuits provide availability when an active-active connectivity is configured between on-premises and Azure. This configuration provides higher availability of your ExpressRoute connection.
- ✓ **Plan for geo-redundant circuits:** Configure ExpressRoute circuits in more than one peering location to ensure that there are multiple, geographically diverse paths between on-premises networks and Azure. This reduces the risk of a single point of failure causing a network outage, thereby increasing the reliability and availability of the connection.

- ✓ **Configure ExpressRoute Global Reach:** As an ExpressRoute circuit Premium SKU feature, ExpressRoute Global Reach allows you to link your on-premises networks across different geographical locations directly through the Azure backbone network. By connecting your on-premises networks to multiple Azure regions, Global Reach provides an additional layer of redundancy. If one Azure region becomes unavailable, you can quickly reroute traffic to another region without relying on the public internet, maintaining secure and reliable connectivity.
- ✓ **Configure site-to-site VPN as a backup to ExpressRoute private peering:** This configuration provides an additional layer of redundancy and ensures that your network remains operational even if the ExpressRoute connection experiences an outage.
- ✓ **Plan for Virtual Network Gateways:** When selecting and configuring your ExpressRoute Virtual Network Gateway for resiliency, consider the following best practices:
    - Select the right ExpressRoute Virtual Network Gateway SKU to reflect the correct performance and throughput for your business.
    - Configure ExpressRoute Gateways in different regions for regional redundancy & resiliency.
    - Configure zone-redundant Virtual Network Gateway within Availability Zones for zone redundancy and resiliency.
    - Deploy scalable Virtual Network Gateways that allow auto-scale based on your required throughput.
- ✓ **Plan for service providers:** Choose different service providers for each circuit to ensure diverse paths. This diversity in service providers minimizes the risk of network downtime due to a single provider's outage.
- ✓ **Conduct reliability testing:** Test the network design for resiliency to ensure that the network can withstand failures. Testing can be achieved by using Azure Connectivity Toolkit to test performance across your ExpressRoute circuit to understand bandwidth capacity and latency of your network connection. Confirm failover mechanisms are working as expected.
- ✓ **Configure monitoring for ExpressRoute circuits and ExpressRoute Virtual Network Gateway health:** Configure monitoring and alerts for ExpressRoute circuit and ExpressRoute Virtual Network Gateway health based on various metrics available.

- ✓ **Use health indicators to identify disruptions:** Configure monitoring and alerts for ExpressRoute circuit and ExpressRoute Virtual Network Gateway health based on various metrics available.

Recommendations

Expand table

Recommendation	Benefit
Anticipate and mitigate potential failures when you design and architect Azure ExpressRoute.	Anticipating failures leads to the design of a more robust and resilient network architecture that can withstand various failure scenarios.
Plan for site resiliency. For Maximum or High resiliency, plan to have multiple paths between the on-premises edge and the peering locations (provider/Microsoft edge locations). For Maximum Resiliency, configure multiple circuits to different peering locations. For High Resiliency, configure a circuit between multiple peering locations within the same metropolitan area (also referred to as ExpressRoute Metro) from the on-premises network.	By having multiple paths between the on-premises edge and the peering locations, the network can continue to operate even if one path fails. This redundancy is crucial for maintaining continuous connectivity and minimizing downtime.
Plan for multiple region and availability zones.	Availability zones are physically separate locations within a region, providing fault isolation. This means that failures in one zone don't affect the others, enhancing overall system reliability.
Plan for ExpressRoute circuit or ExpressRoute Direct. During the initial planning phase, you want to decide whether you want to configure an ExpressRoute circuit or an ExpressRoute Direct connection. You also need to identify the bandwidth requirement and the SKU type requirement for your business needs.	An ExpressRoute circuit allows a private dedicated connection into Azure with the help of a connectivity provider. ExpressRoute Direct allows you to extend on-premises network directly into the Microsoft network at a peering location.
Choose the right circuit SKU for redundancy by using geographic expansion. The Local, Standard, and Premium SKUs offer different levels of connectivity, access, and performance capabilities. Premium SKU provides the highest level of redundancy with global connectivity to any Azure region worldwide.	Choosing the right circuit SKU ensures that you have the appropriate level of redundancy and connectivity for your workloads.
Plan for Active-Active connectivity. To improve high availability, redundancy, and resiliency, we	Active-active mode mode provides higher availability of your ExpressRoute connections.

Recommendation	Benefit
recommend operating both connections of an ExpressRoute circuit in active-active mode. Additionally, configure Bi-Directional Forwarding Detection (BFD) over both private and Microsoft Peering for faster failover during a link failure.	BFD provides rapid detection of link failures, enabling quicker failover to backup paths. This minimizes downtime and ensures continuous connectivity.
Plan for geo-redundant circuits.	There are scenarios where an ExpressRoute peering location or an entire regional service might experience degradation. Geo-redundancy enhances disaster recovery and high availability by ensuring that there are multiple, geographically diverse paths between on-premises networks and Azure. This reduces the risk of a single point of failure causing a network outage, thereby increasing the reliability and availability of the connection.
With ExpressRoute Global Reach you can link ExpressRoute circuits together to make a private network between your on-premises networks. Configure ExpressRoute Global Reach on your ExpressRoute circuit Premium SKU.	ExpressRoute Global Reach provides an additional layer of redundancy by linking your on-premises networks across different geographical locations directly through the Azure backbone network. This ensures that your network remains connected and operational even if one Azure region becomes unavailable.
Choose different ExpressRoute service providers for each circuit.	Diversity in service providers minimizes the risk of network downtime due to a single provider's outage. By choosing different service providers for each circuit, you can ensure that your network remains operational even if one provider experiences an outage. This redundancy is essential for maintaining continuous connectivity and minimizing downtime.
Configure Site-to-Site VPN over Microsoft peering as a backup to ExpressRoute private peering. Site-to-site VPN provides an additional layer of redundancy and ensures that your network remains operational even if the ExpressRoute connection experiences an outage.	By configuring a site-to-site VPN as a backup to ExpressRoute private peering, you can maintain continuous connectivity and minimize downtime.
Planning for zone-redundant Virtual Network Gateways. Select the right ExpressRoute Virtual Network Gateway SKU to reflect the correct performance and throughput for your business. Consider deploying a scalable virtual network	Choosing the appropriate SKU ensures that the gateway can handle the required performance and throughput for your business needs. A scalable virtual network gateway autoscales based on required throughput, allowing the

Recommendation	Benefit
gateway that allows you to achieve 40-Gbps connectivity and will auto-scale based on your required throughput. Deploy ExpressRoute virtual network gateways that are zone-redundant for maximum resiliency and redundancy across Availability Zones.	network to adapt to changing demands. This flexibility helps maintain performance during peak usage times and prevents overloading. Additionally, deploying zone-redundant virtual network gateways ensures that the network remains operational even if one availability zone experiences an outage, enhancing overall reliability and resiliency.
Conduct reliability testing with the Azure Connectivity Toolkit to ensure that the network design is resilient and can withstand failures.	Reliability testing helps identify potential issues and weaknesses in the network design, allowing you to address them proactively. By conducting reliability testing, you can ensure that the network is robust and resilient, minimizing downtime and ensuring continuous connectivity.
Configure monitoring and alerts for ExpressRoute circuits, peering, ports, and Virtual Network Gateway resource health based on various available metrics. This helps in proactively managing and maintaining the health of your network. Use Network Insights for ExpressRoute to visualize topological maps and health dashboards, providing a clear view of your configurations and their status.	By setting up monitoring and alerts based on various metrics, you can proactively detect and address issues such as increased latency, traffic drops, or circuit downtimes before they impact your services.
Configure service health to notify you about planned and unplanned maintenance. Configuring service health notifies you about changes made to your ExpressRoute circuits.	With Service Health, you can view planned and past maintenance in the Azure portal along with configuring alerts and notifications that best suits your needs.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload.

The Security design principles provide a high-level design strategy for achieving those goals by applying approaches to the technical design of Azure ExpressRoute.

Design checklist

Start your design strategy based on the **design review checklist for Security** and identify vulnerabilities and controls to improve the security posture. Extend the strategy to include more approaches as needed.

- ✓ **Leverage Azure Security Baseline for ExpressRoute:** The Microsoft cloud security benchmark provides recommendations on how you can secure your cloud solutions on Azure.
- ✓ **Implement Azure Role-Based Access Control (RBAC):** Use Azure RBAC to configure roles to limit user accounts that can add, update, or delete peering configurations on an ExpressRoute circuit or change ExpressRoute resources.
- ✓ **Configure ExpressRoute encryption:** Encrypt data in transit over ExpressRoute circuits to ensure that data transmitted between on-premises networks and Azure virtual networks is secure and protected from unauthorized access. ExpressRoute supports the following encryption options:
  - **Media Access Control Security (MACsec)** Applied to *ExpressRoute Direct resources*, MACsec enhances ExpressRoute security by providing link-layer encryption and integrity protection, ensuring that data transmitted over ExpressRoute circuits is secure, tamper-proof, and compliant with regulatory requirements.
  - **Internet Protocol Security (IPsec)** Applied to *ExpressRoute private peering*, IPsec provides secure communication between on-premises networks and Azure virtual networks by encrypting data in transit. By configuring IPsec, you can ensure that data transmitted over ExpressRoute circuits is secure and protected from unauthorized access.
- ✓ **Configure MD5 hash on ExpressRoute circuit:** During configuration of private peering or Microsoft peering, apply an MD5 hash to secure messages between the on-premises router and the MSEE routers.
- ✓ **Configure Activity log to send logs to archive:** Activity logs are essential for auditing, compliance, incident response, operational visibility, and policy enforcement for ExpressRoute. Configure Activity log to send logs to an archive for long-term retention and analysis.

Recommendations

Explore the following table of recommendations to optimize your ExpressRoute configuration for security.

Expand table

Recommendation	Benefit
Leverage Azure Security Baseline for ExpressRoute. This security baseline applies guidance from the Microsoft Cloud Security Benchmark version 1.0 to ExpressRoute.	The content is organized by the security controls defined in the benchmark and includes related guidance specific to ExpressRoute.
Implement Azure Role-Based Access Control (RBAC) to control who can manage ExpressRoute resources such as ExpressRoute circuits and gateways.	By providing granular access management to resources, you can maintain an inventory of administrative accounts with access to ExpressRoute resources and ensure that only authorized users can perform specific actions.
Configure MACsec for ExpressRoute Direct ports.	MACsec (Media Access Control security) enhances security by encrypting data, ensuring data integrity, protecting vulnerable protocols. It secures protocols that are typically not protected on Ethernet links, such as ARP, DHCP, and LACP, thereby preventing potential security threats targeting these protocols.
Encrypt traffic using IPsec (Internet Protocol Security) for ExpressRoute private peering or configure a tunnel using private peering.	IPsec encrypts data at the network layer (Layer 3) and enhances security by providing encryption, authentication, integrity protection, and compliance. This ensures that data transmitted over ExpressRoute circuits is secure and protected from unauthorized access and tampering.
Configure MD5 hash on ExpressRoute circuit during configuration of private peering or Microsoft peering to secure messages between the on-premises route and the MSEE routers.	By generating an MD5 hash of the data before transmission and comparing it with the hash generated after reception, you can ensure that the data hasn't been tampered with during transit.
Configure activity logs and send logs an to archive. Data retention is only 90 days and required to be stored in Log Analytics, Event Hubs or a storage account for archive. For more information about Activity logs in ExpressRoute, see Monitor Azure ExpressRoute.	Activity logs provide insights into operations that were performed at the subscription level for ExpressRoute resources. With Activity logs, you can determine who and when an operation was performed at the control plane.

Cost Optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget while meeting business requirements.

The Cost Optimization design principles provide a high-level design strategy for achieving those goals and making tradeoffs as necessary in the technical design related to your web apps and the environment in which they run.

Design checklist

Start your design strategy based on the design review checklist for Cost Optimization for investments and fine tune the design so that the workload is aligned with the budget allocated for the workload. Your design should use the right Azure capabilities, monitor investments, and find opportunities to optimize over time.

- ✓ **Familiarize yourself with ExpressRoute pricing:** As part of your cost model exercise, estimate the cost of ExpressRoute. Ensure that the options are adequately sized to meet the capacity demand and deliver expected performance without wasting resources.
- ✓ **Determine circuit SKU and bandwidth required:** Base your selection of ExpressRoute circuit and virtual network gateway SKU and bandwidth on the capacity demand and performance requirements of your workload.
- ✓ **Determine the ExpressRoute virtual network gateway size:** Choose the right size for your ExpressRoute virtual network gateway based on the capacity demand and performance requirements of your workload.
- ✓ **Monitor cost and create budget alerts:** Monitor the cost of your ExpressRoute circuit and create alerts for spending anomalies and overspending risks.
- ✓ **Deprovision and delete unused ExpressRoute circuits:** Azure Advisor can detect ExpressRoute circuits that have been deployed for a significant time but have a provider status of *Not Provisioned*.

Recommendations

Explore the following table of recommendations to optimize your ExpressRoute configuration for Cost optimization.

Expand table

Recommendation	Benefit
Familiarize yourself with ExpressRoute pricing. Use the Azure Pricing Calculator to estimate the cost. ExpressRoute Direct has a monthly port fee that includes the circuit fee for Local and Standard SKU ExpressRoute circuits. For Premium SKU circuits,	Understanding ExpressRoute pricing enables better cost management, informed decision-making, avoidance

Recommendation	Benefit
there's an additional circuit fee. Outbound data transfer is charged per GB used, depending on the zone number of the peering location. The outbound data charge applies only to Standard and Premium SKUs. For more information, see plan and manage costs for Azure ExpressRoute.	of unexpected charges and maximization of value.
Determine circuit SKU and bandwidth required. The way you're charged for your ExpressRoute usage varies between the three different SKU types. With the Local SKU, you're automatically charged with an Unlimited data plan. With the Standard and Premium SKUs, you can choose between a Metered or an Unlimited data plan. All ingress data is free of charge, except when using the Global Reach add-on, which incurs additional costs for data transfer between different geographical locations. It's important to review and resize your ExpressRoute circuit.	It's important to understand which SKU types and data plan works best for your workload to best optimize cost and budget.
Determine the size of the ExpressRoute Virtual Network Gateway. ExpressRoute virtual network gateways are used to pass traffic into a virtual network over private peering. Select the appropriate gateway SKU on your on-premises to Azure workload. Understand ExpressRoute Gateway pricing based on region and type. ExpressRoute Gateways are charged at an hourly rate plus the cost of an ExpressRoute circuit. Configure scalable ExpressRoute gateways to set minimum and maximum scale units for the gateway, which auto-scales based on active bandwidth or flow count. See ExpressRoute pricing and select ExpressRoute Gateways to see rates for different gateway SKUs.	This benefits you by enabling right-sizing of resources, providing flexibility to scale, optimizing performance, and supporting proactive cost management. This approach ensures that you're using resources efficiently and cost-effectively.
Monitor costs and create budget alerts. Monitor the cost of your ExpressRoute circuit and create alerts for spending anomalies and overspending risks.	Monitoring and alerts provide you with tools to control spending, enhance financial planning, ensure accountability, and optimize resource usage.
Deprovision and delete unused ExpressRoute circuits. Azure Advisor can detect ExpressRoute circuits that have been deployed for a significant time but have a provider status of *Not Provisioned*.	ExpressRoute circuits are charged from the moment they're created. To reduce unnecessary cost, deprovision the circuit with the service provider and delete the ExpressRoute circuit from your subscription.

Operational Excellence

Operational Excellence primarily focuses on procedures for **development practices, observability, and release management**.

The Operational excellence design principles provide a high-level design strategy for achieving those goals towards the operational requirements of the workload.

Design checklist

Start your design strategy based on the design review checklist for Operational Excellence for defining processes for observability, testing, and deployment related to ExpressRoute.

- ✓ **Choose the closest peering locations:** Choose the closest peering locations to your on-premises network to reduce latency and costs.
- ✓ **Configure connection monitoring between your on-premises and Azure network:** Use Connection Monitor to monitor connectivity between your on-premises resources and Azure over the ExpressRoute private peering and Microsoft peering connection.
- ✓ **Configure dynamic routing for your Microsoft peering enabled ExpressRoute circuit:** Dynamic routing for ExpressRoute leverages BGP to provide automatic route updates, optimal path selection, scalability, and interoperability for your network.
- ✓ **Configure Service Health for receiving notification:** Configure Service Health notifications to alert you when planned and upcoming maintenance is happening to all ExpressRoute circuits in your subscription. For more information on how to integrate with the overall health model for your workload, see Health modeling for workloads.
- ✓ **Configure Traffic Collector for ExpressRoute:** ExpressRoute Traffic Collector enables the sampling of network flows over your ExpressRoute circuits.
- ✓ **Collect, analyze, and visualize metrics and logs:** Collect metrics and logs as part of the overall monitoring strategy of your solution. Set alerts to proactively notify you when a certain threshold is met. Review metrics and dashboards available through ExpressRoute Insights to view details of your peering components all in a single place.

Recommendations

Explore the following table of recommendations to optimize your ExpressRoute configuration for Operational excellence.

Expand table

Recommendation	Benefit
Choose the closest peering locations to your on-premises network to reduce latency and costs.	By choosing the closest peering location to your on-premises network, you can reduce latency and costs, ensuring optimal performance and cost-effectiveness.
Configure Connection Monitor between your on-premises and Azure network.	Connection Monitor can detect networking issues by identifying where along the network path the problem is and help you quickly resolve configuration or hardware failures. Connection Monitor is part of Azure Monitor logs.
Configure dynamic routing your Microsoft peering enabled ExpressRoute circuit.	Dynamic routing allows for more efficient and flexible routing, ensuring optimal path selection and automatic updates to routing tables in response to network changes.
Configure Service Health notifications to alert you when planned and upcoming maintenance is scheduled for all ExpressRoute circuits in your subscription. Service Health also displays past maintenance events along with Root Cause Analysis (RCA) if unplanned maintenance event occurs.	Service Health notifications provide timely alerts about planned and unplanned maintenance, outages, and early warnings about potential issues. This allows you to stay informed about the status of your ExpressRoute circuits.
Configure Traffic Collector for ExpressRoute	ExpressRoute Traffic Collector enables the sampling of network flows over your ExpressRoute circuits. It supports both Private peering and Microsoft peering, providing near real-time visibility into network throughput and performance.
Review metrics with Network Insights. ExpressRoute Insights with Network Insights allow you to review and analyze ExpressRoute circuits, gateways, connections metrics and health dashboards. ExpressRoute Insights also provide a topology view of your ExpressRoute connections where you can view details of your peering components all in a single place.	Network Insights offers a centralized platform to monitor various metrics across ExpressRoute circuits, gateways, and connections, providing a comprehensive view of network health and performance.
Review ExpressRoute resource metrics. Use Azure Monitor to collect metrics and create alerts based on your configuration.	Metrics are collected for ExpressRoute circuits, ExpressRoute gateways, ExpressRoute gateway connections, and ExpressRoute Direct. These metrics are

Recommendation	Benefit
	useful for diagnosing connectivity problems and understanding the performance of your ExpressRoute connection.
Review ExpressRoute metrics and create alerts. ExpressRoute uses Azure Monitor to collect metrics and create alerts based on your configuration. Follow the recommendations for designing and creating a monitoring system to implement your monitoring strategy for ExpressRoute and your workloads.	Metrics are collected for ExpressRoute circuits, ExpressRoute gateways, ExpressRoute gateway connections, and ExpressRoute Direct. These metrics are useful for diagnosing connectivity problems and understanding the performance of your ExpressRoute connection.

Performance Efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance. The Performance Efficiency design principles provide a high-level design strategy for achieving those capacity goals against the expected usage.

Design checklist

Start your design strategy based on the design review checklist for Performance Efficiency for defining a baseline based on key performance indicators for Azure ExpressRoute. Plan for scaling the network to meet the demands of your workloads. Failure to plan for scaling correctly may result in downtime or extra costs. Ensure proper scaling when it comes to both circuit planning and gateway planning.

- ✓ **Test ExpressRoute gateway performance to meet work load requirements:** Use the Azure Connectivity Toolkit to test performance across your ExpressRoute circuit to understand bandwidth capacity and latency of your network connection.
- ✓ **Plan for scaling:** Based on your scalability requirements, choose the right ExpressRoute circuit SKU and also the Virtual Network Gateway SKUs. Each SKU offers different features and limits. Take into consideration the performance, feature, and routing needs of your network. For additional scalability guidance for your solution, see Recommendations for optimizing scaling and partitioning.
- ✓ **Monitor the performance of ExpressRoute resources:** Collect and analyze the performance telemetry in accordance with the WAF Recommendations for

collecting performance data. Validate that it meets your performance targets and set up alerts to proactively notify you when a certain threshold is met.

Recommendations

Explore the following table of recommendations to optimize your ExpressRoute configuration for performance efficiency.

Expand table

Recommendation	Benefit
Test ExpressRoute gateway performances to meet work load requirements with the Azure Connectivity Toolkit. Schedule bandwidth-intensive operations such as backups and performance testing at times of low production traffic.	The toolkit provides user-friendly tools and interfaces that simplify the process of configuring and managing network connections to Azure. The toolkit includes tools to optimize network performance, ensuring efficient and reliable connectivity to Azure services.
Plan for scaling of ExpressRoute circuits. Upgrade your ExpressRoute circuit bandwidth to meet your production workload requirements. Circuit bandwidth is shared between all virtual networks connected to the ExpressRoute circuit. Depending on your workload, one or more virtual networks can use up all the bandwidth on the circuit. For more information, see ExpressRoute limits.	Upgrading the bandwidth ensures that the network can handle increasing data volumes and more users without compromising performance.
Plan for scaling of ExpressRoute Virtual Network Gateway. Upgrade your ExpressRoute Virtual Network Gateway SKU to meet your production workload requirements.	Upgrading to a larger gateway SKU provides higher throughput capabilities, allowing more data to be transferred between on-premises networks and Azure more quickly. A larger gateway can manage more simultaneous connections and higher volumes of traffic, reducing the likelihood of network congestion and bottlenecks.
Configure Scalable Gateways to automatically scale for performance.	Scalable Gateways allows you to scale up and down automatically with your gateway instances to accommodate performance needs. ErGwScale SKU also enables you to achieve 40-Gbps connectivity to virtual machines and Private Endpoints within the virtual network.

Recommendation	Benefit
Enable ExpressRoute FastPath for higher throughput on your virtual network gateway.	This feature improves the data path performance between your on-premises network and your virtual network resources by bypassing the gateway. As business needs grow, FastPath provides the necessary bandwidth and performance to support increasing data volumes and more users without compromising performance. Enabling FastPath ensures that the network can handle future expansions and new applications, providing long-term performance efficiency.
Monitor Monitor ExpressRoute circuit, port, and gateway metrics. Configure alerts for ExpressRoute metrics to proactively notify you when a certain threshold is met. ExpressRoute circuit metrics supports metrics such as Arp Availability, BitsInPerSecond, DroppedInBitsPerSecond. ExpressRoute port metrics supports metrics such as AdminState, BitsInPerSecond, and FastPathRoutesCount. ExpressRoute Gateway metrics supports metrics such as Bits In Per Second, Active Flows, and Count Of Routes Advertised to Peer. Monitor performance targets with Connection Monitor.	ExpressRoute circuit, port, and gateway metrics are useful to understand anomalies that can happen with your ExpressRoute connection such as outages and maintenance happening to your ExpressRoute circuits. Connection Monitor can detect networking issues by identifying where along the network path the problem is and help you quickly resolve configuration or hardware failures.

Azure Policy

Azure Policy doesn't provide any built-in policies for ExpressRoute, but custom policies can be created to help govern how ExpressRoute circuits should match your desired end state, such as SKU choice, peering type, peering configurations and so on.

Next steps

Consider the following articles as resources that demonstrate the recommendations highlighted in this article.

- We recommend starting with the Cloud Adoption Framework Ready methodology's guidance
  - Connect to Azure and Architect for hybrid connectivity with Azure ExpressRoute.
  - Traditional Azure network topology
  - Virtual WAN network topology (Microsoft-managed)

- Use the following product documentation to build your implementation expertise:
  - Configure an ExpressRoute circuit or ExpressRoute Direct port to establish communication between your on-premises network and Azure.
  - Azure ExpressRoute product documentation.
  - Architect ExpressRoute for Resiliency to ensure your setup is robust and reliable.
  - For low-code application architectures, review the guidance on Enabling ExpressRoute for Power Platform when planning and configuring ExpressRoute for use with Microsoft Power Platform.

Feedback

Was this page helpful? Yes No

Architecture best practices for Azure Functions

Article • 02/13/2025

Azure Functions is a serverless compute service that you can use to run event-driven code without having to manage the infrastructure directly. Azure Functions is a function as a service offering that abstracts the underlying infrastructure. This abstraction allows you to focus on your code. Azure Functions automatically scales based on demand and charges you only for the resources consumed during operation.

This article assumes that as an architect, you've reviewed the compute decision tree and chose Azure Functions as the compute solution for your workload. The guidance in this article provides architectural recommendations that are mapped to the principles of the Well-Architected Framework pillars.

> **ⓘ Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern along with design strategies localized to the technology scope.
>
> Also included are recommendations for the technology capabilities that can help materialize those strategies. The recommendations don't represent an exhaustive list of all configurations that are available for Azure Functions. Instead, they list the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept or to optimize your existing environments.
>
> Foundational architecture that demonstrates the key recommendations: **Web applications architecture design**.

Technology scope

This review focuses on the interrelated decisions for the following Azure resource:

- Function apps

> **ⓘ Note**

This service guide builds on guidance in the Azure App Service (Web Apps) service guide. Azure Functions is a serverless compute service. Functions are deployed on App Service plans, which provide the underlying compute infrastructure for your workload. You can choose from several hosting plans.

The Consumption plan is fully managed and doesn't support access to the associated App Service instance. The Dedicated and Premium hosting plans give you the ability to configure and manage the App Service instance. Refer to the App Service service guide for recommendations and considerations when you choose the Dedicated or Premium hosting plan.

Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover fast from failures**.

The Reliability design principles provide a high-level design strategy applied for individual components, system flows, and the system as a whole.

Design checklist

Start your design strategy based on the design review checklist for Reliability. Determine its relevance to your business requirements while keeping in mind the specific features and configurations of Azure Functions. Extend the strategy to include more approaches as needed.

- ✓ **Understand the triggers:** Ensure that you fully understand how Azure Functions is triggered. Triggers include HTTP requests, timers, queues, and more. Choose triggers that match the reliability requirements of your application.

  For example, if your function processes messages from a queue, ensure that the queue itself is resilient and that messages can be reprocessed if a failure occurs.

- ✓ **Implement retries and durable patterns:** Use built-in retry policies for transient failures. For more complex workflows, consider using the Durable Functions extension of Azure Functions. It provides state management and coordination across multiple function operations.

  Durable Functions is particularly useful for scenarios that require reliability across long-running workflows. It supports automatic retries and durable task management.

- ✓ **Ensure proper exception handling:** Implement robust exception handling to help ensure that functions fail gracefully. Log errors and consider implementing alerting mechanisms for critical failures.
- ✓ **Plan for scalability:** Azure Functions automatically scales out based on demand. Ensure that your application is designed to handle spikes by testing its performance under load.

  Consider using Application Insights to monitor performance and track how your functions scale in real time.
- ✓ **Design for idempotency:** Ensure that your functions can be safely retried without causing unintended side effects. Idempotency is crucial for functions that interact with external systems or modify data.
- ✓ **Use Durable Functions for long-running operations:** For operations that require orchestration or long-running processes, use Durable Functions to maintain state and help ensure reliability across multiple steps.
- ✓ **Monitor the health of your solution:** Integrate your Azure Functions solution into your overall workload health monitoring and alerting system.

Recommendations

Expand table

Recommendation	Benefit
Configure **automatic retries** for transient errors. Learn more about Azure Functions error handling and retries.	Improves reliability by automatically retrying failed runs, which reduces the likelihood of data loss or interruption.
Use **Durable Functions** for orchestrating complex workflows and long-running processes. Learn more about Durable Functions.	Provides reliable operation of long-running workflows, with built-in state management and automatic retries.
Implement **monitoring** and **centralized logging** by using Application Insights. Set up Application Insights.	Enhances monitoring and troubleshooting by providing detailed insights into function operations and dependencies.
Scale out automatically based on the Functions hosting plan, trigger type, and demand. Learn more about scaling.	Helps ensure that your application can handle increases in traffic without manual intervention, which improves reliability and performance.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload.

The Security design principles provide a high-level design strategy for achieving those goals by applying approaches to the technical design around Azure Functions.

Design checklist

Start your design strategy based on the design review checklist for Security and identify vulnerabilities and controls to improve the security posture. Extend the strategy to include more approaches as needed.

- ✓ **Use managed identities:** Enable managed identities for your function apps to securely access other Azure services without needing to manage credentials.
- ✓ **Secure your function app with Microsoft Entra ID:** Restrict access to your functions by configuring Azure Functions to require Microsoft Entra ID authentication.
- ✓ **Apply network security controls:** Understand the available Azure Functions networking options for your hosting model to help secure the Azure Functions networking configuration. Secure the networking to meet your requirements.

Recommendations

Expand table

Recommendation	Benefit
Enable managed identities for secure access to Azure resources. Enable managed identities	Simplifies credential management by eliminating the need to store and rotate secrets. This approach enhances security.
Use **Azure Key Vault** for secrets management and regular rotation. Integrate Key Vault with Azure Functions.	Helps protect sensitive information, such as API keys and connection strings, by storing it securely and automating secret rotation when it isn't feasible to use a managed identity.
Integrate with a virtual network and use **private endpoints**.	Secures function apps by restricting access to your internal network and preventing exposure to the public internet. Virtual network integration and private endpoints aren't available in the Consumption hosting plan.

Cost Optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget while meeting business requirements.

The **Cost Optimization design principles** provide a high-level design strategy for achieving those goals and making tradeoffs as necessary in the technical design related to Azure Functions and its environment.

Design checklist

Start your design strategy based on the design review checklist for Cost Optimization for investments. Fine-tune the design so that the workload is aligned with the budget that's allocated for the workload. Your design should use the right Azure capabilities, monitor investments, and find opportunities to optimize over time.

- ✓ **Choose the right pricing plan:** Azure Functions has multiple pricing plans, including the Consumption plan, Premium plan, and Dedicated (App Service) plan. Choose a pricing plan that aligns with your workload and cost considerations.

  The Consumption plan is ideal for unpredictable workloads that have infrequent operations because you only pay for the operation time and resources consumed. When your app is idle, you aren't charged.

- ✓ **Optimize operation time:** Reduce operation time by optimizing function code. Minimize the use of external dependencies and optimize code logic to decrease the duration of each operation.

- ✓ **Monitor and analyze costs:** Regularly monitor function app usage and costs by using Microsoft Cost Management and set up alerts to detect cost anomalies.

Recommendations

Expand table

Recommendation	Benefit
Use the **Consumption plan** for workloads with unpredictable traffic. Understand the Consumption plan.	Reduces costs by charging only for the resources used when functions are performed, which avoids costs associated with idle resources.

Recommendation	Benefit
Reserve capacity for **Elastic Premium plans** or **Dedicated App Service plans** if your workload is predictable.	Lowers costs through discounted pricing for predictable workloads with steady usage patterns.
Regularly **monitor costs** and set up alerts for anomalies. Monitor costs with alerts.	Helps identify cost spikes early, which enables proactive management and optimization.

Operational Excellence

Operational Excellence primarily focuses on procedures for **development practices, observability, and release management**.

The **Operational Excellence design principles** provide a high-level design strategy for achieving those goals for the operational requirements of the workload.

Design checklist

Start your design strategy based on the design review checklist for Operational Excellence for defining processes for observability, testing, and deployment related to Azure Functions.

- ✓ **Automate deployments:** Use continuous integration and continuous delivery (CI/CD) pipelines to automate the deployment of your function apps. Integrate with Azure DevOps or GitHub Actions for seamless deployment workflows.
- ✓ **Implement health monitoring:** Use Azure Monitor and Application Insights to track the health and performance of your functions. Set up alerts for critical metrics and use custom dashboards for real-time insights. For more information, see Monitor Azure Functions.
- ✓ **Deploy your solution safely:** Understand the deployment models available for Azure Functions and adopt the model that best meets your safe deployment practices.
- ✓ **Plan for disaster recovery (DR):** Implement DR strategies by using cross-region DR and availability zones for critical functions.

Recommendations

Expand table

Recommendation	Benefit
Automate deployments with **CI/CD pipelines** by using Azure DevOps or GitHub Actions. Set up CI/CD.	Improves deployment consistency, reduces manual errors, and accelerates time-to-market for new features.
Use **deployment slots** for staging changes before the production release.	Reduces the risk of introducing errors into production and enables safe rollbacks if problems are detected.
Implement **centralized monitoring** using Application Insights and Azure Monitor by using available metrics.	Enhances visibility into function performance, which helps identify and resolve problems quickly.

Performance Efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance.

The **Performance Efficiency design principles** provide a high-level design strategy for achieving those capacity goals against the expected usage.

Design checklist

Start your design strategy based on the design review checklist for Performance Efficiency. Define a baseline that's based on key performance indicators for Azure Functions.

- ✓ **Optimize cold starts:** Minimize the effects of cold starts by using Premium Flex plans that provide prewarmed instances or ensure that your functions remain warm by using strategies such as the Azure Functions warmup trigger.
- ✓ **Optimize function code:** Write efficient code to reduce operation time and resource consumption. Avoid long-running operations and optimize external service calls.
- ✓ **Enable autoscaling:** Leverage the autoscaling capabilities of Azure Functions to automatically scale out based on demand. Ensure that your scaling rules are well-defined and tested.
- ✓ **Monitor performance metrics:** Use Application Insights to monitor key performance metrics, such as operation time, CPU, and memory usage. Set up alerts for performance degradation.

Recommendations

Expand table

Recommendation	Benefit
Use the **Elastic Premium plan with a prewarmed instance** to minimize cold-start latency. Understand cold starts.	Reduces the latency associated with cold starts, which improves response times for time-sensitive applications.
Optimize function code to reduce operation time. Understand the best practices for Azure Functions.	Enhances performance by reducing the time and resources required for each function to run.
Enable autoscaling to automatically adjust capacity based on demand. Configure autoscaling.	Ensures that your function apps can handle varying loads without manual intervention. This approach helps maintain performance under pressure.

Azure policies

Azure provides an extensive set of built-in policies related to Azure Functions and its dependencies. A set of Azure policies can audit some of the preceding recommendations. For example, you can check whether:

- All function apps have managed identities enabled.
- Function apps are using only private endpoints for network security.
- Diagnostic logging is enabled for all function apps.

For comprehensive governance, review the Azure Policy built-in definitions and other policies that might affect the security of the network layer.

Azure Advisor recommendations

Azure Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments.

For more information, see Azure Advisor.

Next steps

Consider the following resources to further explore the recommendations highlighted in this document:

- Reference architectures:
  - Serverless web application architecture
  - Serverless event processing

Feedback

Was this page helpful? Yes No

Azure Well-Architected Framework perspective on Log Analytics

Article • 03/01/2024

Well-Architected Framework workload functionality and performance must be monitored in diverse ways and for diverse reasons. Azure Monitor Log Analytics workspaces are the primary log and metric sink for a large portion of the monitoring data. Workspaces support multiple features in Azure Monitor including ad-hoc queries, visualizations, and alerts. For general monitoring principles, see Monitoring and diagnostics guidance. The guidance presents general monitoring principles. It identifies the different types of data. It identifies the required analysis that Azure Monitor supports and it also identifies the data stored in the workspace that enables the analysis.

This article assumes that you understand system design principles. You also need a working knowledge of Log Analytics workspaces and features in Azure Monitor that populate operational workload data. For more information, see Log Analytics workspace overview.

> **ⓘ Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern along with design strategies localized to the technology scope.
>
> Also included are *recommendations* on the technology capabilities or deployment topologies that can help materialize those strategies. The recommendations don't represent an exhaustive list of all configurations available for Log Analytics workspaces and its related Azure Monitor resources. Instead, they list the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept, design your workload monitoring environment, or optimize your existing workload monitoring solution.

Technology scope

This guide focuses on the interrelated decisions for the following Azure resources.

- Log Analytics workspaces
- Workload operational log data

- Diagnostic settings on Azure resources in your workload

Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover fast from failures**.

The **Reliability design principles** provide a high-level design strategy applied for individual components, system flows, and the system as a whole.

The reliability situations to consider for Log Analytics workspaces are:

- Availability of the workspace.
- Protection of collected data in the rare case of an Azure datacenter or region failure.

There's currently no standard feature for failover between workspaces in different regions, but there are strategies to use if you have particular requirements for availability or compliance.

Design checklist for Reliability

Start your design strategy based on the **design review checklist for Reliability** and determine its relevance to your business requirements while keeping in mind the SKUs and features of virtual machines (VMs) and their dependencies. Extend the strategy to include more approaches as needed.

- ✓ **Review service limits for Log Analytics workspaces**. The service limits section helps you understand restrictions on data collection and retention, and other aspects of the service. These limits help you determine how to properly design your workload observability strategy. Be sure to review Azure Monitor service limits since many of the functions discussed therein, like queries, work hand-in-hand with Log Analytics workspaces.
- ✓ **Plan for workspace resilience and recovery.** Log Analytics workspaces are regional, with no built-in support for cross-regional redundancy or replication. Also, availability zone redundancy options are limited. As such, you should determine the reliability requirements of your workspaces and strategize to meet those targets. Your requirements might stipulate that your workspace must be resilient to datacenter failures or regional failures, or they might stipulate that you must be able to recover your data to a new workspace in a failover region. Each of these scenarios require additional resources and processes to be put in place to be

successful, so balancing your reliability targets with cost and complexity should be carefully considered.

- ✓ **Choose the right deployment regions to meet your reliability requirements.** Deploy your Log Analytics workspace and data collection endpoints (DCEs) co-located with the workload components emitting operational data. Your choice of the appropriate region in which to deploy your workspace and your DCEs should be informed by where you deploy your workload. You might need to weigh the regional availability of certain Log Analytics functionality, like dedicated clusters, against other factors more central to your workload's reliability, cost, and performance requirements.
- ✓ **Ensure that your observability systems are healthy.** Like any other component of your workload, ensure that your monitoring and logging systems are functioning properly. To accomplish this, enable features that send health data signals to your operations teams. Set up health data signals specific to your Log Analytics workspaces and associated resources.

Configuration recommendations for Reliability

Expand table

Recommendation	Benefit
Don't include your Log Analytics workspaces in your workload's critical path. Your workspaces are important to a functioning observability system, but the functionality of your workload shouldn't depend on them.	Keeping your workspaces and associated functions out of your workload's critical path minimizes the risk of issues affecting your observability system from affecting the runtime execution of your workload.
To support high durability of workspace data, deploy Log Analytics workspaces into a region that supports data resilience. Data resilience is only possible through linking of the workspace to a dedicated cluster in the same region.	When you use a dedicated cluster, it lets you spread the associated workspaces across availability zones, which offer protection against datacenter outages. If you don't collect enough data now to justify a dedicated cluster, this preemptive regional choice supports future growth.
Choose your workspace deployment based on proximity to your workload. Use data collection endpoints (DCE) in the same region as the Log Analytics workspace.	Deploy your workspace in the same region as the instances of your workload. Having your workspace and DCEs in the same region as your workload mitigates the risk of impacts by outages in other regions. DCEs are used by the Azure Monitor agent and the Logs Ingestion API to send workload operational data to a Log

Recommendation	Benefit
	Analytics workspace. You might need multiple DCEs even though your deployment only has a single workspace. For more information on how to configure DCEs for your particular environment, see How to set up data collection endpoints based on your deployment. If your workload is deployed in an active-active design, consider using multiple workspaces and DCEs spread across the regions in which your workload is deployed. Deploying workspaces in multiple regions adds complexity to your environment. Balance the criteria detailed in Design a Log Analytics workspace architecture with your availability requirements.
If you require the **workspace to be available** in a region failure, or you don't collect enough data for a dedicated cluster, configure data collection to send critical data to multiple workspaces in different regions. This practice is also known as log multicasting. For example, configure DCRs for multiple workspaces for Azure Monitor agent running on VMs. Configure multiple diagnostic settings to collect resource logs from Azure resources and send the logs to multiple workspaces.	
In this way, workload operational data is available in the alternate workspace if there's a regional failure. But know that resources that rely on the data such as alerts and workbooks wouldn't automatically be replicated to the other regions. Consider storing Azure Resource Manager (ARM) templates for critical alerting resources with configuration for the alternate workspace or deploying them in all regions but disabling them to prevent redundant alerts. Both options support quick enablement in a regional failure. Tradeoff: This configuration results in duplicate ingestion and retention charges so only use it for critical data.	

Recommendation	Benefit
If you require **data to be protected** in a datacenter or region failure, configure data export from the workspace to save data in an alternate location. This option is similar to the previous option of multicasting the data to different workspaces. But this option costs less because the extra data is written to storage. Use Azure Storage redundancy options, including geo-redundant storage (GRS) and geo-zone-redundant storage (GZRS), to further replicate this data to other regions. Data export doesn't provide resiliency against incidents impacting the regional ingestion pipeline.	While the historic operational log data might not be readily queryable in the exported state, it ensures the data survives a prolonged regional outage and can be accessed and retained for extended period. If you require the export of tables not supported by data export, you can use other methods of exporting data, including Logic Apps, to protect your data. For this strategy to work as a viable recovery plan, you must have processes in place to reconfigure diagnostic settings for your resources in Azure and on all agents that provide data. You must also plan to manually rehydrate your exported data into a new workspace. As with the previously described option, you also need to define processes for those resources that rely on the data like alerts and workbooks.
For mission-critical workloads requiring high availability, consider implementing a federated workspace model that uses multiple workspaces to provide high availability if there's a regional failure.	Mission-critical provides prescriptive best practice guidance for designing highly reliable applications on Azure. The design methodology includes a federated workspace model with multiple Log Analytics workspaces to deliver high availability if there are multiple failures, including the failure of an Azure region. This strategy eliminates egress costs across regions and remains operational with a region failure. But it requires more complexity that you must manage with configuration and processes described in Health modeling and observability of mission-critical workloads on Azure.
Use infrastructure as code (IaC) to deploy and manage your workspaces and associated functions.	When you automate as much of your deployment and your mechanisms for resilience and recovery as practical, it ensures that these operations are reliable. You save critical time in your operations processes and minimize the risk of human error.

Recommendation	Benefit
	Ensure that functions like saved log queries are also defined through your IaC to recover them to a new region if recovery is required.
Design DCRs with a single responsibility principle to keep DCR rules simple. While one DCR could be loaded with all the input, rules, and destinations for the source systems, it's preferable to design narrowly focused rules that rely on fewer data sources. Use composition of rule assignments to arrive at the desired observability scope for the logical target. Also, minimize transformation in DCRs	When you use narrowly focused DCRs, it minimizes the risk of a rule misconfiguration having a broader effect. It limits the effect to only the scope for which the DCR was built. For more information, see Best practices for data collection rule creation and management in Azure Monitor. While transformation can be powerful and necessary in some situations, it can be challenging to test and troubleshoot the keyword query language (KQL) work being done. When possible, minimize the risk of data loss by ingesting the data raw and handling transformations downstream at query time.
When setting a daily cap or a retention policy, be sure you're maintaining your reliability requirements by ingesting and retaining the logs that you need.	A daily cap stops the collection of data for a workspace once a specified amount is reached, which helps you maintain control over your ingestion volume. But only use this feature after careful planning. Ensure that your daily cap isn't being hit with regularity. If that happens, your cap is set too restrictively. You need to reconfigure the daily cap so you don't miss critical signals coming from your workload. Likewise, be sure to carefully and thoughtfully approach the lowering of your data retention policy to ensure that you don't inadvertently lose critical data.
Use Log Analytics workspace insights to track ingestion volume, ingested data versus your data cap, unresponsive log sources, and failed queries among other data. Create health status alerts to proactively notify you if a workspace becomes unavailable because of a datacenter or regional failure.	This strategy ensures that you're able to successfully monitor the health of your workspaces and proactively act if the health is at risk of degrading. Like any other component of your workload, it's critical that you're aware of health metrics and can identify trends to improve your reliability over time.

Azure Policy

Azure offers no policies related to reliability of Log Analytics workspaces. You can create custom policies to build compliance guardrails around your workspace deployments, such as ensuring workspaces are associated to a dedicated cluster.

While not directly related to the reliability of Log Analytics workspaces, there are Azure policies for nearly every service available. The policies ensure that diagnostics settings are enabled for that service and validate that the service's log data is flowing into a Log Analytics workspace. All services in workload architecture should be sending their log data to a Log Analytics workspace for their own reliability needs, and the policies can help enforce it. Likewise, policies exist to ensure agent-based platforms, such as VMs and Kubernetes, have the agent installed.

Azure Advisor

Azure offers no Azure Advisor recommendations related to the reliability of Log Analytics workspaces.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload.

The **Security design principles** provide a high-level design strategy for achieving these goals by applying approaches to the technical design around your monitoring and logging solution.

Design checklist for Security

Start your design strategy based on the **design review checklist for Security** and identify vulnerabilities and controls to improve the security posture. Extend the strategy to include more approaches as needed.

- ✓ **Review the Azure Monitor security baseline** and Manage access to Log Analytics workspaces topics. These topics provide guidance on security best practices.
- ✓ **Deploy your workspaces with segmentation as a cornerstone principle.** Implement segmentation at the networking, data, and access levels. Segmentation helps ensure that your workspaces are isolated to the appropriate degree and are better protected from unauthorized access to the highest degree possible, while

still meeting your business requirements for reliability, cost optimization, operational excellence, and performance efficiency.

- ✓ **Ensure that you can audit workspace reads and writes activities and associated identities.** Attackers can benefit from viewing operational logs. A compromised identity can lead to log injection attacks. Enable auditing of operations run from the Azure Portal or through API interactions and the associated users. If you're not set up to audit your workspace, you might be putting your organization at risk of being in breach of compliance requirements.
- ✓ **Implement robust network controls.** Helps secure your network access to your workspace and your logs through network isolation and firewall functions. Insufficiently configured network controls might put you at risk of being accessed by unauthorized or malicious actors.
- ✓ **Determine what types of data need immutability or long-term retention.** Your log data should be treated with the same rigor as workload data inside production systems. Include log data in your data classification practices to ensure that you're successfully storing sensitive log data according to its compliance requirements.
- ✓ **Protect log data at rest through encryption.** Segmentation alone won't completely protect confidentiality of your log data. If unauthorized raw access happens, having the log data encrypted at rest helps prevent bad actors from using that data outside of your workspace.
- ✓ **Protect sensitive log data through obfuscation.** Just like workload data residing in production systems, you must take extra measures to ensure confidentiality is retained for sensitive information that might be intentionally or unintentionally present in operational logs. When you use obfuscation methods, it helps you hide sensitive log data from unauthorized eyes.

Configuration recommendations for Security

Expand table

Recommendation	Benefit
Use customer managed keys if you require your own encryption key to protect data and saved queries in your workspaces. Azure Monitor ensures that all data and saved queries are encrypted at rest using Microsoft-managed keys (MMK). If you require your own encryption key and collect enough data for a dedicated cluster, use customer-managed key. You can encrypt data by using your own key in Azure Key Vault, for control over the key lifecycle, and	This strategy lets you encrypt data by using your own key in Azure Key Vault, for control over the key lifecycle, and ability to revoke access to your data.

Recommendation	Benefit
ability to revoke access to your data. If you use Microsoft Sentinel, make sure that you're familiar with the considerations at Set up Microsoft Sentinel customer-managed key.	
Configure Log query auditing to track which users are running queries. Configure the audit logs for each workspace to be sent to the local workspace or consolidate in a dedicated security workspace if you separate your operational and security data. Use Log Analytics workspace insights to periodically review this data. Consider creating log query alert rules to proactively notify you if unauthorized users are attempting to run queries.	Log query auditing records the details for each query run in a workspace. Treat this audit data as security data and secure the LAQueryLogs table appropriately. This strategy bolsters your security posture by helping to ensure that unauthorized access is caught immediately if it ever happens.
Help secure your workspace through private networking and segmentation measures. Use private link functionality to limit communications between log sources and your workspaces to private networking.	When you use private link, it also lets you control which virtual networks can access a given workspace, further bolstering your security through segmentation.
Use Microsoft Entra ID instead of API keys for workspace API access where available.	API key-based access to the query APIs doesn't leave a per-client audit trail. Use sufficiently scoped Entra ID-based access so that you can properly audit programmatic access.
Configure access for different types of data in the workspace required for different roles in your organization. Set the access control mode for the workspace to *Use resource or workspace permissions*. This access control lets resource owners use resource-context to access their data without being granted explicit access to the workspace. Use table level RBAC for users who require access to a set of tables across multiple resources.	This setting simplifies your workspace configuration and helps to ensure users can't access operational data they shouldn't. Assign the appropriate built-in role to grant workspace permissions to administrators at either the subscription, resource group, or workspace level depending on their scope of responsibilities. Users with table permissions have access to all the data in the table regardless of their resource permissions. See Manage access to Log Analytics

Recommendation	Benefit
	workspaces for details on the different options for granting access to data in the workspace.
Export logs that require long-term retention or immutability. Use data export to send data to an Azure Storage account with immutability policies to help protect against data tampering. Not every type of log has the same relevance for compliance, auditing, or security, so determine the specific data types that should be exported.	You might collect audit data in your workspace that's subject to regulations requiring its long-term retention. Data in a Log Analytics workspace can't be altered, but it can be purged. Exporting a copy of the operational data for retention purposes lets you build a solution that meets your compliance requirements.
Determine a strategy to filter or obfuscate sensitive data in your workspace. You might be collecting data that includes sensitive information. Filter records that shouldn't be collected by using the configuration for the particular data source. Use a transformation if only particular columns in the data should be removed or obfuscated. If you have standards that require the original data to be unmodified, you can use the 'h' literal in KQL queries to obfuscate query results displayed in workbooks.	Obfuscating or filtering out sensitive data in your workspace helps ensure you maintain confidentiality on sensitive information. In many cases, compliance requirements dictate the ways that you can handle sensitive information. This strategy helps you comply with the requirements proactively.

Azure Policy

Azure offers policies related to the security of Log Analytics workspaces to help enforce your desired security posture. Examples of such policies are:

- Azure Monitor Logs clusters should be encrypted with customer-managed key
- Saved-queries in Azure Monitor should be saved in customer storage account for logs encryption
- Log Analytics Workspaces should block non-Azure Active Directory based ingestion

Azure also offers numerous policies to help enforce private link configuration, such as Log Analytics workspaces should block log ingestion and querying from public networks or even configuring the solution through DINE policies such as Configure Azure Monitor Private Link Scope to use private DNS zones.

Azure Advisor

Azure offers no Azure Advisor recommendations related to the security of Log Analytics workspaces.

Cost Optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget while meeting business requirements.

The Cost Optimization design principles provide a high-level design strategy for achieving those business goals. They also help you make tradeoffs as necessary in the technical design related to your monitoring and logging solution.

For more information on how data charges are calculated for your Log Analytics workspaces, see Azure Monitor Logs cost calculations and options.

Design checklist for Cost Optimization

Start your design strategy based on the **design review checklist for Cost Optimization** for investments and fine tune the design so that the workload is aligned with the budget allocated for the workload. Your design should use the right Azure capabilities, monitor investments, and find opportunities to optimize over time.

- ✓ **Perform cost modeling exercises**. These exercizes help you understand your current workspace costs and forecast your costs relative to workspace growth. Analyze your growth trends in your workload and ensure that you understand plans for workload expansion to properly forecast your future operational logging costs.
- ✓ **Choose the right billing model.** Use your cost model to determine the best billing model for your scenario. How you use your workspaces currently, and how you plan to you use them as your workload evolves determines whether a pay-as-you-go or a commitment tier model is the best fit for your scenario.

  Remember that you can choose different billing models for each workspace, and you can combine workspace costs in certain cases, so you can be granular in your analysis and decision-making.
- ✓ **Collect just the right amount of log data.** Perform regularly scheduled analysis of your diagnostic settings on your resources, data collection rule configuration, and custom application code logging to ensure that you aren't collecting unnecessary log data.

- ✓ **Treat nonproduction environments differently than production.** Review your nonproduction environments to ensure that you have configured your diagnostic settings and retention policies appropriately. These can often be significantly less robust than production, especially for dev/test or sandbox environments.

Configuration recommendations for Cost Optimization

Expand table

Recommendation	Benefit
Configure the pricing tier for the amount of data that each Log Analytics workspace typically collects.	By default, Log Analytics workspaces uses pay-as-you-go pricing with no minimum data volume. If you collect enough data, you can significantly decrease your cost by using a commitment tier, which lets you commit to a daily minimum of data collected in exchange for a lower rate. If you collect enough data across workspaces in a single region, you can link them to a dedicated cluster and combine their collected volume by using cluster pricing. For more information on commitment tiers and guidance on determining what's most appropriate for your level of usage, see Azure Monitor Logs cost calculations and options. To view estimated costs for your usage at different pricing tiers, see Usage and estimated costs.
Configure data retention and archiving.	There's a charge for retaining data in a Log Analytics workspace beyond the default of 31 days. It's 90 days if Microsoft Sentinel is enabled on the workspace and 90 days for Application Insights data. Consider your particular requirements for having data readily available for log queries. You can significantly reduce your cost by configuring archived logs. Archived logs let you retain data for up to seven years and still access it occasionally. You access the data by using search jobs or restoring a set of data to the workspace.
If you use Microsoft Sentinel to analyze security logs, consider employing a separate workspace to store those logs.	When you use a dedicated workspace for log data that your SIEM uses, it can help you control costs. The workspaces that Microsoft Sentinel uses are subject to Microsoft Sentinel pricing. Your security requirements dictate the types of logs that are required to be included in your SIEM solution. You might be able to exclude operational logs, which would be charged at the

Recommendation	Benefit
	standard Log Analytics pricing if they're in a separate workspace.
Configure tables used for debugging, troubleshooting, and auditing as Basic Logs.	Tables in a Log Analytics workspace configured for Basic Logs have a lower ingestion cost in exchange for limited features and a charge for log queries. If you query these tables infrequently and don't use them for alerting, this query cost can be more than offset by the reduced ingestion cost.
Limit data collection from data sources for the workspace.	The primary factor for the cost of Azure Monitor is the amount of data that you collect in your Log Analytics workspace. Be sure that you collect no more data than you require to assess the health and performance of your services and applications. For each resource, select the right categories for the diagnostic settings you configure to provide the amount of operational data you need. It helps you successfully manage your workload, and not manage ignored data. There might be a tradeoff between cost and your monitoring requirements. For example, you might be able to detect a performance issue more quickly with a high sample rate, but you might want a lower sample rate to save costs. Most environments have multiple data sources with different types of collection, so you need to balance your particular requirements with your cost targets for each. See Cost optimization in Azure Monitor for recommendations on configuring collection for different data sources.
Regularly analyze workspace usage data to identify trends and anomalies. Use Log Analytics workspace insights to periodically review the amount of data collected in your workspace. Further analyze data collection by using methods in Analyze usage in Log Analytics workspace to determine if there's other configurations that can decrease your usage further.	By helping you understand the amount of data collected by different sources, it identifies anomalies and upward trends in data collection that could result in excess cost. This consideration is important when you add a new set of data sources to your workload. For example, if you add a new set of VMs, enable new Azure diagnostics settings on a service, or change log levels in your application.
Create an alert when data collection is high.	To avoid unexpected bills, you should be proactively notified anytime you experience excessive usage. Notification lets you address any potential anomalies before the end of your billing period.

Recommendation	Benefit
Consider a daily cap as a preventative measure to ensure that you don't exceed a particular budget.	A daily cap disables data collection in a Log Analytics workspace for the rest of the day after your configured limit is reached. Don't use this practice as a method to reduce costs as described in When to use a daily cap, but instead to prevent runaway ingestion due to misconfiguration or abuse. If you set a daily cap, create an alert when the cap is reached. Be sure to also create an alert rule when some percentage is reached. For example, you can set an alert rule for when 90 percent capacity is reached. This alert gives you an opportunity to investigate and address the cause of the increased data before the cap shuts off critical data collection from your workload.

Azure Policy

Azure offers no policies related to cost optimization of Log Analytics workspaces. You can create custom policies to build compliance guardrails around your workspace deployments, such as ensuring that your workspaces contain the right retention settings.

Azure Advisor

Azure Advisor makes recommendations to move specific tables in a workspace to the low-cost Basic Log data plan for tables that receive relatively high ingestion volume. Understand the limitations by using basic logs before switching. For more information, see When should I use Basic Logs?. Azure Advisor might also recommend changing pricing commitment tier for the whole workspace based on overall usage volume.

Operational Excellence

Operational Excellence primarily focuses on procedures for **development practices, observability, and release management**.

The Operational Excellence design principles provide a high-level design strategy for achieving those goals towards the operational requirements of the workload.

Design checklist for Operational Excellence

Start your design strategy based on the **design review checklist for Operational Excellence** for defining processes for observability, testing, and deployment related to Log Analytics workspaces.

- ✓ **Use infrastructure as code (IaC) for all functions related to your workload's Log Analytics workspaces.** Minimize the risk of human error that can occur with manually administering and operating your log collection, ingestion, storage and querying functions, including saved queries and query packs, by automating as many of those functions as possible through code. Also, include alerts that report health status changes and the configuration of diagnostic settings for resources that send logs to your workspaces in your IaC code. Include the code with your other workload-related code to ensure that your safe deployment practices are maintained for the management of your workspaces.
- ✓ **Ensure that your workspaces are healthy, and you're notified when issues arise.** Like any other component of your workload, your workspaces can encounter issues. The issues can cost valuable time and resources to troubleshoot and resolve, and potentially leave your team unaware of the production workload status. Being able to proactively monitor workspaces and mitigate potential issues helps your operations teams minimize the time they spend troubleshooting and fixing issues.
- ✓ **Separate your production from nonproduction workloads.** Avoid unnecessary complexity that can cause extra work for an operations team by using different workspaces for your production environment than those used by nonproduction environments. Comingled data can also lead to confusion as testing activities might appear to be events in production.
- ✓ **Prefer built-in tools and functions over non-Microsoft solutions** Use built-in tools to extend the functionality of your monitoring and logging systems. You might need to put additional configurations in place to support requirements like recoverability or data sovereignty that aren't available out-of-the-box with Log Analytics workspaces. In these cases, whenever practical, use native Azure or Microsoft tools to keep the number of tools that your organization must support to a minimum.
- ✓ **Treat your workspaces as static rather than ephemeral components** Like other types of data stores, workspaces shouldn't be considered among the ephemeral components of your workload. The Well-Architected Framework generally favors immutable infrastructure and the ability to quickly and easily replace resources within your workload as part of your deployments. But the loss of workspace data can be catastrophic and irreversible. For this reason, leave workspaces out of deployment packages that replace infrastructure during updates, and only perform in-place upgrades on the workspaces.
- ✓ **Ensure that operations staff is trained on Kusto Query Language** Train staff to create or modify queries when needed. If operators are unable to write or modify

queries, it can slow critical troubleshooting or other functions as operators must rely on other teams to do that work for them.

Configuration recommendations for Operational Excellence

Expand table

Recommendation	Benefit
Design a workspace strategy to meet your business requirements. See Design a Log Analytics workspace architecture for guidance on designing a strategy for your Log Analytics workspaces. Include how many to create and where to place them. If you required your workload to use a centralized platform team offering, ensure that you set all necessary operational access. Also, construct alerts to ensure workload observability needs are met.	A single or at least minimal number of workspaces maximize your workload's operational efficiency. It limits the distribution of your operational and security data, increases visibility into potential issues, makes patterns easier to identify, and minimizes your maintenance requirements. You might have requirements for multiple workspaces such as multiple tenants, or you might need workspaces in multiple regions to support your availability requirements. So, ensure that you have appropriate processes in place to manage this increased complexity.
Use infrastructure as code (IaC) to deploy and manage your workspaces and associated functions.	Use infrastructure as code (IaC) to define the details of your workspaces in ARM templates, Azure BICEP, or Terraform. It lets you use your existing DevOps processes to deploy new workspaces and Azure Policy to enforce their configuration. Colocating all of your IaC code with your application code helps ensure that your safe deployment practices are maintained for all deployments.
Use Log Analytics workspace insights to track the health and performance of your Log Analytics workspaces, and create meaningful and actionable alerts to be proactively notified of operational issues. Log Analytics workspace insights provides a unified view of the usage, performance, health, agents, queries, and change log for all your workspaces. Each workspace has an operation table	Review the information that Log Analytics insights provides regularly to track the health and operation of each of your workspaces. When you use this information, it lets you create easily understood visualizations like dashboards or reports that operations and stakeholders can use to track the health of your workspaces. Create alert rules based on this table to be proactively notified when an operational issue occurs. You can use recommended alerts for the workspace to simplify how you create the most critical alert rules.

Recommendation	Benefit
that logs important activities affecting workspace.	
Practice continuous improvement by frequently revisiting Azure diagnostic settings on your resources, data collection rules, and application log verbosity. Ensure that you're optimizing your log collection strategy through frequent reviews of your resource settings. From an operational standpoint, look to reduce the noise in your logs by focusing on those logs that provide useful information about a resource's health status.	By optimizing in this manner, you enable operators to investigate and troubleshoot issues when they arise, or perform other routine, improvised, or emergency tasks. When new diagnostic categories are made available for a resource type, review the types of logs that are emitted with this category to understand whether enabling them might help you optimize your collection strategy. For example, a new category might be a subset of a larger set of activities that are being captured. The new subset might let you reduce the volume of logs coming in by focusing on the activities that are important for your operations to track.

Azure Policy and Azure Advisor

Azure offers no policies nor Azure Advisor recommendations related to the operational excellence of Log Analytics workspaces.

Performance efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance.

The Performance Efficiency design principles provide a high-level design strategy for achieving those capacity goals against the expected usage.

Design checklist for Performance Efficiency

Start your design strategy based on the **design review checklist for Performance Efficiency** for defining a baseline for your Log Analytics workspaces and associated functions.

- ✓ **Be familiar with fundamentals of log data ingestion latency in Azure Monitor.** There are several factors that contribute to latency when ingesting logs into your workspaces. Many of these factors are inherent to the Azure Monitor platform. Understanding the factors and the normal latency behavior can help you set appropriate expectations within your workload operations teams.

- ✓ **Separate your nonproduction and production workloads.** Production-specific workspaces mitigate any overhead that nonproduction systems might introduce. It reduces the overall footprint of your workspaces, requiring fewer resources to handle log data processing.
- ✓ **Choose the right deployment regions to meet your performance requirements.** Deploy your Log Analytics workspace and data collection endpoints (DCEs) close to your workload. Your choice of the appropriate region in which to deploy your workspace and your DCEs should be informed by where you deploy the workload. You might need to weigh the performance benefits of deploying your workspaces and DCEs in the same region as your workload against your reliability requirements if you have already deployed your workload into a region that cannot support those requirements for your log data.

Configuration recommendations for Performance Efficiency

Expand table

Recommendation	Benefit
Configure log query auditing and use Log Analytics workspace insights to identify slow and inefficient queries. Log query auditing stores the compute time required to run each query and the time until results are returned. Log Analytics workspace insights uses this data to list potentially inefficient queries in your workspace. Consider rewriting these queries to improve their performance. Refer to Optimize log queries in Azure Monitor for guidance on optimizing your log queries.	Optimized queries return results faster and use less resources on the back end, which makes the processes that rely on those queries more efficient as well.
Understand service limits for Log Analytics workspaces. In certain high-traffic implementations, you might run into service limits that affect your performance and your workspace or workload design. For example, the query API limits the number of records and data volume returned by a query. The Logs Ingestion API limits the size of each API call.	Understanding the limits that might affect the performance of your workspace helps you design appropriately to mitigate them. You might decide to use multiple workspaces to avoid hitting limits associated with a single workspace. Weigh the design decisions to mitigate service limits against requirements and targets for other pillars.

Recommendation	Benefit
For a complete list of Azure Monitor and Log Analytics workspaces limits and limits specific to the workspace itself, see Azure Monitor service limits.	
Create DCRs specific to data source types inside one or more defined observability scopes. Create separate DCRs for performance and events to optimize the backend processing compute utilization.	When you use separate DCRs for performance and events, it helps mitigate backend resource exhaustion. By having DCRs that combine performance events, it forces every associated virtual machine to transfer, process, and run configurations that might not be applicable according to the installed software. An excessive compute resource consumption and errors in processing a configuration might happen and cause the Azure Monitor Agent (AMA) to become unresponsive.

Azure Policy and Azure Advisor

Azure offers no policies nor Azure Advisor recommendations related to the performance of Log Analytics workspaces.

Next step

- Get best practices for a complete deployment of Azure Monitor.

Feedback

Was this page helpful? Yes No

Service Bus and reliability

Article • 03/13/2025

Fully manage enterprise message brokering with message queues and publish-subscribe topics used in Azure Service Bus. This service stores messages in a *broker* (for example, a *queue*) until the consuming party is ready to receive the messages.

Benefits include:

- Load-balancing across competing workers.
- Safely routing and transferring data and control across service, and application boundaries.
- Coordinating transactional work that requires a high-degree of reliability.

For more information about using Service Bus, reference Azure Service Bus Messaging. Learn how to set up messaging that connects applications and services across on-premises and cloud environments.

To understand how Service Bus contributes to a reliable workload, reference the following topics:

- Asynchronous messaging patterns and high availability
- Azure Service Bus Geo-disaster recovery
- Handling outages and disasters
- Availability zones

> **Note**
>
> Previously it was required to set the property `zoneRedundant` to `true` to enable availability zones, however this behavior has changed to enable availability zones by default. Existing namespaces are being migrated to availability zones where possible, and the property `zoneRedundant` is being deprecated. The property `zoneRedundant` might still show as `false`, even when availability zones has been enabled.

The following sections are specific to Azure Service Bus and reliability:

- Design considerations
- Configuration checklist
- Recommended configuration options
- Source artifacts

Design considerations

Maximize reliability with an Azure Service Bus uptime SLA. Properly configured applications can send or receive messages, or do other operations on a deployed Queue or Topic. For more information, reference the Service Bus SLA.

Other design considerations include:

- Express Entities
- Partitioned queues and topics

Besides the documentation on Service Bus Premium and Standard messaging tiers, the following features are only available on the Premium Stock Keeping Unit (SKU):

- Dedicated resources.
- Virtual network integration: Limits the networks that can connect to the Service Bus instance. Requires Service Endpoints to be enabled on the subnet. There are Trusted Microsoft services that are not supported when implementing Virtual Networks(for example, integration with Event Grid). For more information, reference Allow access to Azure Service Bus namespace from specific virtual networks.
- Private endpoints.
- IP Filtering/Firewall: Restrict connections to only defined `IPv4` addresses or `IPv4` address ranges.
- Event Grid integration: Available event types.
- Scale messaging units.
- Geo-Disaster Recovery (paired namespace).
- Geo-Replication (full data replication across regions).
- CMK (Customer Managed Key): Azure Service Bus encrypts data at rest and automatically decrypts it when accessed, but customers can also bring their own customer-managed key.

When deploying Service Bus with Geo-disaster recovery and in availability zones, the Service Level Operation (SLO) increases dramatically, but does not change the uptime SLA.

Checklist

Have you configured Azure Service Bus with reliability in mind?

- ✓ Evaluate Premium tier benefits of Azure Service Bus.
- ✓ Ensure that Service Bus Messaging Exceptions are handled properly.

- ✓ Connect to Service Bus with the Advanced Messaging Queue Protocol (AMQP) and use Service Endpoints or Private Endpoints when possible.
- ✓ Review the Best Practices for performance improvements using Service Bus Messaging.
- ✓ Implement geo-replication on the sender and receiver side to protect against outages and disasters.
- ✓ Configure Geo-Disaster.
- ✓ If you need mission-critical messaging with queues and topics, Service Bus Premium is recommended with Geo-Disaster Recovery.
- ✓ Implement high availability for the Service Bus namespace.
- ✓ Ensure related messages are delivered in guaranteed order.
- ✓ Evaluate different Java Messaging Service (JMS) features through the JMS API.
- ✓ Use .NET Nuget packages to communicate with Service Bus messaging entities.
- ✓ Implement resilience for transient fault handling when sending or receiving messages.
- ✓ Implement auto-scaling of messaging units.

Configuration recommendations

Consider the following recommendations to optimize reliability when configuring Azure Service Bus:

Expand table

Recommendation	Description
Evaluate Premium tier benefits of Azure Service Bus.	Consider migrating to the Premium tier of Service Bus to take advantage of platform-supported outage and disaster protection.
Connect to Service Bus with the AMQP protocol and use Service Endpoints or Private Endpoints when possible.	This recommendation keeps traffic on the Azure Backbone. *Note: The default connection protocol for* `Microsoft.Azure.ServiceBus` *and* `Windows.Azure.ServiceBus` *namespaces is* `AMQP`.
Implement geo-replication on the sender and receiver side to protect against outages and disasters.	Standard tier supports only the implementation of sender and receiver-side geo-redundancy. An outage or disaster in an Azure Region could cause downtime for your solution.
Configure Geo-Disaster.	- Active/Active - Active/Passive - Paired Namespace (Active/Passive) - Geo-Replication (Active/Passive with full data replication)

Recommendation	Description
	- *Note: The secondary region should preferably be an Azure paired region*.
If you need mission-critical messaging with queues and topics, Service Bus Premium is recommended with Geo-Disaster Recovery.	Choosing the pattern is dependent on the business requirements and the recovery time objective (RTO).
Implement high availability for the Service Bus namespace.	Premium tier supports Geo-disaster recovery and replication at the namespace level. At this level, Premium tier provides high availability for metadata disaster recovery using primary and secondary disaster recovery namespaces.
Ensure related messages are delivered in guaranteed order.	Be aware of the requirement to set a Partition Key, Session ID, or Message ID on each message to ensure related messages send to the same partition in the messaging entity.
Evaluate different JMS features through the JMS API.	Features available through the JMS 2.0 API (and its Software Development Kit (SDK)) are not the same as the features available through the native SDK. For example, Service Bus Sessions are not available in JMS.
Implement resilience for transient fault handling when sending or receiving messages.	It is essential to implement suitable transient fault handling and error handling for send and receive operations to maintain throughput and to prevent message loss.
Implement auto-scaling of messaging units, to ensure that you have enough resources available for your workloads.	

Source artifacts

- To identify premium Service Bus Instances that are not using private endpoints, use the following query:

```kusto
Resources
| where
  type == 'microsoft.servicebus/namespaces'
| where
```

```
  sku.tier == 'Premium'
  and isempty(properties.privateEndpointConnections)
```

- To identify Service Bus Instances that are not on the premium tier, use the following query:

```Kusto
Resources
| where
  type == 'microsoft.servicebus/namespaces'
| where
  sku.tier != 'Premium'
```

Next step

Service Bus and operational excellence

Feedback

Was this page helpful? Yes No

Service Bus and operational excellence

Article • 11/14/2023

Fully manage enterprise message brokering with message queues and publish-subscribe topics using Azure Service Bus. This service stores messages in a *broker* (for example, a *queue*) until the consuming party is ready to receive the messages.

Benefits include:

- Load-balancing work across competing workers.
- Safely routing and transferring data and control across service, and application boundaries.
- Coordinating transactional work that requires a high-degree of reliability.

For more information about using Service Bus, reference Azure Service Bus Messaging. Learn how to set up messaging that connects applications and services across on-premises and cloud environments.

To understand how Service Bus promotes operational excellence, reference the following topics:

- Handling outages and disasters
- Throttling operations on Azure Service Bus

The following sections are specific to Azure Service Bus and operational excellence:

- Design considerations
- Configuration checklist
- Recommended configuration options
- Source artifacts

Design considerations

Maximize reliability with an Azure Service Bus uptime Service Level Agreement (SLA). Properly configured applications can send or receive messages, or do other operations on a deployed Queue or Topic. For more information, reference the Service Bus SLA.

Other design considerations include:

- Express Entities
- Partitioned queues and topics

Besides the documentation on Service Bus Premium and Standard messaging tiers, the following features are only available on the Premium Stock Keeping Unit (SKU):

- Dedicated resources.
- Virtual network integration: Limits the networks that can connect to the Service Bus instance. Requires Service Endpoints to be enabled on the subnet. There are Trusted Microsoft services that are not supported when implementing Virtual Networks (for example, integration with Event Grid). For more information, reference Allow access to Azure Service Bus namespace from specific virtual networks.
- Private endpoints.
- IP Filtering/Firewall: Restrict connections to only defined `IPv4` addresses or `IPv4` address ranges.
- Availability zones: Provides enhanced availability by spreading replicas across availability zones within one region at no extra cost.
- Event Grid integration: Available event types.
- Scale messaging units.
- Geo-Disaster Recovery (paired namespace).
- BYOK (Bring Your Own Key): Azure Service Bus encrypts data at rest and automatically decrypts it when accessed, but customers can also bring their own customer-managed key.

When deploying Service Bus with Geo-disaster recovery and in availability zones, the Service Level Objective (SLO) increases dramatically, but does not change the uptime SLA.

Checklist

Have you configured Azure Service Bus with operational excellence in mind?

- ✓ Ensure that Service Bus Messaging Exceptions are handled properly.
- ✓ Connect to Service Bus with the Advanced Message Queuing Protocol (AMQP) and use Service Endpoints or Private Endpoints when possible.
- ✓ Establish a process to actively monitor the dead-letter queue (dlq) messages.
- ✓ Review the Best Practices for performance improvements using Service Bus Messaging.
- ✓ Analyze the differences between Azure Storage Queues and Azure Service Bus Queues.

Configuration recommendations

Consider the following recommendation to optimize reliability when configuring Azure Service Bus:

Recommendation	Description
Connect to Service Bus with the AMQP protocol and use Service Endpoints or Private Endpoints when possible.	This recommendation keeps traffic on the Azure Backbone. *Note: The default connection protocol for* `Microsoft.Azure.ServiceBus` *and* `Windows.Azure.ServiceBus` *namespaces is* `AMQP`.
Establish a process to actively monitor the dead-letter queue (dlq) messages.	The dead-letter queue holds messages that cannot be processed or cannot be delivered to any receiver. It is important to monitor this queue to examine the issue cause, apply required corrections, and to resubmit messages.
Analyze the differences between Azure Storage Queues and Azure Service Bus Queues.	You will find that Azure Service Bus Messaging Entities are more advanced, reliable, and feature-rich than Azure Storage Queues. If your requirement is for simple queue messaging without requirements for reliable messaging, then Azure Storage Queues may be a more suitable option.

Source artifacts

- To identify premium Service Bus Instances that aren't using private endpoints, use the following query:

  Kusto

  ```
  Resources
  | where
    type == 'microsoft.servicebus/namespaces'
  | where
    sku.tier == 'Premium'
    and isempty(properties.privateEndpointConnections)
  ```

- To identify Service Bus Instances that are not on the premium tier, use the following query:

  Kusto

  ```
  Resources
  | where
    type == 'microsoft.servicebus/namespaces'
  | where
    sku.tier != 'Premium'
  ```

Next step

Queue Storage and reliability

Architecture best practices for Azure Virtual Machines and scale sets

Article • 03/27/2025

Azure Virtual Machines is a type of compute service that you can use to create and run virtual machines (VMs) on the Azure platform. It provides flexibility in different SKUs, operating systems, and configurations with various billing models.

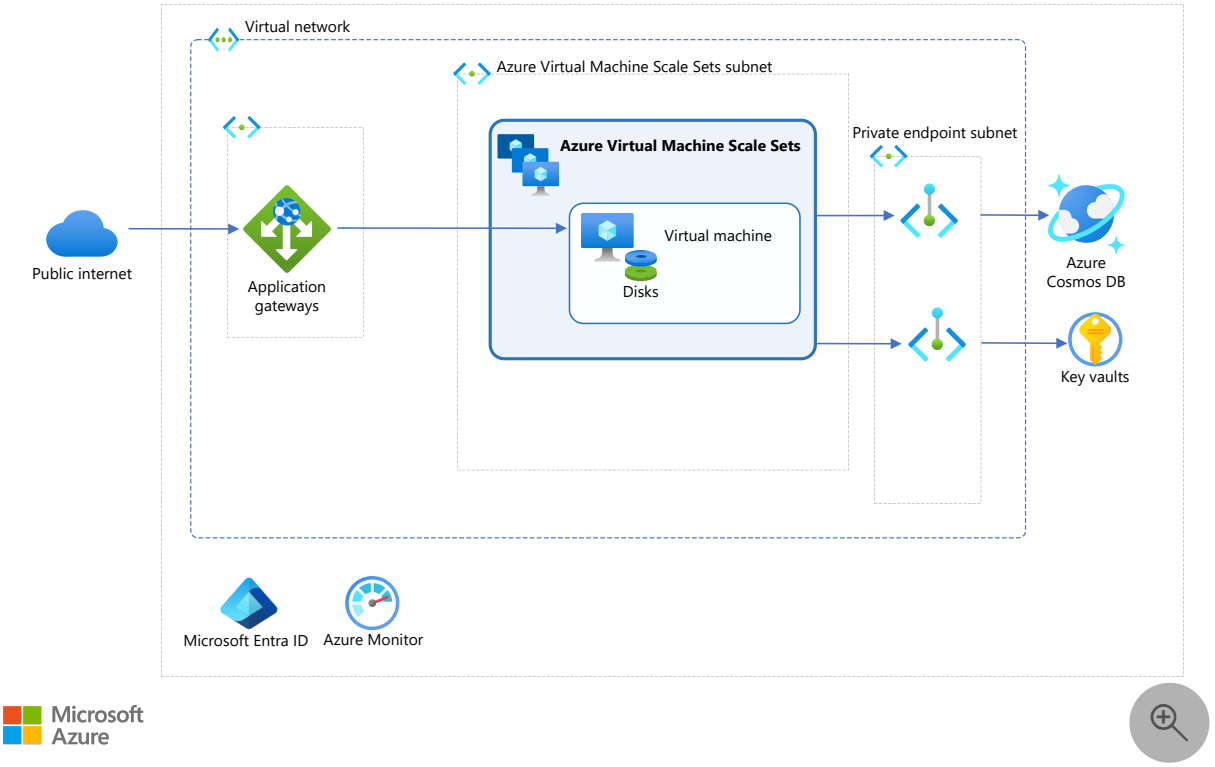
This article assumes that as an architect, you've reviewed the compute decision tree and chose Virtual Machines as the compute service for your workload. The guidance in this article provides architectural recommendations that are mapped to the principles of the Well-Architected Framework pillars.

> **Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern along with design strategies localized to the technology scope.
>
> Also included are recommendations on the technology capabilities that can help materialize those strategies. The recommendations don't represent an exhaustive list of all configurations available for Virtual Machines and its dependencies. Instead, they list the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept or to optimize your existing environments.
>
> Foundational architecture that demonstrates the key recommendations: Virtual Machines baseline architecture.

Technology scope

This review focuses on the interrelated decisions for the following Azure resources:

- Virtual Machines
- Azure Virtual Machine Scale Sets



Disks are a crucial dependency for VM-based architectures but aren't covered in this article. For more information, see Architecture best practices for Azure Disk Storage.

Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover fast from failures**.

Reliability design principles provide a high-level design strategy applied for individual components, system flows, and the system as a whole.

Design checklist

Start your design strategy based on the design review checklist for Reliability. Determine its relevance to your business requirements while keeping in mind the SKUs and features of VMs and their dependencies. Extend the strategy to include more approaches as needed.

- ✓ **Review Virtual Machines quotas and limits** that might pose design restrictions. VMs have specific limits and quotas, which vary based on the type of VM or the region. There might be subscription restrictions, such as the number of VMs per subscription or the number of cores per VM. If other workloads share your subscription, then your ability to consume data might be reduced. Check limits on VMs, virtual machine scale sets, and managed disks.

- ✓ **Conduct a failure mode analysis** to minimize points of failure by analyzing VM interactions with the network and storage components. Choose configurations like ephemeral operating system (OS) disks to localize disk access and avoid network hops. Add a load balancer to enhance self-preservation by distributing network traffic across multiple VMs, which improves availability and reliability.

- ✓ **Calculate your composite service-level objectives (SLOs) based on Azure service-level agreements (SLAs).** Ensure that your SLO isn't higher than the Azure SLAs to avoid unrealistic expectations and potential problems.

  Be aware of the complexities that dependencies introduce. For example, some dependencies, like virtual networks and network interface cards (NICs), don't have their own SLAs. Other dependencies, such as an associated data disk, have SLAs that integrate with VM SLAs. You should consider these variations because they can affect VM performance and reliability.

  Factor in the critical dependencies of VMs on components like disks and networking components. If you understand these relationships, then you can determine the critical flows that affect reliability.

- ✓ **Create state isolation.** Workload data should be on a separate data disk to prevent interference with the OS disk. If a VM fails, you can create a new OS disk with the same data disk, which ensures resilience and fault isolation. For more information, see Ephemeral OS disks.

- ✓ **Make VMs and their dependencies redundant across zones.** If a VM fails, the workload should continue to function because of redundancy. Include dependencies in your redundancy choices. For example, use the built-in redundancy options that are available with disks. Use zone-redundant IP addresses to ensure data availability and high uptime.

- ✓ **Be ready to scale up and scale out** to prevent service-level degradation and to avoid failures. Virtual Machine Scale Sets have autoscale capabilities that create new instances as required and distribute the load across multiple VMs and availability zones.

- ✓ **Explore the automatic recovery options.** Azure supports health degradation monitoring and self-healing features for VMs. For example, scale sets provide automatic instance repairs. In more advanced scenarios, self-healing involves using Azure Site Recovery, having a passive standby to fail over to, or redeploying from infrastructure as code (IaC). The method that you choose should align with the business requirements and your organizational operations. For more information, see VM service disruptions.

- ✓ **Rightsize the VMs and their dependencies.** Understand your VM's expected work to ensure that it's not undersized and can handle the maximum load. Have extra capacity to mitigate failures.

- ✓ **Create a comprehensive disaster recovery plan.** Disaster preparedness involves creating a comprehensive plan and deciding on a technology for recovery.

  Dependencies and stateful components, such as attached storage, can complicate recovery. If disks go down, then that failure affects the VM's functioning. Include a clear process for these dependencies in your recovery plans.

- ✓ **Run operations with rigor.** Reliability design choices must be supported by effective operations based on the principles of monitoring, resiliency testing in production, automated application VM patches and upgrades, and consistency of deployments. For operational guidance, see Operational Excellence.

Recommendations

Expand table

Recommendation	Benefit
(Scale set) **Use Virtual Machine Scale Sets in** Flexible orchestration mode to deploy VMs.	Future-proof your application for scaling and take advantage of the high availability guarantees that spread VMs across fault domains in a region or an availability zone.
(VMs) **Implement heath endpoints** that emit instance health statuses on VMs. (Scale set) Enable automatic repairs on the scale set by specifying the preferred repair action. Consider setting a time frame during which automatic repairs pause if the VM's state changes.	Maintain availability even if an instance is deemed unhealthy. Automatic repairs initiate recovery by replacing the faulty instance. Setting a time window can prevent inadvertent or premature repair operations.
(Scale set) Enable overprovisioning on scale sets.	Overprovisioning reduces deployment times and has a cost benefit because the extra VMs aren't billed.
(Scale set) Preallocate instances with standby pools.	Standby pool instances remain dormant but are ready to take over workloads if a failure occurs. This capability enhances the system's reliability.
(Scale set) Allow Flexible orchestration to spread the VM instances across as many fault domains as possible.	This option isolates fault domains. During maintenance periods, when one fault domain is updated, VM instances are available in the other fault domains.

Recommendation	Benefit
(Scale set) **Deploy across availability zones** on scale sets. Set up at least two instances in each zone. **Zone balancing** equally spreads the instances across zones.	The VM instances are provisioned in physically separate locations within each Azure region that are tolerant to local failures. Keep in mind that, depending on resource availability, there might be an uneven number of instances across zones. Zone balancing supports availability by making sure that, if one zone is down, the other zones have sufficient instances. Two instances in each zone provide a buffer during upgrades.
(Scale set) To enhance service uptime while maintaining control over the cost implications of upgrades, enable **MaxSurge**.	New instances are created in batches by using the latest scale model. After the new instances are healthy, the old instances are deleted in batches. This process continues until all instances are updated, which ensures no downtime during updates.
(VMs) Take advantage of the **capacity reservations feature**.	Capacity is reserved for your use and is available within the scope of the applicable SLAs. You can delete capacity reservations when you no longer need them, and billing is consumption based.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload.

The Security design principles provide a high-level design strategy for achieving those goals by applying approaches to the technical design of Virtual Machines.

Design checklist

Start your design strategy based on the design review checklist for Security and identify vulnerabilities and controls to improve the security posture. Extend the strategy to include more approaches as needed.

- ✓ **Review the security baselines** for Linux and Windows VMs and Virtual Machine Scale Sets.

  As part of your baseline technology choices, consider the security features of the VM SKUs that support your workload.

- ✓ **Ensure timely and automated security patching and upgrades.** Make sure updates are automatically rolled out and validated by using a well-defined process. Use a solution like Azure Automation to manage OS updates and maintain security compliance by making critical updates.

- ✓ **Identify the VMs that hold state.** Make sure that data is classified according to the sensitivity labels that your organization provides. Protect data by using security controls like appropriate levels of at-rest and in-transit encryption. If you have high sensitivity requirements, consider using high-security controls like double encryption and Azure confidential computing to protect data-in-use.

- ✓ **Provide segmentation** to the VMs and scale sets by setting network boundaries and access controls. Place VMs in resource groups that share the same lifecycle.

- ✓ **Apply access controls to the identities** that try to reach the VMs and also to the VMs that reach other resources. Use Microsoft Entra ID for authentication and authorization needs. Put strong passwords, multifactor authentication, and role-based access control (RBAC) in place for your VMs and their dependencies, like secrets, to permit allowed identities to perform only the operations that are expected of their roles.

  Restrict resource access based on conditions by using Microsoft Entra Conditional Access. Define the conditional policies based on duration and the minimum set of required permissions.

- ✓ **Use network controls to restrict ingress and egress traffic.** Isolate VMs and scale sets in Azure Virtual Network and define network security groups to filter traffic. Protect against distributed denial of service (DDoS) attacks. Use load balancers and firewall rules to protect against malicious traffic and data exfiltration attacks.

  Use Azure Bastion to provide more secure connectivity to the VMs for operational access.

  Communication to and from the VMs to platform as a service (PaaS) solutions should be over private endpoints.

- ✓ **Reduce the attack surface** by hardening OS images and removing unused components. Use smaller images and remove binaries that aren't required to run the workload. Tighten the VM configurations by removing features, like default accounts and ports, that you don't need.

- ✓ **Protect secrets** such as the certificates that you need to protect data in transit. Consider using the Azure Key Vault extension for Windows or Linux that automatically refreshes the certificates stored in a key vault. When it detects a change in the certificates, the extension retrieves and installs the corresponding certificates.

- ✓ **Threat detection.** Monitor VMs for threats and misconfigurations. Use Defender for Servers to capture VM and OS changes, and maintain an audit trail of access, new accounts, and changes in permissions.

- ✓ **Threat prevention.** Protect against malware attacks and malicious actors by implementing security controls like firewalls, antivirus software, and intrusion detection systems. Determine whether a Trusted Execution Environment (TEE) is required.

Recommendations

Expand table

Recommendation	Benefit
(Scale set) **Assign a managed identity to scale sets.** All VMs in the scale set get the same identity through the specified VM profile. (VMs) You can also assign a managed identity to individual VMs when you create them and then add it to a scale set if needed.	When VMs communicate with other resources, they cross a trust boundary. Scale sets and VMs should authenticate their identity before communication is allowed. Microsoft Entra ID handles that authentication by using managed identities.
(Scale set) **Choose VM SKUs that have security features.** For example, some SKUs support BitLocker encryption, and confidential computing provides encryption of data-in-use. Review the features to understand the limitations.	Azure-provided features are based on signals that are captured across many tenants and can protect resources better than custom controls. You can also use policies to enforce those controls.
(VMs, scale set) **Apply organization-recommended tags** in the provisioned resources.	Tagging is a common way to segment and organize resources and can be crucial during incident management. For more information, see Purpose of naming and tagging.
(VMs, scale set) **Set a security profile** with the security features that you want to enable in the VM configuration. For example, when you specify encryption at host in the profile, the data that's stored on the VM host is encrypted at rest and flows are encrypted to the storage service.	The features in the security profile are automatically enabled when the VM is created. For more information, see Azure security baseline for Virtual Machine Scale Sets.
(VMs) **Choose secure networking options** for your VM's network profile. Don't directly associate public IP addresses to your VMs and don't enable IP forwarding. Ensure that all virtual network interfaces have an associated network security group.	You can set segmentation controls in the networking profile. Attackers scan public IP addresses. This activity makes VMs vulnerable to threats.

Recommendation	Benefit
(VMs) **Choose secure storage options** for your VM's storage profile. Enable disk encryption and data-at-rest encryption by default. Disable public network access to the VM disks.	Disabling public network access helps prevent unauthorized access to your data and resources.
(VMs, scale set) **Include extensions in your VMs** that protect against threats. For example, - Key Vault extension for Windows and Linux - Microsoft Entra ID authentication - Microsoft Antimalware for Azure Cloud Services and Virtual Machines - Azure Disk Encryption extension for Windows and Linux.	The extensions are used to bootstrap the VMs with the right software that protects access to and from the VMs. Microsoft-provided extensions are updated frequently to keep up with the evolving security standards.

Cost Optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget while meeting business requirements.

The Cost Optimization design principles provide a high-level design strategy for achieving those goals and making tradeoffs as necessary in the technical design related to Virtual Machines and its environment.

Design checklist

Start your design strategy based on the design review checklist for Cost Optimization for investments. Fine-tune the design so that the workload is aligned with the budget that's allocated for the workload. Your design should use the right Azure capabilities, monitor investments, and find opportunities to optimize over time.

- ✓ **Estimate realistic costs.** Use the pricing calculator to estimate the costs of your VMs. Identify the best VM for your workload by using the VM selector. For more information, see Linux and Windows pricing.
- ✓ **Implement cost guardrails.** Use governance policies to restrict resource types, configurations, and locations. Use RBAC to block actions that can lead to overspending.

- ✓ **Choose the right resources.** Your selection of VM plan sizes and SKUs directly affect the overall cost. Choose VMs based on workload characteristics. Is the workload CPU intensive or does it run interruptible processes? Each SKU has associated disk options that affect the overall cost.

- ✓ **Choose the right capabilities for dependent resources.** Save on backup storage costs for the vault-standard tier by using Azure Backup storage with reserved capacity. It offers a discount when you commit to a reservation for either one year or three years.

  The archive tier in Azure Storage is an offline tier that's optimized for storing blob data that's rarely accessed. The archive tier offers the lowest storage costs but higher data retrieval costs and latency compared to the hot and cool online tiers.

  Consider using zone-to-zone disaster recovery for VMs to recover from site failure while reducing the complexity of availability by using zone-redundant services. There can be cost benefits from reduced operational complexity.

- ✓ **Choose the right billing model.** Evaluate whether commitment-based models for computing optimize costs based on the business requirements of workload. Consider these Azure options:
  - **Azure reservations**: Prepay for predictable workloads to reduce costs compared to consumption-based pricing.

    > ⓘ **Important**
    >
    > Purchase reserved instances to reduce Azure costs for workloads that have stable usage. Manage usage to make sure that you're not paying for more resources than you're using. Keep reserved instances simple and keep management overhead low to reduce costs.

  - **Savings plan**: If you commit to spend a fixed hourly amount on compute services for one or three years, then this plan can reduce costs.
  - **Azure Hybrid Benefit**: Save when you migrate your on-premises VMs to Azure.

- ✓ **Monitor usage.** Continuously monitor usage patterns and detect unused or underutilized VMs. For those instances, shut down VM instances when they're not in use. Monitoring is a key approach of Operational Excellence. For more information, see the recommendations in Operational Excellence.

- ✓ **Look for ways to optimize.** Some strategies include choosing the most cost-effective approach between increasing resources in an existing system, or scaling up, and adding more instances of that system, or scaling out. You can offload demand by distributing it to other resources, or you can reduce demand by implementing priority queues, gateway

offloading, buffering, and rate limiting. For more information, see the recommendations in Performance Efficiency.

Recommendations

Expand table

Recommendation	Benefit
(VMs, scale set) **Choose the right VM plan size and SKU.** Identify the best VM sizes for your workload. Use the VM selector to identify the best VM for your workload. See Windows and Linux pricing. For workloads like highly parallel batch processing jobs that can tolerate some interruptions, consider using Azure Spot Virtual Machines. Spot virtual machines are good for experimenting, developing, and testing large-scale solutions.	SKUs are priced according to the capabilities that they offer. If you don't need advanced capabilities, don't overspend on SKUs. Spot virtual machines take advantage of the surplus capacity in Azure at a lower cost.
(Scale set) **Mix regular VMs with spot virtual machines.** Flexible orchestration lets you **distribute spot virtual machines** based on a specified percentage.	Reduce compute infrastructure costs by applying the deep discounts of spot virtual machines.
(Scale set) **Reduce the number of VM instances when demand decreases.** **Set a scale-in policy** based on criteria.	Scaling in resources when they're not in use reduces the number of VMs that run in the scale set, which saves costs.
(VMs) **Stop VMs during off-hours.** You can use the **Azure Automation Start/Stop** feature and configure it according to your business needs.	The Start/Stop feature is a low-cost automation option that can significantly affect your idle instance costs.
(VMs) **Free up CPU resources** by using **Azure Boost**.	Offloading back-end virtualization processes frees up CPU resources for the guest virtual machines. This optimization results in improved performance. Azure Boost is only available on specific VMs, so ensure that you also choose VM sizes that have Azure Boost enabled.
(VMs, scale set) **Take advantage of license mobility by using Azure Hybrid Benefit.** VMs have a licensing option that allows you to bring your own on-premises Windows Server OS licenses to Azure.	You can maximize your on-premises licenses while getting the benefits of the cloud.

Recommendation	Benefit
Azure Hybrid Benefit also lets you bring certain Linux subscriptions to Azure.	

Operational Excellence

Operational Excellence primarily focuses on procedures for **development practices, observability, and release management**.

The Operational Excellence design principles provide a high-level design strategy for achieving those goals for the operational requirements of the workload.

Design checklist

Start your design strategy based on the design review checklist for Operational Excellence for defining processes for observability, testing, and deployment related to Virtual Machines and scale sets.

- ✓ **Monitor the VM instances.** Collect logs and metrics from VM instances to monitor resource usage and measure the health of the instances. Some common metrics include CPU usage, number of requests, and input/output (I/O) latency. Set up Azure Monitor alerts to be notified about problems and to detect configuration changes in your environment.

- ✓ **Monitor the health of the VMs and their dependencies.**
  - Deploy monitoring components to collect logs and metrics that give a comprehensive view of your VMs, guest OS, and boot diagnostics data. Virtual Machine Scale Sets roll up telemetry, which allows you to view health metrics at an individual VM level or as an aggregate. Use Azure Monitor to view this data for each VM or aggregated across multiple VMs. For more information, see Recommendations on monitoring agents.
  - Take advantage of networking components that check the health status of VMs. For example, Azure Load Balancer pings VMs to detect unhealthy VMs and reroute traffic accordingly.
  - Set up Azure Monitor alert rules. Determine important conditions in your monitoring data to identify and address problems before they affect the system.

- ✓ **Create a maintenance plan** that includes regular system patching as a part of routine operations. Include emergency processes that allow for immediate patch application. You can have custom processes to manage patching or partially delegate the task to Azure. Azure provides features for individual VM maintenance. You can set up maintenance windows to minimize disruptions during updates. During platform updates, fault domain

considerations are key for resilience. We recommend that you deploy at least two instances in a zone. Two VMs per zone guarantees a minimum of one VM in each zone because only one fault domain in a zone is updated at a time. So, for three zones, provision at least six instances.

- ✓ **Automate processes for bootstrapping, running scripts, and configuring VMs.** You can automate processes by using extensions or custom scripts. We recommend the following options:
  - The Key Vault VM extension automatically refreshes certificates that are stored in a key vault.
  - The Azure Custom Script Extension for Windows and Linux downloads and runs scripts on Virtual Machines. Use this extension for post-deployment configuration, software installation, or any other configuration or management task.
  - Use cloud-init to set up the startup environment for Linux-based VMs.
- ✓ **Have processes for installing automatic updates.** Consider using Automatic VM guest patching for a timely rollout of critical patches and security patches. Use Azure Update Manager to manage OS updates for your Windows and Linux VMs in Azure.
- ✓ **Build a test environment** that closely matches your production environment to test updates and changes before you deploy them to production. Have processes in place to test the security updates, performance baselines, and reliability faults. Take advantage of Azure Chaos Studio fault libraries to inject and simulate error conditions. For more information, see Azure Chaos Studio fault and action library.
- ✓ **Manage your quota.** Plan what level of quota your workload requires and review that level regularly as the workload evolves. If you need to increase or decrease your quota, request those changes early.

Recommendations

Expand table

Recommendation	Benefit
(Scale set) **Virtual Machine Scale Sets in Flexible orchestration mode** can help simplify the deployment and management of your workload. For example, you can easily manage self-healing by using automatic repairs.	Flexible orchestration can manage VM instances at scale. Handing individual VMs adds operational overhead. For example, when you delete VM instances, the associated disks and NICs are also automatically deleted. VM instances are spread across multiple fault

Recommendation	Benefit
	domains so that update operations don't disrupt service.
(Scale set) **Keep your VMs up to date** by setting an **upgrade policy**. We recommend rolling upgrades. However, if you need granular control, choose to upgrade manually. For Flexible orchestration, you can use **Azure Update Manager**.	Security is the primary reason for upgrades. Security assurances for the instances shouldn't decay over time. Rolling upgrades are done in batches. This approach ensures that all instances aren't down at the same time.
(VMs, scale set) Automatically deploy VM applications from the Azure Compute Gallery by defining the applications in the **profile**.	The VMs in the scale set are created and the specified apps are preinstalled, which makes management easier.
Install prebuilt software components as extensions as part of bootstrapping. Azure supports many extensions that can be used to configure, monitor, secure, and provide utility applications for your VMs. **Enable automatic upgrades** on extensions.	Extensions can help simplify the software installation at scale without you having to manually install, configure, or upgrade it on each VM.
(VMs, scale set) **Monitor and measure the health** of the VM instances. Deploy the **Monitor agent** extension to your VMs to collect monitoring data from the guest OS with OS-specific **data collection rules**. Enable **VM insights** to monitor health and performance and to view trends from the collected data. Use **boot diagnostics** to get information as VMs boot. Boot diagnostics also diagnose boot failures.	Monitoring data is at the core of incident resolution. A comprehensive monitoring stack provides information about how the VMs are performing and their health. By continuously monitoring the instances, you can be ready for or prevent failures like performance overload and reliability problems.

Performance Efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance.

The Performance Efficiency design principles provide a high-level design strategy for achieving those capacity goals against the expected usage.

Design checklist

Start your design strategy based on the design review checklist for Performance Efficiency. Define a baseline that's based on key performance indicators for Virtual Machines and scale sets.

- ✓ **Define performance targets.** Identify VM metrics to track and measure against performance indicators as response time, CPU utilization, and memory utilization, as well as workload metrics such as transactions per second, concurrent users, and availability and health.
- ✓ **Factor in the performance profile of VMs, scale sets, and disk configuration in your capacity planning.** Each SKU has a different profile of memory and CPU and behaves differently depending on the type of workload. Conduct pilots and proofs of concept to understand performance behavior under the specific workload.
- ✓ **VM performance tuning.** Take advantage of performance optimization and enhancing features as required by the workload. For example, use locally attached Non-Volatile Memory Express (NVMe) for high performance use cases and accelerated networking, and use Premium SSD v2 for better performance and scalability.
- ✓ **Take the dependent services into account.** Workload dependencies, like caching, network traffic, and content delivery networks, that interact with the VMs can affect performance. Also, consider geographical distribution, like zones and regions, which can add latency.
- ✓ **Collect performance data.** Follow the Operational Excellence best practices for monitoring and deploy the appropriate extensions to view metrics that track against performance indicators.
- ✓ **Proximity placement groups.** Use proximity placement groups in workloads where low latency is required to ensure that VMs are physically located close to each other.

Recommendations

Expand table

Recommendation	Benefit
(VMs, scale set) **Choose SKUs for VMs** that align with your capacity planning.	Rightsizing your VMs is a fundamental decision that significantly affects the performance of your workload.

Recommendation	Benefit
Have a good understanding of your workload requirements, including the number of cores, memory, storage, and network bandwidth so that you can filter out unsuitable SKUs.	Without the right set of VMs, you might experience performance problems and accrue unnecessary costs.
(VMs, scale set) Deploy latency-sensitive workload VMs in **proximity placement groups**.	Proximity placement groups reduce the physical distance between Azure compute resources, which can improve performance and reduce network latency between stand-alone VMs, VMs in multiple availability sets, or VMs in multiple scale sets.
(VMs) Consider enabling **accelerated networking**.	It enables single root I/O virtualization (SR-IOV) to a VM, which greatly improves its networking performance.
(VMs, scale set) **Set autoscale rules** to increase or decrease the number of VM instances in your scale set based on demand.	If your application demand increases, the load on the VM instances in your scale set increases. Autoscale rules ensure that you have enough resources to meet the demand.

Azure policies

Azure provides an extensive set of built-in policies related to Virtual Machines and its dependencies. Some of the preceding recommendations can be audited through Azure Policy. For example, you can check whether:

- **Encryption is enabled at the host level.** Ensure that encryption is enabled at the host level to provide extra security for your VM data.
- **Anti-malware extensions are deployed.** Verify that anti-malware extensions are deployed on VMs that run Windows Server and set for automatic updates to ensure ongoing protection.
- **Automatic OS image patching is enabled.** Check that automatic OS image patching is enabled on scale sets to ensure that your VMs stay updated with security patches.
- **Only approved VM extensions are installed.** Verify that only approved extensions are installed on your VMs. This approach helps minimize the risk of security vulnerabilities.
- **Monitor and dependency agents are enabled.** Ensure that the Monitor agent and dependency agents are enabled on all new VMs to facilitate monitoring and dependency management.

- **Only allowed VM SKUs are deployed.** Confirm that only approved VM SKUs are deployed. This policy ensures adherence to your cost constraints and resource requirements.
- **Private endpoints are used for disk access.** Ensure that private endpoints are used to securely access disk resources. This approach helps prevent exposure to public networks.
- **Vulnerability detection is enabled.** Enable vulnerability detection for your VMs. For Windows machines, configure rules such as daily scans with Microsoft Defender Antivirus to detect potential threats.

For comprehensive governance, review the Azure Policy built-in definitions for Virtual Machines and other policies that might affect the security of the compute layer.

Azure Advisor recommendations

Azure Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments.

For more information, see Azure Advisor.

Related content

Consider the following articles as resources that demonstrate the recommendations highlighted in this article.

- Use the following reference architectures as examples of how you can apply this article's guidance to a workload:
  - Single VM architectures: Linux VM and Windows VM
  - Foundational architecture that focuses on infrastructure recommendations: Virtual Machines baseline architecture
- Build implementation expertise by using the following product documentation:
  - Virtual Machines
  - Virtual Machine Scale Sets

Architecture best practices for Azure Virtual Network

Article • 02/03/2025

Azure Virtual Network is a fundamental building block for establishing a private network on Azure. You can use it to enable communication between Azure resources and provide internet connectivity. Virtual Network also integrates with on-premises systems. It includes built-in filtering capabilities to ensure that only expected, allowed, and safe traffic reaches the components within the network boundaries.

This article assumes that as an architect, you're familiar with the networking constructs on Azure. The guidance is focused on architectural recommendations that are mapped to the principles of the Well-Architected Framework pillars.

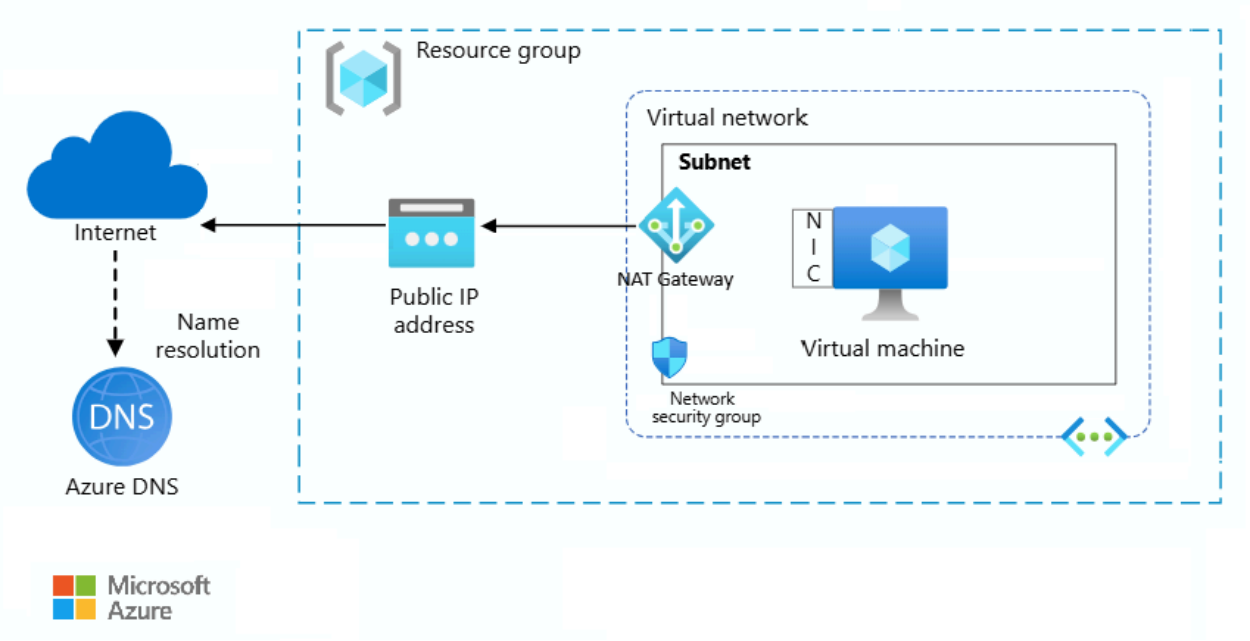
> **ⓘ Important**
>
> **How to use this guide**
>
> Each section has a *design checklist* that presents architectural areas of concern along with design strategies localized to the technology scope.
>
> Also included are recommendations for the technology capabilities that can help materialize those strategies. The recommendations don't represent an exhaustive list of all configurations that are available for Virtual Network and its dependencies. Instead, they list the key recommendations mapped to the design perspectives. Use the recommendations to build your proof-of-concept or to optimize your existing environments.
>
> Foundational architecture that demonstrates the key recommendations: **Hub-spoke network topology in Azure**.

Technology scope

This review focuses on the interrelated decisions for the following Azure resources:

- Virtual Network and its subnets
- Network interface cards (NICs)
- Private endpoints
- Network security groups (NSGs)
- IP addresses and IP address allocations
- Route tables

- Network managers



There are other services associated with Virtual Network, such as load balancers. Those services are covered in their respective guides.

Reliability

The purpose of the Reliability pillar is to provide continued functionality by **building enough resilience and the ability to recover fast from failures**.

Reliability design principles provide a high-level design strategy applied for individual components, system flows, and the system as a whole.

Design checklist

Start your design strategy based on the design review checklist for Reliability. Determine its relevance to your business requirements while keeping in mind the features of Virtual Network and its dependencies. Extend the strategy to include more approaches as needed.

- ✓ **Set your reliability targets**. Virtual Network and most of its subservices don't have Microsoft-backed service-level agreement (SLA) guarantees. However, specific services like load balancers, NICs, and public IP addresses do have SLAs. You should have a good understanding of the coverage provided around the published percentile by Azure. Keep in mind that your central IT services organization typically owns the virtual network and central services. Make sure that your objective calculations include this dependency.

- **Mitigate points of failure**. Do failure mode analysis and identify single points of failure in network connections.

  The following examples show single points of failure in network connections:

  Expand table

Failure	Mitigation
Public IP address failure in a single availability zone.	Deploy IP address resources across zones or use a secondary IP address with a load balancer.
Network virtual appliance (NVA) failure in a single zone.	Deploy a secondary NVA in another zone and use a load balancer to direct traffic to the NVA.
Latency in workloads that are spread across regions or zones, which lowers throughput and causes time-outs.	Co-locate resources in one region or zone. Redesign the architecture to use reliability patterns, such as deployment stamps with load balancers, so that each stamp can handle the load and work together with nearby resources.
Single-region workload failure with a cold failover site.	Pre-configure network settings in the failover region. This approach ensures that no IP addresses overlap.
Single-region application failure in a virtual network that communicates with a database via Azure Private Link by using a cold failover site.	Replicate connections in the secondary region and peer virtual networks for communication.

- **Overprovision IP address spaces**. To help ensure reliable scaling, a common strategy is to overprovision capacity to prevent IP-address exhaustion. However, this approach has a trade-off between reliability and operational efficiency. Subnets should only use a portion of the virtual network's address space. The goal should be to have only enough extra address space in your virtual network and subnets to balance reliability with operational efficiency.

- **Be aware of the networking limits**. Azure imposes limits on the number of resources that you can deploy. While most Azure networking limits are set to their maximum values, you can increase some limits. For more information, see Azure Resource Manager networking limits.

- **Create network diagrams that focus on user flows**. These diagrams can help you visualize network segmentation, identify potential points of failure, and pinpoint key transitions like internet ingress and egress points. They're also important tools for audits and incident response.

Highlight high-priority traffic flows between the user and workload resources. For example, if you prioritize Azure ExpressRoute for enterprise network flows or secure user requests in a perimeter network design, you can gain insights into capacity planning for firewalls and other services.

- ✓ **Add redundancy**. Consider deploying NAT gateways and virtual networks across multiple regions if needed. Ensure that public IP addresses and other availability zone-aware services have zone redundancy enabled, and make shared resources like firewalls region redundant.

  For more information, see Virtual Network business continuity.

- ✓ **Avoid complexity**. Pay close attention to virtual networks, subnets, IP addresses, routes, application security groups (ASGs), and tags. Simple configurations decrease the likelihood of misconfigurations and error. Misconfigurations and errors contribute to reliability problems and add to operational and maintenance costs. Some examples of simplification include:
  - Use private DNS when possible and minimize the number of DNS zones.
  - Simplify routing configurations. Consider routing all traffic through the firewall, if it's used in the architecture.

- ✓ **Test the resiliency of the network**. Use Azure Chaos Studio to simulate network connectivity disruptions. This approach ensures that your workloads remain redundant and helps you assesses the effect of potential failures.

- ✓ **Monitor network traffic for reliability effects**. Traffic flow monitoring is a crucial operation for reliability. For example, you want to identify high-volume communicators in your network to determine whether they can cause disruptions. Azure provides flow logging capabilities. For more information, see Operational Excellence.

Recommendations

Expand table

Recommendation	Benefit
Size virtual networks and subnets according to your scaling strategy. Choose fewer, larger virtual networks to accommodate redundancy as a mitigation strategy for failures.	By overprovisioning, you can ensure that the network scales efficiently without experiencing address-space limitations.
Ensure that there isn't overlapping address space with other virtual networks that you need to communicate	Plan the address space in advance to

Recommendation	Benefit
with, and plan the address space in advance. For more information, see Create, change, or delete a virtual network.	help prevent conflicts and ensure a smooth, scalable network architecture.
Use the Standard IP SKU for better reliability support through availability zones. By default, public IP addresses are deployed across multiple zones unless restricted to one zone.	This SKU helps ensure that communication within a public IP address remains operational during zonal failures.

Security

The purpose of the Security pillar is to provide **confidentiality, integrity, and availability** guarantees to the workload.

The Security design principles provide a high-level design strategy for achieving those goals by applying approaches to the technical design of virtual network.

Design checklist

Start your design strategy based on the design review checklist for Security and identify vulnerabilities and controls to improve the security posture. Extend the strategy to include more approaches as needed.

- ✓ **Establish a security baseline**. Review the security baseline for Virtual Network and incorporate applicable measures in your baseline.
- ✓ **Keep your network perimeter up to date**. Security settings, such as NSGs, ASGs, and IP address ranges must be updated regularly. Outdated rules might not align with current network architecture or traffic patterns. This security gap can leave your network exposed to potential attacks by lowering restrictions on ingress and egress traffic.
- ✓ **Use segmentation to enhance security**. Use NSGs as L4 firewalls at the subnet level. Route all external traffic through an network virtual appliance, such as a firewall, by using user-defined routes for monitoring and management. Use fully qualified domain names (FQDN) to filter internet access.

  Secure platform as a service connectivity with private endpoints while blocking outbound connections.

- ✓ **Apply the principle of least privilege**. Configure role-based access control (RBAC) with a no-access mindset for network-related roles. Make sure that users are only able to modify settings as required by their job function.
- ✓ **Limit public IP addresses**. Use shared public IP addresses from services like Azure Front Door for better security and initial request checks. Managing a dedicated public IP address requires you to oversee its security, including port management and request validation. When possible, use private connectivity.

Recommendations

Expand table

Recommendation	Benefit
Use Virtual Network encryption.	By enforcing encrypted traffic, you can protect data in transit between Azure Virtual Machines and Azure Virtual Machine Scale Sets within the same virtual network. It also encrypts traffic between regionally and globally peered virtual networks.
Enable Virtual Network Verifier in Azure Virtual Network Manager. Use this feature in your preproduction environment to test connectivity between resources. This feature isn't recommended in production.	Make sure that the Azure resources within the network are reachable and not blocked by policies.
Enable Azure DDoS Protection for the virtual network. Alternatively, you can protect individual public IP addresses through Azure DDoS IP Protection. Review the security features provided in DDoS IP Protection and DDoS Network Protection and choose one that fits your requirements. For example, the DDoS Network Protection tier provides support from the Rapid Response Team when attacks occur. The DDoS IP Protection tier doesn't provide this support.	You're able to safeguard against distributed denial-of-service attacks.
Safeguard segments within a virtual network by using NSGs. When possible, use ASGs to define the rules.	Traffic that enters and leaves the network can be filtered based on IP address and port ranges. ASGs simplify management by abstracting the underlying IP address ranges.

Recommendation	Benefit
Use private endpoints to access Azure services over a private IP address within the virtual network. Another way to implement private networking is through service endpoints. These endpoints route traffic to a service over the Azure network backbone. If available for the service, choose private endpoints over service endpoints.	Private endpoints remove the need for public IP addresses, which reduces the attack surface.

Cost Optimization

Cost Optimization focuses on **detecting spend patterns, prioritizing investments in critical areas, and optimizing in others** to meet the organization's budget while meeting business requirements.

The Cost Optimization design principles provide a high-level design strategy for achieving those goals and making tradeoffs as necessary in the technical design related to your networking environment.

Design checklist

Start your design strategy based on the design review checklist for Cost Optimization for investments. Fine-tune the design so that the workload is aligned with the budget that's allocated for the workload. Your design should use the right Azure capabilities, monitor investments, and find opportunities to optimize over time.

- ✓ **Optimize high-volume data transfers between endpoints**. Use virtual network peering to efficiently move data between virtual networks. Even though peering has inbound and outbound costs, this approach can be cost-effective because it reduces bandwidth consumption and network performance problems. Avoid routing through a hub to minimize inefficiencies and costs.

  To optimize data transfer between regions, it's important to consider both the frequency and the method of transfer. For example, when you deal with backups, the location where you save your backups can significantly affect costs. Storing backup data in a different region incurs bandwidth. To mitigate these costs, ensure that data is compressed before you transfer it across regions. You can further optimize costs and efficiency by adjusting the frequency of data transfers.

- ✓ **Include networking components in your cost model**. Account for hidden costs when you create or adjust your budget. For example, in multiregion architectures, it

costs extra to transfer data between regions.

Azure cost reports might not include the expenses that are associated with non-Microsoft NVAs, which have separate licensing costs. They might also have different billing models for fixed-price and consumption-based options. Make sure to include these factors in your budget considerations.

Some networking resources can be expensive, like Azure Firewall and ExpressRoute. You can provision these resources in centralized hub models, and allocate charges to teams for the incurred cost. Include that charge in your cost model.

- ✓ **Don't pay for unused capabilities**. Review component costs regularly and remove legacy features or default configurations. Limit the number of public IP addresses to save costs. This approach also enhances security by reducing the attack surface.

- ✓ **Optimize private endpoints**. Determine whether you can reuse a private link to a resource from other virtual networks. When you use a private endpoint across a regional virtual network peering, you're not charged peering fees for traffic to and from the private endpoint. You only pay for the private link access itself, not for the traffic between virtual networks. For more information, see Private Link in a hub-and-spoke network.

- ✓ **Align the network traffic inspection functions with the priority and security requirements of the flow**. For large bandwidth requirements, consider routing traffic to lower-cost paths. ExpressRoute is suitable for large traffic, but it can be expensive. Consider alternatives like public endpoints for cost savings. However, there's a trade-off on security. Use network peering for network-to-network traffic, to bypass the firewall, and to avoid unnecessary inspections.

  Only allow necessary traffic between components and block unexpected traffic. If traffic is expected and the flow is aligned with your security requirements, you can omit those checkpoints. For example, evaluate whether you need to route traffic through a firewall if the remote resource is within the trust boundary.

  Evaluate the number of subnets and their associated NSGs, even within a virtual network. The more NSGs you have, the higher the operational costs for managing the rule sets. When possible, use ASGs to streamline management and reduce costs.

- ✓ **Optimize code costs**. When you develop your application, choose more efficient protocols and apply data compression to optimize performance. For example, you can enhance efficiency in a web app by configuring components to compress data. These optimizations also affect performance.

- ✓ **Take advantage of resources in the centralized virtual network**. Use centralized resources to reduce duplication and overhead. Also, offloading responsibilities to existing teams can further help optimize costs and allow for delegation of expertise for specific functions.

Recommendations

Expand table

Recommendation	Benefit
Use virtual network peering to make network flow more efficient by bypassing controls. Avoid excessive peering.	Transfers data directly between peered virtual networks to bypass the firewall, which reduces bandwidth consumption and network performance problems. Avoid putting all your resources in a single virtual network. You might incur peering cost, but it's not practical to put all resources in a single virtual network only to save costs. It can hinder growth. The virtual network can eventually reach a point where new resources don't fit anymore.
Minimize public IP address resources if you don't need them. Before deletion, make sure that IP address isn't linked with any IP address configuration or virtual machine network interface.	Unnecessary public IPs can increase costs because of resource charges and operational overhead.

Operational Excellence

Operational Excellence primarily focuses on procedures for **development practices, observability, and release management**.

The Operational Excellence design principles provide a high-level design strategy for achieving those goals towards the operational requirements of the workload.

Design checklist

Start your design strategy based on the design review checklist for Operational Excellence for defining processes for observability, testing, and deployment related to your networking environment.

- ✓ **Learn new Azure networking constructs**. When you onboard to Azure, networking teams often assume that their existing knowledge is enough. However, Azure has many different aspects. Make sure that the team understands the fundamental Azure networking concepts, DNS complexities, routing, and security capabilities. Build a taxonomy of networking services so that the team can share knowledge and have a common understanding.

- ✓ **Formalize your network design and strive for simplicity**. Document the design and any changes, including configuration details like route tables, NSGs, and firewall rules. Include the governance policies that are in place, such as blocking ports. Clear documentation makes collaboration with other teams and stakeholders effective.

  Simplified networks are easier to monitor, troubleshoot, and maintain. For example, if you have a hub-and-spoke topology, minimize direct peerings between spokes to reduce operational burden and strengthen security. Always document the design, and provide justifications for each design decision.

  Reduce complexity by using aliases instead of direct IP address ranges. This method lowers operational burden.

- ✓ **Use design patterns that optimize network traffic**. To optimize network use and configuration, implement known design patterns that minimize or optimize network traffic. Validate the network configuration during builds by using security scanners to ensure that everything is set up correctly.

- ✓ **Do consistent network deployments**. Use infrastructure as code (IaC) for all components, including network peerings and private endpoints. Understand that core networking components are likely to change less frequently than other components. Implement a layered deployment approach for your stack so that you can deploy each layer independently. Avoid combining IaC with scripting to prevent complexity.

- ✓ **Monitor the networking stack**. Monitor the traffic patterns continuously to identify anomalies and problems, such as connection drops, before they cause cascading failures. When possible, set alerts to get notified about these disruptions.

  Similar to other components in this architecture, capture all relevant metrics and logs from various networking components, such as the virtual network, subnets, NSGs, firewalls, and load balancers. Aggregate, visualize, and analyze them in your dashboards. Create alerts for important events.

- ✓ **Include networking in your failure mitigation strategy**. Virtual networks and subnets are deployed initially and typically remain unchanged, which makes

rollbacks difficult. However, you can optimize your recovery by following a few strategies:

- ✓ **Duplicate networking infrastructure in advance, especially for hybrid setups**. Ensure that separate routes in different regions are ready to communicate with each other beforehand. Replicate and maintain consistent NSGs and Azure Firewall rules across both primary and disaster recovery (DR) sites. This process can be time consuming and requires approval, but doing it in advance helps prevent problems and failures. Make sure that you test the networking stack in the DR site.

- ✓ **Avoid overlapping IP address ranges between your production and DR networks**. By maintaining distinct IP address ranges, you can simplify network management and expedite the transition during a failover event.

  Consider the trade-offs between cost and reliability. For more information, see Trade-offs.

- ✓ **Offload network operations to central teams**. Centralize management and governance of networking infrastructure, when possible. For example, in a hub-spoke topology, services like Azure Firewall and ExpressRoute, DNS that are intended for shared use are placed in the hub network. That networking stack should be centrally managed, which takes the burden off the workload team.

  Offload the administration of the virtual network to the central team, even in the spoke network. Minimize network operations to what is pertinent to the workload, such as the management of NSGs.

  Keep the central teams informed of any necessary changes in the workload that might affect the configuration of shared resources. Aliases abstract the underlying IP addresses, which simplifies operations.

- ✓ **Rightsize your virtual network and subnets**. Choose fewer, larger virtual networks to reduce management overhead and avoid making subnets excessively large. The management of subnets and their NSGs can add to operational burden. For environments that have limited private IP addresses (RFC 1918) availability, consider using IPv6.

Expand table

Recommendation	Benefit
Deploy Virtual Network Manager.	Instead of configuring each virtual network individually, Virtual Network Manager centrally

Recommendation	Benefit
	manages connectivity based on rules. This approach streamlines networking operations.
Use networking monitoring tools. Regularly use virtual network flow logs and traffic analytics to identify changes in demand and patterns. Use the connection monitor feature to analyze and identify problems like connection drops before they affect applications.	You're able to understand how data flows through your network, identify bottlenecks, and identify unusual or unauthorized access attempts.
When you define routes, use service tags instead of specific IP addresses. Similarly, use ASGs when you define traffic rules for NSGs.	This approach ensures reliability because IP addresses can change but the configuration doesn't need to. Also, it helps overcome limits on the number of routes or rules that you can set by using more generic names.

Performance Efficiency

Performance Efficiency is about **maintaining user experience even when there's an increase in load** by managing capacity. The strategy includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing for peak performance.

The Performance Efficiency design principles provide a high-level design strategy for achieving those capacity goals against the expected usage.

Design checklist

Start your design strategy based on the design review checklist for Performance Efficiency for defining a baseline based on the key performance indicators.

- ✓ **Define performance targets**. To define performance targets, rely on monitoring metrics, specifically for latency and bandwidth. Use connection monitor data, such as latency and the number of hops, to set targets and thresholds for acceptable performance. Application Insights provides a detailed view of the time that workload requests spend in the network, which helps to refine these targets.
- ✓ **Rightsize your subnets**. When you allocate subnets, it's important to balance size and scalability. You want subnets to be large enough to accommodate projected growth without operational burden.

To manage capacity effectively, a common strategy is to overprovision capacity because of uncertainty, but the goal should be to optimize over time. Continuously analyze data so that your network can handle the load but you don't pay extra for unused resources.

- ✓ **Conduct performance testing**. Use a combination of synthetic and production data to test latency and bandwidth. This approach helps assess how these factors might affect workload performance. For example, it can detect resources that cause noisy neighbor problems and consume more bandwidth than expected. Also, it identifies traffic that makes multiple hops and causes high latency.

  We recommend that you test in production or capture and replay production data as test data. This approach ensures that tests reflect actual usage, which helps you set realistic performance targets.

- ✓ **Monitor traffic across regions**. It's important to consider that workload resources might be located in different regions. Communication across regions can add significant latency. Traffic across availability zones in the same region has low latency, but it might not be fast enough for some specialized workloads.

Recommendations

Expand table

Recommendation	Benefit
Enable the Azure Network Watcher connection monitor. Use the connection monitor during testing, which can generate synthetic traffic.	You're able to collect metrics that indicate loss and latency across networks. Also, you can trace the entire traffic path, which is important for detecting network bottlenecks.
Keep the virtual network address space large enough to support scaling.	You're able to accommodate projected growth without operational burden.

Trade-offs

You might have to make design trade-offs if you use the approaches in the pillar checklists. The following examples highlight the advantages and drawbacks.

Redundant networking stack

When you choose to implement a redundant networking stack, including its NSGs, routes, and other configurations, there's added cost of the infrastructure and thorough testing.

This upfront investment enhances reliability. You can be sure that everything works as expected and speed up recovery during disruptions.

Virtual network peering

Direct virtual network peering enhances performance by reducing latency because it avoids the need to route traffic through a hub where a firewall decrypts, inspects, and re-encrypts the payload.

That performance gain comes at the cost of decreased security. Without the firewall inspections that hub routing provides, the workload is more vulnerable to potential threats.

Large subnets

Large subnets provide ample address space, which allows workloads to scale out seamlessly. A large address space can safeguard against unexpected spikes in demand. However, it might lead to inefficient use of IP addresses. Over time, this inefficiency can potentially cause IP address exhaustion as the workload evolves. Also, this strategy comes with higher operational costs. From an Operational Excellence perspective, it's ideal to keep your subnets as small as possible.

Azure policies

Azure provides an extensive set of built-in policies related to Virtual Network and its dependencies. Define and assign policies to help ensure that resources comply with organizational standards. Create an Azure Policy Compliance Dashboard to identify noncompliant resources and take corrective actions.

A set of Azure policies can audit some of the preceding recommendations. For example, you can set policies that automatically:

- Protect the virtual network against volumetric and protocol attacks.
- Deny the creation of network interfaces that have public IP addresses.
- Deploy Network Watcher for virtual networks.
- Enable traffic analytics and flow logs to monitor traffic patterns.

For comprehensive governance, review the Azure Policy built-in definitions and other policies that might affect the security of the network layer.

Azure Advisor recommendations

Azure Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments. Here are some recommendations that can help you improve the reliability, security, cost effectiveness, performance, and operational excellence of your virtual network.

- Reliability
- Security
- Cost Optimization
- Performance
- Operational Excellence

Next steps

The following articles demonstrate the recommendations that are discussed in this article.

- For a hub-spoke topology, use the hub-spoke network topology reference architecture to set up your initial stamp.
- Use the following product documentation to enhance your implementation expertise:
  - Virtual Network
  - Virtual Network IP Services documentation

Feedback

Was this page helpful? Yes No

Health modeling for workloads

Article • 10/23/2024

Cloud applications generate high volumes of operational data, which makes it challenging to pinpoint and resolve problems quickly. A common reason for this challenge is the absence of a health baseline that's customized to the workload's functionality and the inability to detect drift from that baseline.

Health modeling is an observability exercise that combines business context with raw monitoring data to quantify the overall health of a workload. It helps set a baseline that you can monitor the workload against. You should consider data like telemetry from infrastructure and application components. Health modeling might also incorporate other information that's necessary to achieve the workload's quality targets.

Performance problems or operational degradation can cause drift from the expected operational state. By modeling the health of a workload, you can identify drift and make informed operational decisions that consider business impact.

Health modeling bridges the gap between tribal operational knowledge and actionable insights. It helps you manage critical issues effectively. The concept is essential to maximize reliability and operational effectiveness.

This guide offers practical guidance about health modeling, including how to build a model that assesses the runtime health of a workload and all of its subsystems.

Expand table

Terminology	Definition
Health modeling	An observability exercise that uses business context to interpret monitoring data as health states.
Health model	A graphical representation of logical entities and their relationships for a given scope. Each node has a health state definition to rationalize monitoring data across the model.
Health entity	A logical component that represents an individual unit of a system, a logical combination of multiple related entities, or the overall system.
Health state	A defined and measurable status that provides meaningful operational insights about the health of an entity.
Health signal	Individual data streams that provide insights into the operational behavior of an entity.

Terminology	Definition
Model of models	An aggregated modeling scope in which entities represent distinct health models for component systems.

We recommend that you watch this video to get a high-level understanding of health modeling.
https://learn-video.azurefd.net/vod/player?id=fd8c4e50-9d7f-4df0-97cb-d0474b581398&embedUrl=%2Fazure%2Fwell-architected%2Fdesign-guides%2Fhealth-modeling&locale=en-us

What is health, health modeling, and a health model?

The term *health* refers to the operational status of an entity and its dependencies. That entity can be an individual unit of a system, a logical combination of multiple related entities, or the overall system.

We recommend that you represent health in one of three states:

- **Healthy**: Operates optimally and meets quality expectations
- **Degraded**: Exhibits less than healthy behavior, which indicates potential problems
- **Unhealthy**: In a critical state and requires immediate attention

> **Note**
>
> You can represent health with a score instead of states to provide more data granularity.

Health states are derived by combining monitoring data with domain information. Each state must be defined and must be measurable. Health states are calculated by using *health signals*, which are individual data streams that provide insights into an entity's operational behavior. Signals can include metrics, logs, traces, or other quality characteristics. For example, a health signal for a virtual machine (VM) entity might track the CPU utilization metric. Other signals for this entity can include memory usage, network latency, or error rates.

As you define health signals, factor in the nonfunctional requirements for the workload. In the example of CPU utilization, include the expected thresholds for each health state. If utilization exceeds the tolerated threshold in accordance with the workload

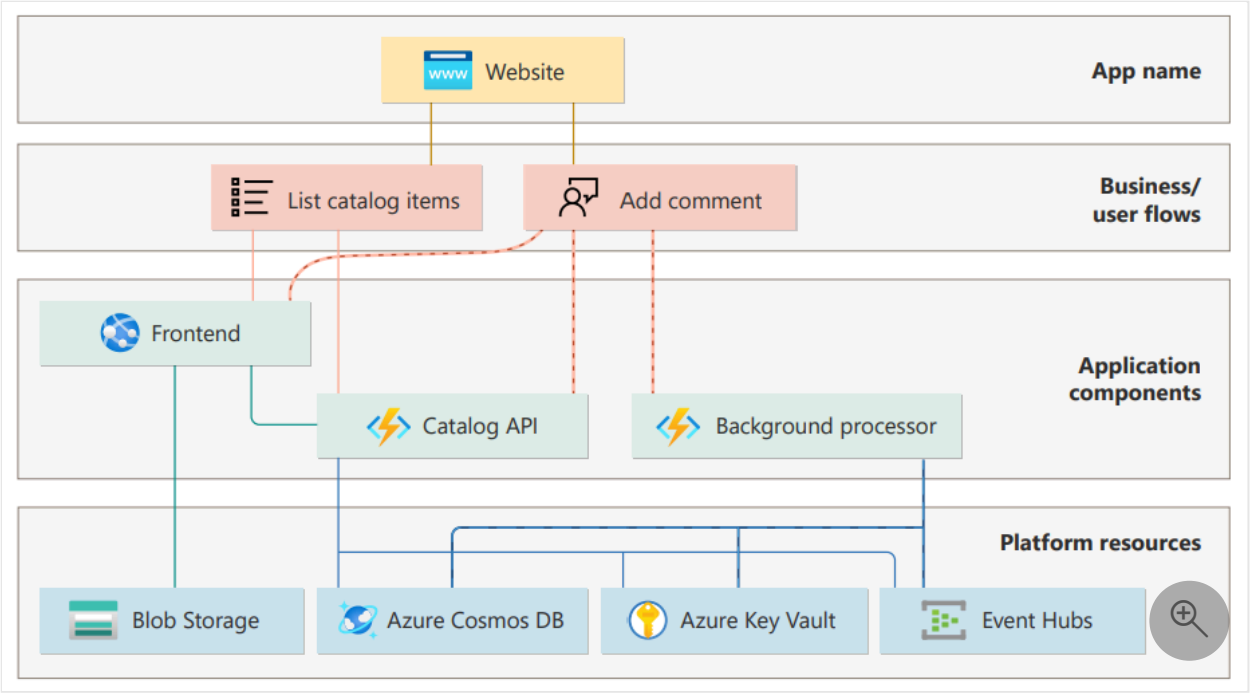
requirements, the system transitions from *Healthy* to *Degraded* or *Unhealthy*. These state changes trigger the appropriate alerts or actions.

Health modeling requires entities to have well-defined states that are derived from multiple health signals and are contextualized for the workload. For example, the health definition for a VM might be:

- **Healthy**: Key nonfunctional requirements and targets, such as response time, resource utilization, and overall system performance, are fully satisfied. For example, 95% of requests are processed within 500 milliseconds. The workload uses VM resources like CPU, memory, and storage optimally and maintains a balance between workload demands and available capacity. User experience is at expected levels.
- **Degraded**: Resources aren't performing optimally but are still operational. For example, the storage disk is experiencing throttling problems. Users might experience slow responses.
- **Unhealthy**: Degradation is beyond the tolerated limits. Resources are no longer responsive or available, and the system is no longer meeting acceptable performance levels. User experience is severely affected.

The outcome of health modeling is a *model* or a graphical representation of logical entities and their relationships for a workload architecture. Each node has a health state definition.

> **ⓘ Important**
>
> *Health modeling* is an abstract concept that you can implement and apply at different scopes if you have a good understanding of the business scenarios.



In the image:

- *Entities* are logical components of the workload that represent aspects of the system. They can be infrastructure components, like servers, databases, and networks. They can also be specific application modules, pods, services, or microservices. Or, entities can capture user interactions and system flows within the workload.

  > **Note**
  >
  > User and system flows summarize nonfunctional requirements across business scenarios that involve application and infrastructure components. This summary reflects business value for the application.

- *Relationships* between entities mirror the dependency chains within the system. For example, an application module might call specific infrastructure components that form a *relationship*.

Consider a scenario in which an e-commerce workload experiences a spike in failed messages on an Azure Service Bus queue, which is causing payments to fail. This problem is critical for the organization due to the implied revenue loss. Although an application developer might understand the effect of this metric spike on payments, this tribal knowledge isn't often shared across the operations team.

A health model can give operators immediate visibility into the problem and its effects. The payment flow depends on Service Bus, which is one of the workload components. The visual representation reveals the degraded state of the Service Bus instance and its

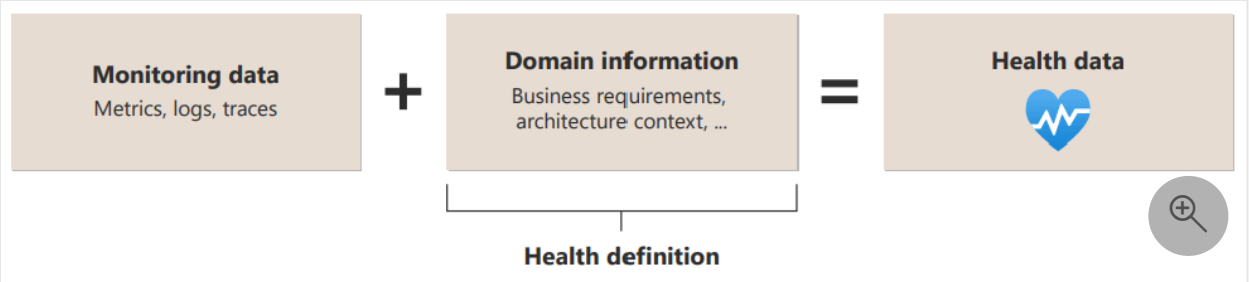
effect on the payment flow. Operators can understand the importance of the issue and focus their remediation efforts on that specific component.

Health modeling was important in the preceding scenario in the following ways:

- It improved the time to detect (TTD) and time to mitigate (TTM) by enabling faster problem isolation, which led to quicker detection of problems and potential fixes.
- Operators received alerts based on health states, which reduced unnecessary noise. Operators received notifications that provided specific context about the business impact on payments.
- Dependency chains helped operators fully understand the extent of operational issues. This knowledge accelerated impact assessments and led to prioritized responses. Operators also easily identified cascading or correlated issues.
- Operators conducted post-incident activities with accuracy because the health model provided insights into the root causes of anomalies and the specific health signals that were involved.
- It made the monitoring data meaningful for all team members. It bridged the gap between tribal knowledge and shared insights.
- The organization used the health model as a baseline for future investments in AI-driven operations to derive intelligent insights.

Health model schema

Health models provide a distinct data schema optimized for observability use cases. This schema takes health modeling from an abstract concept to a measurable solution. By modeling your specific requirements, objectives, and architectural context, you can tailor health data to your unique scenario.



Health is a relative data concept. Each model represents health data that's unique and prioritized for its contextual scope, even if it uses the same set of entities. What constitutes *healthy* in a specific scenario might differ significantly in other contexts.

For example, consider Azure resources of the same type within your workload.

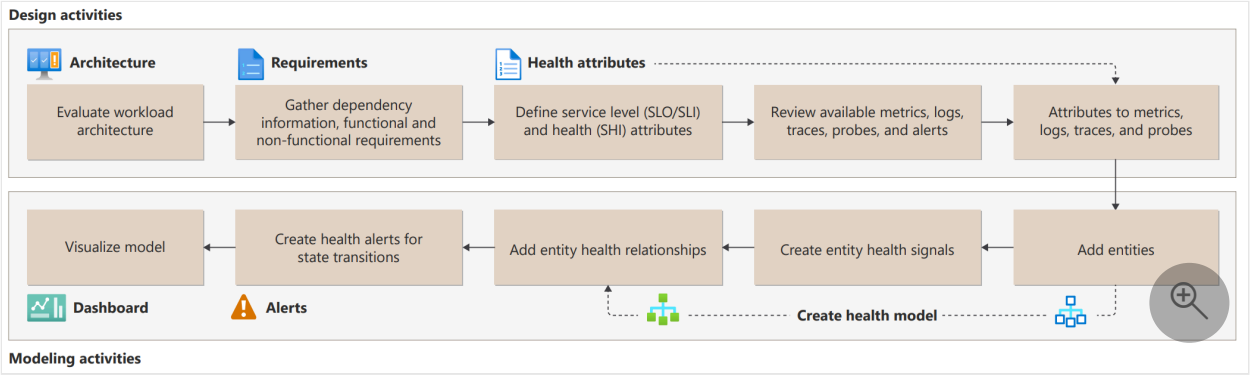
- VM A runs a CPU-sensitive application.
- VM B handles a memory-intensive service.

The health definitions for these machines are different. CPU utilization metrics likely influence VM A's health status, and VM B might prioritize memory-related metrics.

> **Important**
>
> A health model shouldn't treat all failures the same. It should clearly distinguish between expected or transient but recoverable failures and a true disaster state.

Build a health model

The first step to build a health model is a logical design exercise, which typically involves the activities that are described in the following sections.



Evaluate your workload design

Begin this logical design exercise by evaluating the following components of your workload design.

- Infrastructure components like compute clusters and databases
- Application components that run on compute and their relevant components
- Logical or physical dependencies between components
- User and system flows

For example, the health model for an e-commerce application should represent the current state of critical processes like user sign-in, checkout, and payments.

Contextualize using business requirements

Evaluate the relative importance and overall impact of each flow on your organization. Consider factors like user experience, security, and operational efficiency. For example, in most scenarios, the failure of a payment process is likely more significant than the failure of a reporting process.

Identify escalation paths for handling problems related to each flow. For more information, see Optimize workload design using flows.

> **Note**
>
> You realize the value of health modeling only when you incorporate your business scenarios and context. Then you can rationalize the business impact from operational issues.

Map to reliability metrics

Look for relevant reliability metrics across the application design.

Consider defining service-level indicators (SLIs) and service-level objectives (SLOs) for the entire application and its individual business processes. These SLIs and SLOs should align with the specific health signals considered for your health model. By doing so, you create a comprehensive definition of health that accurately reflects the achievement of an acceptable service level for the application.

> **Important**
>
> SLIs and SLOs are critical health signals. They create a meaningful definition of health that reflects the level of service that you want along with other quality attributes. You can also define service health objectives (SHOs) to capture the health that you want to attain over an aggregated time range.

Identify health signals

To build a comprehensive health model, correlate various types of monitoring data, including metrics, logs, and traces. By doing so, you ensure that the concept of health accurately reflects the runtime state of a specific entity or the entire workload.

Use platform metrics and logs

In the context of health modeling, it's essential to gather platform-level metrics and logs from underlying Azure resources. These metrics include CPU percentage, network in and network out, and disk operations per second. You can use this data in your health model to detect and predict potential problems while maintaining a reliable environment.

Furthermore, this approach helps you differentiate between transient faults, or temporary disruptions, and nontransient faults, or persistent problems.

> ⓘ **Note**
>
> As a best practice, you should configure all application resources to direct diagnostic logs and metrics to the chosen log aggregation technology. Build guardrails by using Azure Policy to ensure consistent diagnostic settings across the application and enforce the chosen configuration for each Azure service.

Add application logs

Application logs are an important source of diagnostics data for your health model. Here are some best practices for application logging:

- **Use semantic or structured logging**. Structured logs facilitate automated consumption and analysis of log data at scale.

  Consider storing Azure resource metrics and diagnostics data in an Azure Monitor Logs workspace instead of a storage account. By using this method, you can create health signals by using Kusto queries for efficient evaluation.

- **Log data in the production environment**. Capture comprehensive data while the application operates in the production environment. Sufficient information is essential for health assessment and to diagnose any detected production problems.

- **Log events at service boundaries**. Include a correlation ID that traverses service boundaries. If a transaction involves multiple services and one of them fails, the correlation ID helps you track requests throughout your application and pinpoint the cause of failure.

- **Use asynchronous logging**. Avoid synchronous logging operations that might block application code. Asynchronous logging ensures availability by preventing request backlogs during log writes.

- **Separate application logging from auditing**. Maintain audit logs separately from diagnostic logs. Although audit records serve compliance or regulatory

requirements, keeping them distinct prevents dropped transactions.

Implement distributed tracing

Implement distributed tracing by correlating telemetry across critical system flows. Correlated telemetry provides insights into end-to-end transactions and is essential for effective root cause analysis (RCA) when failures occur.

Use health probes

Implement and run health probes outside of the application to explicitly check the health and responsiveness of your application. Use probe responses as signals within your health model.

You can implement health probes by measuring the response time from the application as a whole or from its individual components. Probes can run processes to measure latency and check availability or to extract information from the application. For more information, see Health Endpoint Monitoring pattern.

Most load balancers support running health probes that ping application endpoints at configured intervals. Alternatively, you can use an external watchdog service. A watchdog service aggregates health checks from across multiple components in the workload. Watchdogs can also host code that does immediate remediation for known health conditions.

Adopt structural and functional monitoring techniques

Structural monitoring involves equipping the application with semantic logs and metrics. The application directly collects these metrics, which include current memory consumption, request latency, and other relevant application-level data.

Strengthen your monitoring processes by using functional monitoring. This approach focuses on measuring platform services and their effect on the overall user experience. Unlike structural monitoring, functional monitoring doesn't require detailed knowledge of the system. It tests the externally visible behavior of the application. This approach is useful for assessing SLOs and SLIs.

Model the design

Represent the identified application design as entities and relationships. Map health signals to specific components to quantify health states at an entity level. Consider the

criticality of components to determine how health states should propagate through the model. For example, reporting components might not be as critical as other components, which results in different effects on overall workload health.

Set actionable alerts

Use the evaluated health states to trigger alerts and automated action. Health should be integrated within existing operational runbooks as a core observability data tenet.

Typically, there's a one-to-one mapping between monitoring data and alert rules, which can lead to undesirable outcomes, like alert storms and ambient alert noise. For example, in a compute cluster, high volumes of VM-level alerts based on CPU utilization and error count can overwhelm operators during failures and cause delays in resolution. Similarly, when there's a high number of configured alerts, ambient alert noise often results in alerts that are overlooked or ignored.

A health model introduces separation between monitoring data and alert rules. A health definition aggregates many signals into a single health state, which decreases the number of alerts so that operators can focus solely on high-value alerts that are critical for the organization. Consider the e-commerce scenario. You can set up an alert to send notifications about changes in the process payments flow health instead of changes in underlying resources like the Service Bus queue.

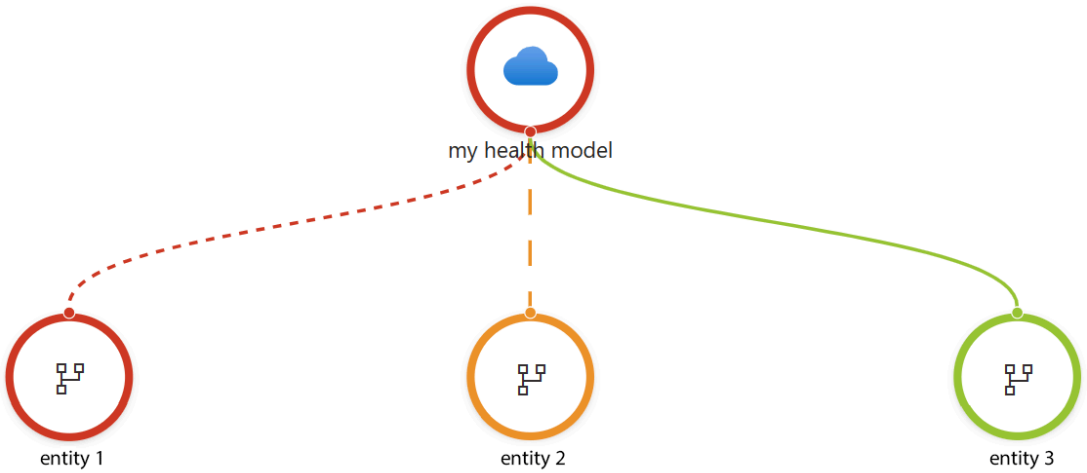
> **Note**
>
> The ability to alert across all layers of the health model provides flexibility for the different workload personas. Application owners and product managers can be alerted to health state changes in key business scenarios or in the entire workload. Operators can be alerted based on the health of infrastructure or application components.

Visualize the model

Create visual representations, such as tables or graphs, to effectively convey the current state and history of the health model. Ensure that the visualization aligns with the business context and provides actionable insights.

When you visualize your health model, consider adopting a *traffic light* approach to make health states immediately insightful across dependency chains.

Assign green for healthy, amber for degraded, and red for unhealthy. By quickly identifying the color-coded states, you can efficiently locate the root cause of any application degradation.



> **Note**
>
> We recommend that you consider accessibility requirements for people who have a vision disability when you create a dashboard for your health model. For diagramming best practices, see Architecture design diagrams.

Adopt your health model

After you build a health model, consider the following use cases to drive detection and interpretation of failures or operational problems.

Applicability to various roles

Health modeling can provide information that's specific to job functions or to roles within the same context of the workload. For example, a DevOps role might need operational health information. A security officer might be more concerned about intrusion signals and security exposure. A database administrator is likely only interested in a subset of the application model through the database resources.

Tailor health insights for different stakeholders. Consider creating separate models from overlapping data sets.

Continuous validation

Use your health model to optimize testing and validation processes, such as load testing and chaos testing. You can validate the runtime operational state during testing and assess your model's effectiveness in scale and failure scenarios by incorporating health models into your engineering lifecycle.

Organizational health

Although health modeling is commonly associated with quantifying health states for individual applications, its applicability extends beyond that scope.

At an individual workload level, health models provide a foundation for application observability and operational insights. Each application can have its own health model that captures what each health state means within its context.

You can combine multiple health models into a high-level construct by building a *model of models*. For example, you can build the observability footprint of a business unit or an entire cloud estate by using health models as components within a larger model. Health models represent workloads within the estate as nodes within the top-level graph. Use the relationships in this model to capture inter-application dependencies, including data flows, service interactions, and shared infrastructure.

Consider a retail company that has various applications for e-commerce, payments, and order processing. You can define each of these applications as an independent health model to quantify what health means for that workload. You can then use a parent model to map all of these component health models as entities and capture inter-application operational impact through dependency chains. For example, if the e-commerce application becomes unhealthy, it has a cascading effect on the payment application.

Health trends and AI for IT operations

Health modeling provides a quantified operational baseline that's tuned to a specific business context. AI for IT operations (AIOps) is a popular way to enhance operational efficiency. Health data is a foundational input for machine learning models to analyze health trends. For example, machine learning models can:

- Extract more insights from state changes and recommend actions.
- Analyze health trends over time to drive issue prediction and model refinement.

Maintain your health model

Maintaining a heath model is a continuous engineering activity that aligns with your application's development and operations. As your application evolves, make sure that your health model evolves in parallel.

Also, treat health models like workload artifacts that should be integrated into your development lifecycle. Adopt infrastructure as code (IaC) for consistent, version-controlled management of your health model. Use automation so that the model stays up to date as you add or remove infrastructure and application components from the workload.

Health data gradually diminishes in value over time. To optimize operational efficiency and minimize costs, avoid retaining health data beyond 30 days. If necessary, you can archive data to satisfy audit requirements or in scenarios that involve long-term pattern analysis in AI for IT operations.

> **Note**
>
> When you archive health data, make sure you couple it with the configuration state of the model. Interpreting state changes can be challenging without this context.

Related links

- For implementing health probes in ASP.NET, see Health checks in ASP.NET Core.
- For information on monitoring metrics, see Azure Monitor Metrics overview.
- For information on using Application Insights, see Application Insights.
- For design considerations and recommendations that pertain to mission-critical workloads, see Health modeling and observability for mission-critical workloads on Azure.
- For a hands-on experience, see Design a health model for your mission critical workload.

Next step

Recommendations for designing a reliable monitoring and alerting strategy

Feedback

Was this page helpful?


Yes
No

Complete an Azure Well-Architected Review assessment

Article • 10/29/2024

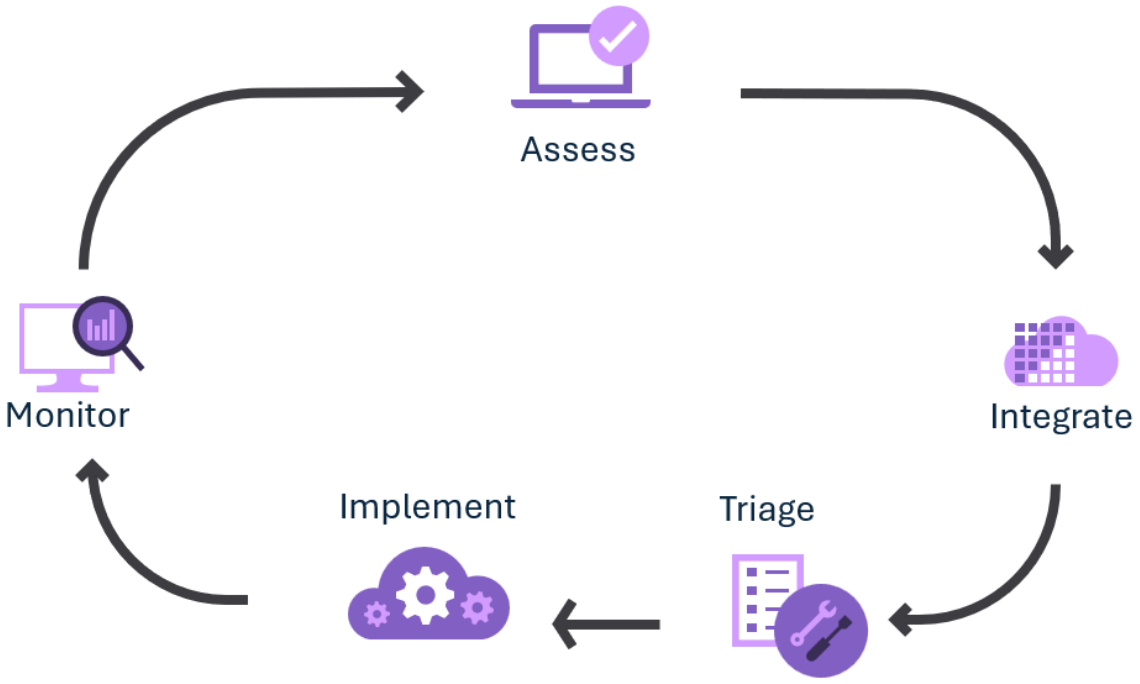
Azure Well-Architected Review is a self-assessment that can help a workload team examine a workload from the perspective of the Azure Well-Architected Framework. It consists of approximately 60 questions that are based on the key recommendations provided in the pillars of the Well-Architected Framework. The assessment tool can also pull in Azure Advisor recommendations for an Azure subscription or resource group.

Before initiating the assessment, it is essential to prioritize the pillars according to your specific business needs. This approach will allow you to allocate your efforts more effectively and efficiently. At the end of the assessment, you get recommendations and corresponding links to supporting material that can help you improve your workload's design. You can export these recommendations into a file that you can use to incorporate the recommendations into the operational processes for continuous workload improvement.

When to take the assessment

For greenfield workloads, we recommend that you perform the assessment during the initial design process, entering the proposed decisions. The guidance then acts as a baseline and starts a feedback loop that you can use to refine the workload design as you make additional design decisions and periodically capture them in additional assessment milestones.

Brownfield workloads should be examined as well, as part of the continuous improvement cycle of the workload. Set a cadence, for example every four months, and use milestones to track how the workload design can continue to improve.



Receive and integrate recommendations

Assess your workload by completing the assessment. The recommendations for your current milestone are available on the assessment's guidance page. Export these recommendations by selecting the **Export to CSV** button. You can use the offline copy to share the recommendations and start to prioritize them. Although some teams might consider the CSV file sufficient, we recommend that you add the recommendations to the workload's backlog so they can be integrated into the workload's software development lifecycle (SDLC).

> **Tip**
>
> DevOps Tooling for Well-Architected Recommendation Process provides example scripts that can help you create automation for backlog integration. These scripts show one way to import the recommendations from the Well-Architected Review CSV file into an existing Azure DevOps or GitHub organization.

Prioritize and implement recommendations

Workload owners and key stakeholders should prioritize the recommendations in accordance with the team's standard work prioritization process, factoring in the applicability of the recommendations and any tradeoffs associated with a specific design decision. For example, recommendations might be assigned to a specific owner, or a

recommendation might be postponed or dismissed. Like all planned work, the recommendation should be tracked until it's resolved, as part of the workload's SDLC.

Monitor improvements

Over time, the workload will evolve due to functionality changes, eliminating or accruing technical debt, and making tradeoffs. Use the milestone feature of the assessment to track this change over time, using the prior milestone as a baseline. You'll see the change over time in the Azure Well-Architected Review. The workload's component of the subscription's Azure Advisor score will probably improve as well.

Tips

- The assessment is a collaborative effort that requires the active and honest participation of multiple experts and stakeholders within a workload. It is essential that all participants feel comfortable discussing shortcomings openly, without fear of repercussions, so the team avoids obscuring workload risks or misses opportunities for improvement.
- You should always sign in when you take assessments so that the tool can generate milestones.

  > ⚠ **Warning**
  >
  > Assessments are tied to a Microsoft Learn profile. They can't be transferred to or accessed by other profiles.

- Select the Azure subscription or resource group that contains the biggest portion of your workload. Doing so helps ensure that only relevant Advisor recommendations are included in exported CSV files. It's not possible include more than one subscription or to exclude resource groups.
- Choose a meaningful name for the assessment, not the default value. The assessment's name should include the workload's name.
- Use meaningful milestone names to indicate when you're evaluating the workload.
- Use the notes feature on questions and on recommendations to capture any specifics that you want to discuss with the workload team.
- Rather than answering the 60 questions across all five pillars in one assessment, consider taking the assessment one pillar at a time, staggered by month. Be sure

to include the name of the pillar in the assessment's name.

Get personalized support

Work with your Microsoft partner or your account team to learn how they can help you perform an assessment as a formal engagement. As part of that engagement, they can provide further details on the recommendations. These details can help you determine the applicability of recommendations and how to prioritize them for remediation.

Next step

Complete an Azure Well-Architected Review

Feedback

Was this page helpful? Yes No

Optimize workload design using flows

Article • 10/23/2024

This article covers the targeted optimization of workloads using flows. Different components of a workload have varying requirements and levels of importance. By segmenting a workload into flows, you can prioritize different parts of a workload and better align workload investments with the importance of each flow.

This workload optimization process is iterative and involves three key steps: (1) define the flow structure within your workload, (2) define technical requirements, and (3) design the flow to meet the requirements (*see figure 1*).



Figure 1: The process to optimize workloads using flows.

Define the flow

Before you can define flow requirements, you need to understand the business drivers for the flow. The prerequisites to defining a flow are identifying the business process and use case its supports. When you understand the prerequisites, you can start documenting the flow.

Understand the prerequisites

Flows are sequences of actions that support workload functionality. There are two primary types of flows: user flows and system flows. User flows determine user interactions. System flows determine communication between workload components. Flows support business processes and use cases. A workload consists of multiple use cases. You need to identify the business process and use case the flow supports before documenting a flow (*see figure 2*).

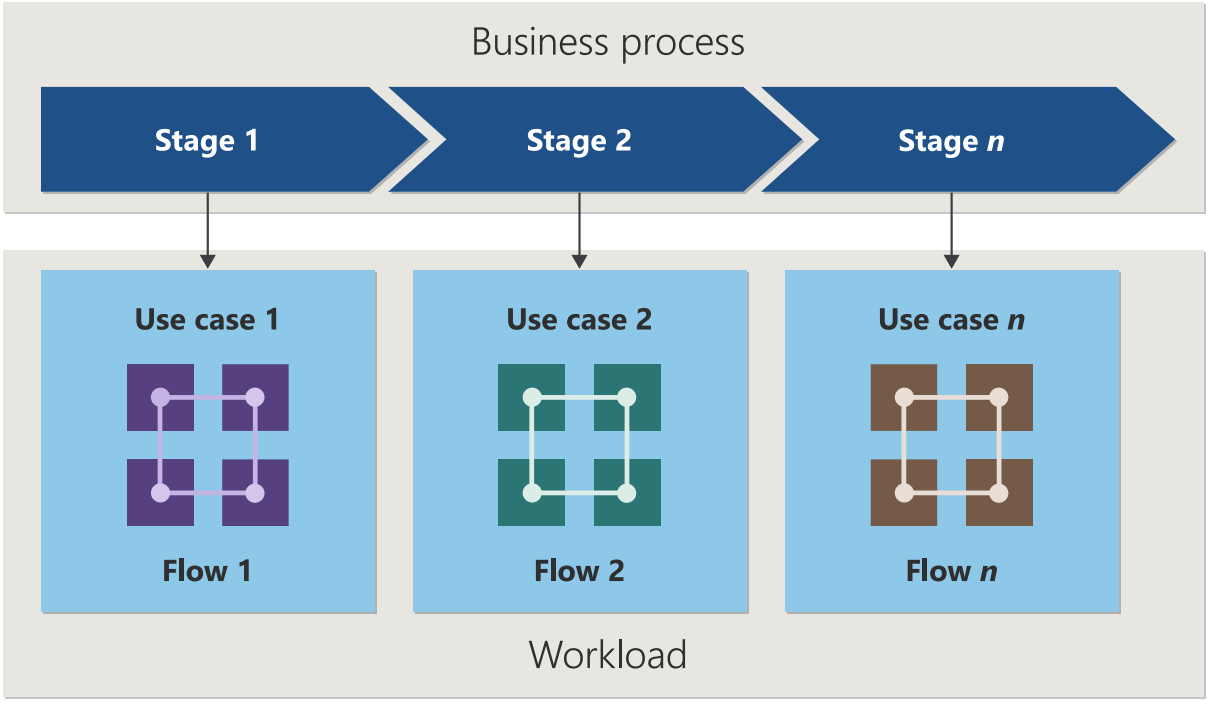


Figure 2: The relationship between business processes, use cases, flows, and workload.

Identify the business process

A business process is a series of actions (stages) that fulfill a business requirement. Flows determine the sequence a user or data takes to accomplish each stage of a business process. For example, selling products online is a business process. The stages in this business process might be listing the product online, receiving orders, and delivering the product.

Identify the use case

A use case defines the functional requirements of a flow. You need to identify and understand the use case a flow supports before establishing the technical requirements of a flow. Each use case should support one stage in a business process (*see figure 2*). A use case should define the following attributes:

- *Purpose*: Clearly articulate the tasks or objectives, like enabling online purchases. This clarity guides the functional design and sets clear goals for flow design.
- *Criticality*: Assess the importance of the use case, ranging from routine to critical. The value assigned to a use case informs the prioritization and design of the flow. High-value use cases might require enhanced error handling, performance tuning, or user experience considerations.
- *Consumers*: Identify whether users (customers, staff) or system components are the primary consumers. This categorization determines whether it's a user flow or

system flow and influences the design.

- *Events*: Define triggers or conditions that initiate and conclude the use case. These events define the flow's boundaries.
- *Execution*: Understand the operational frequency and variability of the use case to anticipate system load. You must design a flow to handle different execution scenarios.
- *Dependencies*: Identify interdependencies with other use cases for risk management. Recognizing a use case's dependencies aids in designing flows that integrate smoothly with other system parts. You need to ensure the availability of necessary inputs and compatibility of outputs with subsequent processes.

Document the flow

Use the use case to document the flow. You should outline or map each action you need in a flow. Capture decision criteria and pathways. Identify interactions with other use cases. This outline serves as a blueprint for flow design and management. You also need to capture business information about the flow. Make sure to include the following details in the flow documentation:

- *Flow description*: A high-level description of the flow.
- *Business process*: The business process the flow supports.
- *Process owner*: The individual that owns the business process.
- *Stakeholders*: The individuals that you should inform or consult on flow status or changes.
- *Escalation paths*: The individuals or groups you should contact to resolve issues. It's a sequence of people. The scope of individual responsibility grows with each person on the path.
- *Business impact*: The importance of this flow to the business.
- *Criticality rating*: A qualitative label that indicates the relative importance of the flow.

For more information, see Flow examples.

Define flow requirements

Utilize the use case to establish the technical targets of the flow. Define measurable targets for the flow that align to the five pillars of the Well-Architected Framework (WAF). These pillars provide a framework for setting technical targets:

- *Reliability targets*: Assess each flow's importance and set reliability targets accordingly. Determine performance thresholds and establish clear service level agreements (SLAs) and objectives (SLOs). Higher criticality flows require more stringent reliability targets.
- *Security targets*: Analyze the security needs of each flow based on data sensitivity and user activities. Implement and continuously update security measures to meet these needs while ensuring compliance with regulatory standards.
- *Cost targets*: Understand the demands of each flow for effective resource allocation. Set targets to balance cost with performance. Ensure resource usage aligns with business priorities.
- *Operational targets*: Define metrics for effective monitoring and troubleshooting. Targets should ensure efficient resource use and alignment with organizational goals.
- *Performance targets*: Base performance targets on the initial requirements of each flow. Ensure that essential flows receive adequate resources and continuously adjust targets to meet evolving demands and enhance user experiences.

Design the flow

Design the flow to meet the technical targets. You should familiarize yourself with flow design best practices so that you achieve the right result. Build and test the flow. Iterate on the design until it meets the technical targets you established.

Follow flow design best practices

As you design a flow, follow flow design best practices. A well-designed flow has the following attributes:

- *Scoped*: Identify distinct starting and ending points for each flow. Clear boundaries help optimize user or system interactions.
- *Logical:* Design your flows with a logical order of steps. Optimize for the most efficient path and reduce unnecessary steps.

- *Maintainable*: Design flows that are easy to update and maintain. Use modular components that you can modify without affecting the entire workload.
- *Defined*: Incorporate specific conditions that trigger or guide each step in a flow. This precision ensures that the flow responds accurately to user inputs, data changes, or system states.
- *Reliable*: Build error handling and exception paths into your flows. Effective error management prevents disruption and maintains flow integrity under unexpected circumstances.
- *Scalable*: Ensure it can handle varying loads and adapt to growing or shrinking user bases or data volumes.
- *Secure*: Embed security measures within the flow. Protect data and user interactions against unauthorized access and threats.
- *Efficient*: Plan for efficient use of resources without over-provisioning. Keep cost optimization in mind.
- *User-centric*: For user flows, align the flow design with user expectations and behaviors. Make it intuitive and reduce the learning curve for new users.

Develop and test the flow

Develop the flow to meet technical targets and test it to ensure it meets its requirements. This process validates that the flow operates as intended, efficiently handles its tasks, and meets the technical targets. Here's guidance to build and test a flow:

- *Select technologies*: Choose technologies that align with the set targets in terms of reliability, security, and performance.
- *Develop flow*: Build the flow according to the design, keeping the set targets in mind.
- *Test flow*: Conduct testing to ensure the flow meets targets. Iterate as needed to meet targets.
- *Monitor*: Implement monitoring tools to track resource usage and costs.

Periodically review the flow against set targets and industry standards. Use feedback from monitoring and audits to improve the flow. Adjust targets and processes as necessary to align with changing business needs or technological advancements.

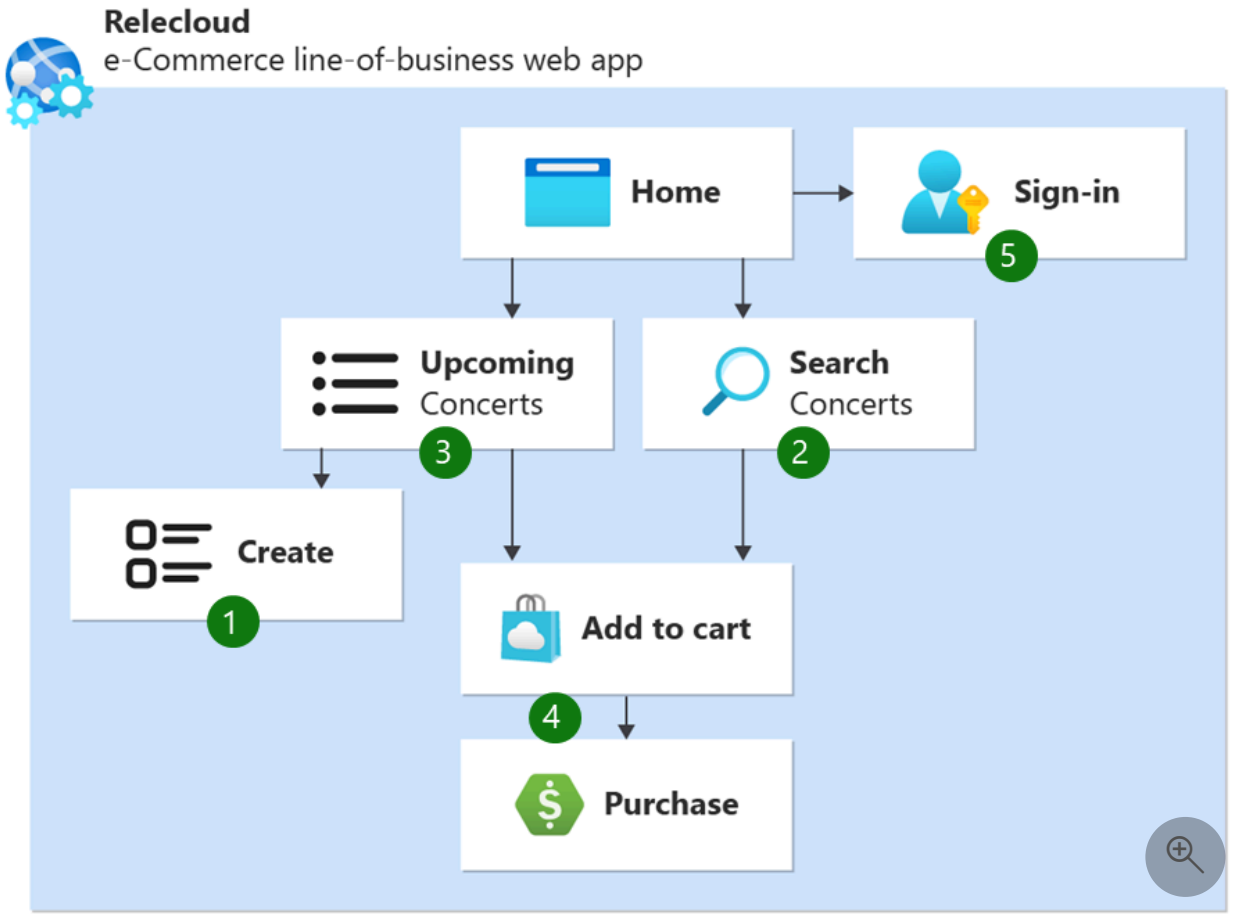
Optimize flows

Repeat the process defined in this article throughout the lifecycle of the flow. As you iterate on the flow design, use the Well-Architected Framework to optimize flows from the perspective of each pillar:

- Flow reliability
- Flow security
- Flow cost optimization
- Flow operational excellence
- Flow performance efficiency

Flow examples

Here are a few flow examples to help you design your flows. The examples use the reliable web app pattern reference architecture as the basis and shows the documentation you should have on each flow.



User flow 1: Create upcoming concerts

Flow description: Call-center employees use the application to create an upcoming concert.

- *Business processes*: This flow supports the *purchasing ticket* process, but it's asynchronous, lowering its criticality.
- *Process owner*: Director of Sales.
- *Stakeholders*: Sales department, concert planning and operations, platform team, and application team.
- *Escalation paths*: Application team, platform team, then sales department.
- *Business impact*: This flow is important for making new concerts available on sales platforms, directly influencing the main revenue stream of the business. When call-center employees are unable to create concerts due to the unavailability of this flow, it negatively impacts both revenue and the company's reputation. However, high availability isn't essential for this process since concerts are typically scheduled in advance on a weekly basis. The sales department specified a requirement of 95% availability for this process and is agreeable to downtime outside of business hours for maintenance purposes.
- *Criticality rating*: Low.

User flow 2: Search concerts

Flow description: Call-center employees use the application to search for upcoming concerts.

- *Business processes*: This flow supports the *purchasing ticket* process, but call-center employees can opt to list all concerts if the search function isn't available.
- *Process owner*: The user experience (UX) department.
- *Stakeholders*: Sales department, platform team, and application team.
- *Escalation path*: Application team, platform team, sales department manager on-call.
- *Business impact*: This flow allows call-center employees to quickly find concerts and is part of the normal sales process. High availability of this flow isn't critical since employees have the capability to list concerts even in its absence. It does degrade the call-center employee's experience might degrade and affect productivity. Customers could experience frustration due to increased wait times or delays. The

sales department requested a 99% availability of this flow during regular business hours.

- *Criticality rating*: Medium.

User flow 3: Get a list of the concerts

Flow description: Call-center employees use the application to get a list of concerts.

- *Business processes*: This flow directly supports the *purchasing ticket* process.
- *Process owner*: Director of Platform.
- *Stakeholders*: Sales department, platform team, data team.
- *Escalation path*: Data team, data team on-call engineer, platform team on-call engineer.
- *Business impact*: This flow is integral to the critical path of revenue-generating transactions for the business. High availability is essential, as call-center employees rely on this flow to process ticket purchases. In recognition of its importance, the business mandates a 99.9% uptime for this flow, which includes extended business hours.
- *Criticality rating*: High.

User flow 4: Purchase ticket

Flow description: Call-center employees use the application (the *authentication and authorization* process) to buy tickets for an upcoming concert (the *list upcoming concerts* process) on behalf of Relecloud customers.

- *Business processes*: This flow is the core feature and flow of the application.
- *Process owner*: Director of Sales.
- *Stakeholders*: Sales department and all technical teams.
- *Escalation path*: Application team on-call engineer, platform team on-call engineer, data team on-call engineer, Chief Operating Officer.
- *Business impact*: High availability of this flow is crucial, as it directly enables customer ticket purchases. Any malfunction or unavailability of this flow can significantly impact both revenue and the company's reputation. The business set

a stringent requirement for this vital process, expecting 99.9% uptime, even during extended business hours.

- *Criticality rating*: High.

User flow 5: Authentication and authorization

Flow description: Call-center employees securely sign in to the application. Administrators provide them with the proper roles to purchase tickets on behalf of Relecloud customers.

- *Business processes*: This flow directly supports the *purchasing ticket* process. Without this functionality, call-center employees can't sign into the application to buy tickets.
- *Process owner*: Platform team.
- *Stakeholders*: Platform team, operations team, and sales department.
- *Escalation path*: Platform team on-call engineer, Chief Operating Officer.
- *Business impact*: This flow requires high availability because call-center employees can't purchase tickets if this flow isn't working properly. If this flow isn't available, it directly affects revenue and reputation. It's a key process that the business expects 99.9% uptime for, including during extended business hours.
- *Criticality rating*: High.

System flow: Collect telemetry

Flow description: To understand state changes in the production system, web application and API instances collect and send information, errors, and warnings. This data helps the operations team perform anomaly detection, troubleshooting, and profiling.

- *Business processes*: This flow doesn't support any business processes, but it provides important data for the operations team.
- *Process owner*: Director of Operations.
- *Stakeholders*: Operations team, platform team, and data team.
- *Escalation path*: Operations team (24/7), data team on-call engineer.
- *Business impact*: This flow is essential for the business's monitoring and continuous improvement efforts. It needs to be as redundant and resilient as possible. The

operations team is responsible for quickly restoring this flow after any failure to avoid missing critical information and warnings. If the flow fails to achieve the expected availability, there's a risk of overlooking production issues, potentially leading to severe consequences. To mitigate this risk, the operations department aims for 99% uptime, 24/7. They must schedule maintenance-related downtime at least 48 hours in advance.

- *Criticality rating*: Medium.

Feedback

Was this page helpful? Yes No

Recommendations for handling transient faults

Article • 02/28/2025

This guide describes the recommendations for handling transient faults in your cloud applications. All applications that communicate with remote services and resources must be sensitive to transient faults. This is especially true for applications that run in the cloud, where, because of the nature of the environment and connectivity over the internet, this type of fault is likely to be encountered more often. Transient faults include the momentary loss of network connectivity to components and services, the temporary unavailability of a service, and timeouts that occur when a service is busy. These faults are often self-correcting, so, if the action is repeated after a suitable delay, it's likely to succeed.

This article provides general guidance for transient fault handling. For information about implementing retries in your application code to handle transient faults, see the retry pattern and when you're using Azure services, see the Retry guidance for Azure services.

Transient Faults

Transient faults can occur in any environment, on any platform or operating system, and in any kind of application. For solutions that run on local on-premises infrastructure, the performance and availability of the application and its components are typically maintained via expensive and often underused hardware redundancy, and components and resources are located close to each other. This approach makes failure less likely, but transient faults can still occur, as can outages caused by unforeseen events like external power supply or network issues, or by disaster scenarios.

Cloud hosting, including private cloud systems, can offer higher overall availability by using shared resources, redundancy, automatic failover, and dynamic resource allocation across many commodity compute nodes. However, because of the nature of cloud environments, transient faults are more likely to occur. There are several reasons for this:

- Many resources in a cloud environment are shared, and access to these resources is subject to throttling in order to protect the resources. Some services refuse connections when the load rises to a specific level, or when a maximum throughput rate is reached, to allow processing of existing requests and to maintain performance of the service for all users. Throttling helps to maintain the quality of service for neighbors and other tenants that use the shared resource.

- Cloud environments use large numbers of commodity hardware units. They deliver performance by dynamically distributing load across multiple computing units and infrastructure components. They deliver reliability by automatically recycling or replacing failed units. Because of this dynamic nature, transient faults and temporary connection failures might occasionally occur.

- There are often more hardware components, including network infrastructure like routers and load balancers, between the application and the resources and services that it uses. This additional infrastructure can occasionally introduce additional connection latency and transient connection faults.

- Network conditions between the client and the server might be variable, especially when communication crosses the internet. Even in on-premises locations, heavy traffic loads can slow communication and cause intermittent connection failures.

Transient faults can have a significant effect on the perceived availability of an application, even if it's been thoroughly tested under all foreseeable circumstances. To ensure that cloud-hosted applications operate reliably, you need to ensure that they can respond to the following challenges:

- The application must be able to detect faults when they occur and determine if the faults are likely to be transient, are long-lasting, or are terminal failures. Different resources are likely to return different responses when a fault occurs, and these responses can also vary depending on the context of the operation. For example, the response for an error when the application is reading from storage might differ from the response for an error when it's writing to storage. Many resources and services have well-documented transient-failure contracts. However, when such information isn't available, it can be difficult to discover the nature of the fault and whether it's likely to be transient.

- The application must be able to retry the operation if it determines that the fault is likely to be transient. It also needs to keep track of the number of times the operation is retried.

- The application must use an appropriate strategy for retries. The strategy specifies the number of times the application should retry, the delay between each attempt, and the actions to take after a failed attempt. The appropriate number of attempts and the delay between each one are often difficult to determine. The strategy varies depending on the type of resource and on the current operating conditions of the resource and the application.

The following guidelines can help you design suitable transient fault handling mechanisms for your applications.

Implementing retries

Determine if there's a built-in retry mechanism

- Many services provide an SDK or client library that contains a transient fault handling mechanism. The retry policy it uses is typically tailored to the nature and requirements of the target service. Alternatively, REST interfaces for services might return information that can help you determine whether a retry is appropriate and how long to wait before the next retry attempt.
- You should use the built-in retry mechanism when one is available, unless you have specific and well-understood requirements that make a different retry behavior more appropriate.

Determine if the operation is suitable for retrying

- Perform retry operations only when the faults are transient (typically indicated by the nature of the error) and when there's at least some likelihood that the operation will succeed when retried. There's no point in retrying operations that attempt an invalid operation, like a database update to an item that doesn't exist or a request to a service or resource that suffered a fatal error.
- In general, implement retries only when you can determine the full effect of doing so and when the conditions are well understood and can be validated. Otherwise, let the calling code implement retries. Remember that the errors returned from resources and services outside your control might evolve over time, and you might need to revisit your transient fault detection logic.
- When you create services or components, consider implementing error codes and messages that help clients determine whether they should retry failed operations. In particular, indicate whether the client should retry the operation (perhaps by returning an **isTransient** value) and suggest a suitable delay before the next retry attempt. If you build a web service, consider returning custom errors that are defined within your service contracts. Even though generic clients might not be able to read these errors, they're useful in the creation of custom clients.

Determine an appropriate retry count and interval

- Optimize the retry count and the interval to the type of use case. If you don't retry enough times, the application can't complete the operation and will probably fail. If you retry too many times, or with too short an interval between tries, the

application might hold resources like threads, connections, and memory for long periods, which adversely affects the health of the application.

- Adapt values for the time interval and the number of retry attempts to the type of operation. For example, if the operation is part of a user interaction, the interval should be short and only a few retries should be attempted. By using this approach, you can avoid making users wait for a response, which holds open connections and can reduce availability for other users. If the operation is part of a long running or critical workflow, where canceling and restarting the process is expensive or time-consuming, it's appropriate to wait longer between attempts and retry more times.

- Keep in mind that determining the appropriate intervals between retries is the most difficult part of designing a successful strategy. Typical strategies use the following types of retry interval:

    - **Exponential back-off**. The application waits a short time before the first retry and then exponentially increases the time between each subsequent retry. For example, it might retry the operation after 3 seconds, 12 seconds, 30 seconds, and so on. To further improve this strategy, you can add jitter to the exponential back-off. Jitter introduces a random delay to each retry attempt, which helps to prevent multiple clients from retrying simultaneously and causing a spike in load

    - **Incremental intervals**. The application waits a short time before the first retry, and then incrementally increases the time between each subsequent retry. For example, it might retry the operation after 3 seconds, 7 seconds, 13 seconds, and so on.

    - **Regular intervals**. The application waits for the same period of time between each attempt. For example, it might retry the operation every 3 seconds.

    - **Immediate retry**. Sometimes a transient fault is brief, possibly caused by an event like a network packet collision or a spike in a hardware component. In this case, retrying the operation immediately is appropriate because it might succeed if the fault is cleared in the time that it takes the application to assemble and send the next request. However, there should never be more than one immediate retry attempt. You should switch to alternative strategies, like exponential back-off or fallback actions, if the immediate retry fails.

    - **Randomization**. Any of the retry strategies listed previously can include a randomization to prevent multiple instances of the client sending subsequent retry attempts at the same time. For example, one instance might retry the

operation after 3 seconds, 11 seconds, 28 seconds, and so on, while another instance might retry the operation after 4 seconds, 12 seconds, 26 seconds, and so on. Randomization is a useful technique that can be combined with other strategies.

- As a general guideline, use an exponential back-off with jitter strategy for background operations, and use immediate or regular interval retry strategies for interactive operations. In both cases, you should choose the delay and the retry count so that the maximum latency for all retry attempts is within the required end-to-end latency requirement.

- Take into account the combination of all factors that contribute to the overall maximum timeout for a retried operation. These factors include the time it takes for a failed connection to produce a response (typically set by a timeout value in the client), the delay between retry attempts, and the maximum number of retries. The total of all these times can result in long overall operation times, especially when you use an exponential delay strategy where the interval between retries grows rapidly after each failure. If a process must meet a specific service-level agreement (SLA), the overall operation time, including all timeouts and delays, must be within the limits defined in the SLA.

- Don't implement overly aggressive retry strategies. These are strategies that have intervals that are too short or retries that are too frequent. They can have an adverse effect on the target resource or service. These strategies might prevent the resource or service from recovering from its overloaded state, and it will continue to block or refuse requests. This scenario results in a vicious circle, where more and more requests are sent to the resource or service. Consequently, its ability to recover is further reduced.

- Take into account the timeout of the operations when you choose retry intervals in order to avoid launching a subsequent attempt immediately (for example, if the timeout period is similar to the retry interval). Also, consider whether you need to keep the total possible period (the timeout plus the retry intervals) below a specific total time. If an operation has an unusually short or long timeout, the timeout might influence how long to wait and how often to retry the operation.

- Use the type of the exception and any data it contains, or the error codes and messages returned from the service, to optimize the number of retries and the interval between them. For example, some exceptions or error codes (like the HTTP code 503, Service Unavailable, with a Retry-After header in the response) might indicate how long the error might last, or that the service failed and won't respond to any subsequent attempt.

- Consider using a dead-letter queue approach to make sure that all the information from the incoming invocation doesn't get lost after all retry attempts have been exhausted.

Avoid anti-patterns

- In most cases, avoid implementations that include duplicated layers of retry code. Avoid designs that include cascading retry mechanisms or that implement retry at every stage of an operation that involves a hierarchy of requests, unless you have specific requirements that require doing so. In these exceptional circumstances, use policies that prevent excessive numbers of retries and delay periods, and make sure you understand the consequences. For example, say one component makes a request to another, which then accesses the target service. If you implement retry with a count of three on both calls, there are nine retry attempts in total against the service. Many services and resources implement a built-in retry mechanism. You should investigate how you can disable or modify these mechanisms if you need to implement retries at a higher level.

- Never implement an endless retry mechanism. Doing so is likely to prevent the resource or service from recovering from overload situations and to cause throttling and refused connections to continue for a longer time. Use a finite number of retries, or implement a pattern like Circuit Breaker to allow the service to recover.

- Never perform an immediate retry more than once.

- Avoid using a regular retry interval when you access services and resources on Azure, especially when you have a high number of retry attempts. The best approach in this scenario is an exponential back-off strategy with a circuit-breaking capability.

- Prevent multiple instances of the same client, or multiple instances of different clients, from sending retries simultaneously. If this scenario is likely to occur, introduce randomization into the retry intervals.

Test retry strategies and implementation

- Fully test your retry strategy under as wide a set of circumstances as possible, especially when both the application and the target resources or services that it uses are under extreme load. To check behavior during testing, you can:

- Include transient faults in your chaos engineering and fault injection practices by purposely introducing them into your nonproduction and production environments. For example, send invalid requests or add code that detects test requests and responds with different types of errors.
- Create a mockup of the resource or service that returns a range of errors that the real service might return. Cover all the types of errors that your retry strategy is designed to detect.
- For custom services that you create and deploy, force transient errors to occur by temporarily disabling or overloading the service. (Don't attempt to overload any shared resources or shared services in Azure.)
- Use libraries or solutions that intercept and modify network traffic to replicate unfavorable scenarios from your automated tests. For example, the tests can add extra roundtrip times, drop packets, modify headers, or even change the body of the request itself. Doing so enables deterministic testing of a subset of the failure conditions, for transient faults and other types of failures.
- When testing a client web application's resiliency to transient faults, use the browser's developer tools or your testing framework's ability to mock or block network requests.
- Perform high load factor and concurrent tests to ensure that the retry mechanism and strategy works correctly under these conditions. These tests also help ensure that the retry doesn't have an adverse effect on the operation of the client or cause cross-contamination between requests.

Manage retry policy configurations

- A *retry policy* is a combination of all the elements of your retry strategy. It defines the detection mechanism that determines whether a fault is likely to be transient, the type of interval to use (like regular, exponential back-off, and randomization), the actual interval values, and the number of times to retry.
- Implement retries in many places, even in the simplest application, and in every layer of more complex applications. Rather than hard-coding the elements of each policy at multiple locations, consider using a central point to store all policies. For example, store values like the interval and retry count in application configuration files, read them at runtime, and programmatically build the retry policies. Doing so makes it easier to manage the settings and to modify and fine-tune the values in order to respond to changing requirements and scenarios. However, design the

system to store the values rather than rereading a configuration file every time, and use suitable defaults if the values can't be obtained from configuration.

- Store the values that are used to build the retry policies at runtime in the application's configuration system so that you can change them without needing to restart the application.

- Take advantage of built-in or default retry strategies that are available in the client APIs that you use, but only when they're appropriate for your scenario. These strategies are typically generic. In some scenarios, they might be all you need, but in other scenarios they don't offer the full range of options to suit your specific requirements. To determine the most appropriate values, you need to perform testing to understand how the settings affect your application.

Log and track transient and nontransient faults

- As part of your retry strategy, include exception handling and other instrumentation that logs retry attempts. An occasional transient failure and retry are expected and don't indicate a problem. Regular and increasing numbers of retries, however, are often an indicator of a problem that might cause a failure or that degrades application performance and availability.

- Log transient faults as warning entries rather than as error entries so that monitoring systems don't detect them as application errors that might trigger false alerts.

- Consider storing a value in your log entries that indicates whether retries are caused by throttling in the service or by other types of faults, like connection failures, so that you can differentiate them during analysis of the data. An increase in the number of throttling errors is often an indicator of a design flaw in the application or the need to switch to a premium service that offers dedicated hardware.

- Consider measuring and logging the overall elapsed times for operations that include a retry mechanism. This metric is a good indicator of the overall effect of transient faults on user response times, process latency, and the efficiency of application use cases. Also log the number of retries that occur so you can understand the factors that contribute to the response time.

- Consider implementing a telemetry and monitoring system that can raise alerts when the number and rate of failures, the average number of retries, or the overall times elapsed before operations succeed is increasing.

Manage operations that continually fail

- Consider how to handle operations that continue to fail at every attempt. Situations like this are inevitable.
    - Although a retry strategy defines the maximum number of times that an operation should be retried, it doesn't prevent the application from repeating the operation again with the same number of retries. For example, if an order processing service fails with a fatal error that puts it out of action permanently, the retry strategy might detect a connection timeout and consider it to be a transient fault. The code retries the operation a specified number of times and then gives up. However, when another customer places an order, the operation is attempted again, even though it will fail every time.
    - To prevent continual retries for operations that continually fail, you should consider implementing the Circuit Breaker pattern. When you use this pattern, if the number of failures within a specified time window exceeds a threshold, requests return to the caller immediately as errors, and there's no attempt to access the failed resource or service.
    - The application can periodically test the service, on an intermittent basis and with long intervals between requests, to detect when it becomes available. An appropriate interval depends on factors like the criticality of the operation and the nature of the service. It might be anything between a few minutes and several hours. When the test succeeds, the application can resume normal operations and pass requests to the newly recovered service.
    - In the meantime, you might be able to fall back to another instance of the service (maybe in a different datacenter or application), use a similar service that offers compatible (maybe simpler) functionality, or perform some alternative operations based on the hope that the service will be available soon. For example, it might be appropriate to store requests for the service in a queue or data store and retry them later. Or you might be able to redirect the user to an alternative instance of the application, degrade the performance of the application but still offer acceptable functionality, or just return a message to the user to indicate that the application isn't currently available.

Optimize retry implementation

- When you're deciding on the values for the number of retries and the retry intervals for a policy, consider whether the operation on the service or resource is part of a long-running or multistep operation. It might be difficult or expensive to

compensate all the other operational steps that have already succeeded when one fails. In this case, a very long interval and a large number of retries might be acceptable as long as that strategy doesn't block other operations by holding or locking scarce resources.

- Consider whether retrying the same operation could cause inconsistencies in data. If some parts of a multistep process are repeated and the operations aren't idempotent, inconsistencies might occur. For example, if an operation that increments a value is repeated, it produces an invalid result. Repeating an operation that sends a message to a queue might cause an inconsistency in the message consumer if the consumer can't detect duplicate messages. To prevent these scenarios, design each step as an idempotent operation. For more information, see Idempotency patterns.

- Consider the scope of operations that are retried. For example, it might be easier to implement retry code at a level that encompasses several operations and retry them all if one fails. However, doing so might result in idempotency issues or unnecessary rollback operations.

- If you choose a retry scope that encompasses several operations, take into account the total latency of all of them when you determine retry intervals, when you monitor the elapsed times of the operation, and before you raise alerts for failures.

- Consider how your retry strategy might affect neighbors and other tenants in a shared application and when you use shared resources and services. Aggressive retry policies can cause an increasing number of transient faults to occur for these other users and for applications that share the resources and services. Likewise, your application might be affected by the retry policies implemented by other users of the resources and services. For business-critical applications, you might want to use premium services that aren't shared. Doing so gives you more control over the load and consequent throttling of these resources and services, which can help to justify the extra cost.

> **Note**
>
> See Issues and considerations in the Retry pattern article for further guidance on tradeoffs and risks.

Azure facilitation

Most Azure services and client SDKs provide a retry mechanism. However, these mechanisms differ because each service has different characteristics and requirements, and each retry mechanism is tuned to the specific service. This section summarizes the retry mechanism features for some commonly used Azure services.

Expand table

Service	Retry capabilities	Policy configuration	Scope	Telemetry features
Microsoft Entra ID	Native in the Microsoft Authentication Library (MSAL)	Embedded into the MSAL library	Internal	None
Azure Cosmos DB	Native in the service	Not configurable	Global	TraceSource
Azure Data Lake Storage	Native in the client	Not configurable	Individual operations	None
Azure Event Hubs	Native in the client	Programmatic	Client	None
Azure IoT Hub	Native in the client SDK	Programmatic	Client	None
Azure Cache for Redis	Native in the client	Programmatic	Client	TextWriter
Azure Cognitive Search	Native in the client	Programmatic	Client	ETW or custom
Azure Service Bus	Native in the client	Programmatic	NamespaceManager, MessagingFactory, and client	ETW
Azure Service Fabric	Native in the client	Programmatic	Client	None
Azure SQL Database with ADO.NET	Polly	Declarative and programmatic	Single statements or blocks of code	Custom
SQL Database with Entity Framework	Native in the client	Programmatic	Global per AppDomain	None

Service	Retry capabilities	Policy configuration	Scope	Telemetry features
SQL Database with Entity Framework Core	Native in the client	Programmatic	Global per AppDomain	None
Azure Storage	Native in the client	Programmatic	Client and individual operations	TraceSource

> **Note**
>
> For most of the Azure built-in retry mechanisms, there's currently no way to apply a different retry policy for different types of errors or exceptions. You should configure a policy that provides the optimum average performance and availability. One way to fine-tune your policy is to analyze log files to determine the type of transient faults that are occurring.

Example

See Reliable web app pattern for .NET for an example that uses many of the patterns discussed in this article. There's also a reference implementation on GitHub.

Related links

- Circuit Breaker pattern
- Retry pattern
- Throttling pattern
- Compensating Transaction pattern
- A blog post on idempotency patterns
- Connection Resiliency
- Inject mock services
- Dead-letter queue pattern

Reliability checklist

Refer to the complete set of recommendations.

Reliability checklist

Feedback

Was this page helpful? Yes No

Recommendations for developing background jobs

Article • 02/28/2025

This guide describes the recommendations for developing background jobs. Background jobs run automatically without the need for user interaction. Many applications require background jobs that run independent of the UI.

Some examples of background jobs include batch jobs, intensive processing tasks, and long-running processes, such as workflows. The application starts the job and processes interactive requests from users. For example, if an application needs to generate thumbnails of images that users upload, a background job can be performed to generate the thumbnail and save it to storage. The user doesn't have to wait for the process to complete. As another example, a customer places an order, which initiates a background workflow that processes the order. The customer continues to browse the web app while the background job runs. After the background job finishes, it updates the stored order data and sends an email to the customer to confirm the order.

Background jobs help minimize the load on the application UI, which improves availability and reduces interactive response time.

To choose which task to designate as a background job, consider whether the task runs without user interaction and whether the UI needs to wait for the task to complete. Tasks that require the user or the UI to wait while they run are typically not appropriate background jobs.

Evaluate the need for background jobs

Some examples of background jobs are:

- CPU-intensive jobs, such as mathematical calculations or structural model analysis.
- I/O-intensive jobs, such as running a series of storage transactions or indexing files.
- Batch jobs, such as nightly data updates or scheduled processing.
- Long-running workflows, such as order fulfillment or provisioning services and systems.

- Sensitive-data processing that transfers the task to a more secure location for processing. For example, you might not want to process sensitive data within a web app. Instead, you might use a pattern such as the Gatekeeper pattern to transfer the data to an isolated background process that has access to protected storage.

Choose the right triggers

Initiate background jobs with:

- Event-driven triggers: An event, typically a user action or a step in a workflow, triggers the task.
- Schedule-driven triggers: A schedule that's based on a timer invokes the task. The job can be scheduled on a recurring basis or for a single run.

Event-driven triggers

An action triggers an event-driven invocation that starts the background task. Examples of event-driven triggers include:

- The UI or a different job places a message in a queue, as described in the Web-Queue-Worker architectural style. The message contains data about a previously performed action, such as a customer that placed an order. The background job monitors this queue and detects the arrival of a new message. It reads the message and uses the message's data as the input for the background job. This pattern is called asynchronous message-based communication.
- The UI or a different job saves or updates a value that's in storage. The background job monitors the storage and detects changes. It reads the data and uses it as the input for the background job.
- The UI or a different job makes a request to an endpoint, such as an HTTPS URI or an API that's exposed as a web service. As part of the request, the UI or job transfers the data that the background task requires. The endpoint or web service invokes the background task, which uses the data as its input.

Other examples of tasks that are suited to event-driven invocation include image processing, workflows, sending information to remote services, sending email messages, and provisioning new users in multitenant applications.

Schedule-driven triggers

A timer triggers a schedule-driven invocation that starts the background task. Examples of schedule-driven triggers include:

- A timer that runs locally within the application or as part of the application's operating system regularly invokes a background task.
- A timer that runs in a different application, such as Azure Logic Apps, regularly sends a request to an API or web service. The API or web service invokes the background task.
- A separate process or application starts a timer that invokes the background task after a time delay or at a specific time.

Other examples of tasks that are suited to schedule-driven invocation include batch-processing routines (such as updating related products lists for customers based on their recent behavior), routine data-processing tasks (such as updating indexes or generating accumulated results), data analysis for daily reports, data retention cleanup, and data consistency checks.

If you use a schedule-driven task that must run as a single instance, review the following considerations:

- If the compute instance that runs the scheduler, such as a virtual machine (VM) that uses Windows scheduled tasks, is scaled, then you are running multiple instances of the scheduler. Multiple instances of the scheduler can start multiple instances of the task. For more information, see What does idempotent mean in software systems?
- If tasks run longer than the period between the scheduler events, the scheduler might start another instance of the task while the previous task runs.

Return data to the workload

Background jobs run asynchronously in a separate process, or even in a separate location, from the UI or the process that invoked the background job. Ideally, background jobs are *fire and forget* operations. Their runtime progress doesn't have an effect on the UI or the calling process, which means that the calling process doesn't wait for the tasks to complete. The UI and the calling process can't detect when the task ends.

If you require a background task to communicate with the calling task to indicate progress or completion, you must implement a mechanism. Some examples are to:

- Write a status indicator value to storage that's accessible to the UI or the caller task, which can monitor or check this value. Other data that the background task returns to the caller can be placed in the same storage.

- Establish a reply queue that the UI or caller monitors. The background task can send messages to the queue that indicate the status. Data that the background task returns to the caller can be placed in the messages. For Azure Service Bus, use the `ReplyTo` and `CorrelationId` properties to implement this capability.

- Expose an API or endpoint from the background task that the UI or caller can access to obtain status information. The response can include the data that the background task returns to the caller.

- Configure the background task to call back to the UI or caller via an API to indicate the status at predefined points or on completion. You can use events raised locally or a publish-and-subscribe mechanism. The request or the event payload can include the data that the background task returns to the caller.

Partition background jobs

If you include background jobs in an existing compute instance, consider how these changes affect the quality attributes of the compute instance and the background job. Consider these factors to decide whether to colocate the tasks with the existing compute instance or separate them into a different compute instance:

- **Availability**: Background tasks might not need the same level of availability as other parts of the application, in particular the UI and parts that directly involve user interaction. Background tasks might have a higher tolerance for latency, retried connection failures, and other factors that affect availability because the operations can be queued. However, there must be sufficient capacity to prevent backed up requests that can block queues and affect the entire application.

- **Scalability**: Background tasks likely have different scalability requirements compared to the UI and the interactive parts of the application. You might need to scale the UI to meet peaks in demand. Outstanding background tasks can run during less busy times and with fewer compute instances.

- **Resiliency**: If a compute instance that hosts only background tasks fails, it might not fatally affect the entire application. The requests for these tasks can be queued or postponed until the task is available. If the compute instance or tasks can restart within an appropriate interval, it might not affect the application users.

- **Security**: Background tasks might have different security requirements or restrictions compared to the UI or other parts of the application. Use a separate compute instance to specify a different security environment for the tasks. To maximize security and separation, you can also use patterns such as Gatekeeper to isolate the background compute instances from the UI.
- **Performance**: Choose the type of compute instance for background tasks that specifically matches the task's performance requirements. You might use a less expensive compute option if the tasks don't require the same processing capabilities as the UI. Or you might use a larger instance if the tasks require more capacity and resources.
- **Manageability**: Background tasks might have a different development and deployment rhythm compared to the main application code or the UI. To simplify updates and versioning, deploy background tasks to a separate compute instance.
- **Cost**: If you add compute instances to run background tasks, hosting costs increase. Carefully consider the tradeoff between more capacity and extra costs.

For more information, see Leader Election pattern and Competing Consumers pattern.

Prevent resource conflict

If you have multiple instances of a background job, they might compete for access to resources and services, such as databases and storage. This concurrent access can result in resource contention, which might cause service availability conflicts and harm the integrity of the data that's in storage. Resolve resource contention by using a pessimistic-locking approach. This approach prevents competing instances of a task from concurrently accessing a service or corrupting data.

Another approach to resolve conflicts is to define background tasks as a singleton, so that only one instance runs. However, this approach eliminates the reliability and performance benefits that a multiple-instance configuration provides. This disadvantage is especially true if the UI supplies enough work to keep more than one background task busy.

Ensure that the background task can automatically restart and that it has sufficient capacity to handle peaks in demand. Allocate a compute instance with sufficient resources, implement a queueing mechanism that stores requests to run when demand decreases, or use a combination of these techniques.

Orchestrate multiple tasks

Background tasks can be complex and require multiple tasks to run. In these scenarios, it's common to divide the task into smaller discrete steps or subtasks that multiple consumers can run. Multistep jobs are more efficient and more flexible because individual steps are often reusable in multiple jobs. It's also easy to add, remove, or modify the order of the steps.

It can be a challenge to coordinate multiple tasks and steps, but there are three common patterns to guide your solution:

- **Decompose a task into multiple reusable steps**. An application might be required to perform various tasks of different complexity on the information that it processes. A straightforward but inflexible approach to implementing such an application is to perform this processing as a monolithic module. But this approach is likely to reduce the opportunities for refactoring the code, optimizing it, or reusing it if the application requires parts of the same processing elsewhere. For more information, see the Pipes and Filters pattern.

- **Manage the orchestration of the steps for a task**. An application might perform tasks that comprise many steps, some of which might invoke remote services or access remote resources. Sometimes the individual steps are independent of each other, but they're orchestrated by the application logic that implements the task. For more information, see Scheduler Agent Supervisor pattern.

- **Manage the recovery for task steps that fail**. If one or more of the steps fail, an application might need to undo the work that a series of steps performs, which together defines an eventually consistent operation. For more information, see Compensating Transaction pattern.

Make jobs resilient

Create resilient background tasks to provide reliable services for the application. When you plan and design background tasks, consider the following points:

- Background tasks need to gracefully handle restarts without corrupting data or introducing inconsistency into the application. For long-running or multistep tasks, consider using *checkpoints*. Use checkpoints to save the state of jobs in persistent storage or as messages in a queue. For example, you can store state information in a message that's in a queue and incrementally update this state information with the task progress. The task can be processed from the last known checkpoint instead of restarting from the beginning.

For Service Bus queues, use message sessions for this purpose. With message sessions, save and retrieve the application processing state by using the SetState and GetState methods. For more information about designing reliable multistep processes and workflows, see Scheduler Agent Supervisor pattern.

- When you use queues to communicate with background tasks, the queues can act as a buffer to store requests that are sent to the tasks while the application is under higher than usual load. The tasks can catch up with the UI during less busy periods, and restarts don't block the UI. For more information, see Queue-Based Load Leveling pattern. If some tasks are more important than others, consider implementing the Priority Queue pattern to ensure that these tasks run first.

Messages

Configure background tasks that are initiated by messages or that process messages to handle inconsistencies, such as messages that arrive out of order, messages that repeatedly cause an error (*poison messages*), and messages that are delivered more than once. Consider the following recommendations:

- Sometimes you need messages to be processed in a specific order, like messages that change data based on the existing data value, for example adding a value to an existing value. Messages don't always arrive in the order that they were sent. Also, different instances of a background task might process messages in a different order due to varying loads on each instance.

  For messages that must be processed in a specific order, include a sequence number, key, or another indicator that background tasks can use to process messages in the correct order. For Service Bus, use message sessions to guarantee the correct order of delivery. It's more efficient to design the process so that the message order isn't important. For more information, see message sequencing and timestamps.

- Typically, a background task peeks at messages in the queue, which temporarily hides them from other message consumers. After the task successfully processes the message, it deletes the message. If a background task fails when it processes a message, that message reappears in the queue after the peek timeout expires. A different instance of the task processes the message, or the next processing cycle of the original instance processes the message.

  If the message consistently causes an error in the consumer, it blocks the task, the queue, and eventually the application itself when the queue becomes full. It's vital to detect and remove poison messages from the queue. If you use Service Bus,

automatically or manually move poison messages to an associated dead letter queue.

- Queues are *at-least-once* delivery mechanisms, but they might deliver the same message more than once. If a background task fails after it processes a message but before it deletes it from the queue, the message is available for processing again.

  Background tasks should be idempotent, which means that when the task processes the same message more than once, it doesn't cause an error or inconsistency in the application's data. Some operations are naturally idempotent, for example if a stored value is set to a specific new value. However, some operations cause inconsistencies, for example if a value is added to an existing stored value without verifying that the stored value is still the same as when the message was originally sent. Configure Service Bus queues to automatically remove duplicated messages. For more information, see Idempotent message processing.

- Some messaging systems, such as Azure Storage queues and Service Bus queues, support a dequeue count property that indicates how many times a message from the queue is read. This data is useful for handling repeated messages and poison messages. For more information, see Asynchronous messaging primer and Idempotency patterns.

Make jobs scalable

Background tasks must offer sufficient performance to ensure that they don't block the application or delay operation when the system is under load. Typically, performance improves when you scale the compute instances that host the background tasks. When you plan and design background tasks, consider the following points related to scalability and performance:

- Azure Virtual Machines and the Web Apps feature of Azure App Service can host deployments. They support automatic scaling, both scaling out and scaling in. The automatic scaling is determined by the demand and load or a predefined schedule. Use automatic scaling to help ensure that the application has sufficient performance capabilities while minimizing runtime costs.

- Some background tasks have a different performance capability compared to other parts of an application, for example the UI or components, such as the data access layer. In that scenario, host the background tasks together in a separate compute service so the UI and background tasks can scale independently to

manage the load. If multiple background tasks have significantly different performance capabilities, divide them and scale each type independently. This technique might increase runtime costs.

- To prevent the loss of performance under load, you might also need to scale storage queues and other resources so a single point of the processing chain doesn't cause a bottleneck. Consider other limitations, such as the maximum throughput of storage and other services that the application and the background tasks rely on.

- Design background tasks for scaling. For example, background tasks must dynamically detect the number of utilized storage queues to monitor messages or send messages to the appropriate queue.

- By default, a WebJob scales with its associated Web Apps instance. However, if you want a WebJob to run as only a single instance, you can create a Settings.job file that contains the JSON data `{ "is_singleton": true }`. This method forces Azure to only run one instance of the WebJob, even if there are multiple instances of the associated web app. This technique is useful for scheduled jobs that must run as only a single instance.

- Background jobs can create challenges for data synchronization and process coordination, especially if the background tasks depend on each other or on other data sources. For example, background jobs might handle data consistency problems, race conditions, deadlocks, or timeouts.

- Background jobs might affect the user experience if the results of the background tasks are presented to the user. For example, background jobs might require the user to wait for a notification, refresh the page, or manually check the status of the task. These behaviors can increase the complexity of the user interaction and negatively affect the user experience.

>  **Tradeoff**: Background jobs introduce more components and dependencies to the system, which can increase the complexity and maintenance costs of the solution. For example, background jobs might require a separate queue service, worker service, monitoring service, and retry mechanism.

Azure facilitation

The following sections describe the Azure services that you can use to host, run, configure, and manage background jobs.

Host environments

There are several Azure platform services that can host background tasks:

- Web Apps and WebJobs: Use the WebJobs feature of App Service to run custom jobs that are based on different scripts or programs that you can run in a web app.
- Azure Functions: Use function apps for background jobs that don't run for a long time. You can also use function apps if you host your workload on an underutilized App Service plan.
- Azure Logic Apps: Use logic apps for background jobs that require orchestration across multiple services and systems.
- Virtual Machines: If you have a Windows service or want to use Windows Task Scheduler, host your background tasks in a dedicated VM.
- Azure Batch: Batch is a platform service that you can use to schedule compute-intensive work to run on a managed collection of VMs. It can automatically scale compute resources.
- Azure Kubernetes Service (AKS): AKS provides a managed hosting environment for Kubernetes on Azure.
- Azure Container Apps: With Container Apps, you can build serverless microservices that are based on containers.

The following sections provide considerations for each of these options to help you choose the best option for you.

Web Apps and WebJobs

You can use the WebJobs feature to run custom jobs as background jobs in a web app. A WebJob runs as a continuous process in the context of your web app. A WebJob can also run in response to a trigger event from Logic Apps or external factors, such as changes to storage blobs or message queues. WebJobs can be started and stopped on demand, and shut down gracefully. If a continuously running WebJob fails, it's automatically restarted. You can configure retry and error actions.

When you configure a WebJob:

- If you want the job to respond to an event-driven trigger, configure it to **Run continuously**. The script or program is stored in the folder that's named site/wwwroot/app_data/jobs/continuous.

- If you want the job to respond to a schedule-driven trigger, configure it to **Run on a schedule**. The script or program is stored in the folder named site/wwwroot/app_data/jobs/triggered.
- If you choose the **Run on demand** option when you configure a job, it runs the same code as the **Run on a schedule** option when you start the job.

A WebJob runs in the sandbox of the web app. It has access to environment variables and it can share information, such as connection strings, with the web app. The WebJob has access to the unique identifier of the machine that runs the WebJob. The connection string named `AzureWebJobsStorage` provides access to Storage queues, blobs, and tables for application data. It also provides access to Service Bus for messaging and communication. The connection string named `AzureWebJobsDashboard` provides access to the WebJob action log files.

WebJobs have the following characteristics:

- **Security**: The deployment credentials of the web app provide protection for WebJobs.
- **Supported file types**: Define WebJobs by using command scripts (.cmd), batch files (.bat), PowerShell scripts (.ps1), Bash shell scripts (.sh), PHP scripts (.php), Python scripts (.py), JavaScript code (.js), and executable programs (.exe and .jar).
- **Deployment**: You can deploy scripts and executables by using the Azure portal, Visual Studio, or the WebJobs SDK, or you can copy them directly to the following locations:
  - For triggered deployment: site/wwwroot/app_data/jobs/triggered/<job name>
  - For continuous deployment: site/wwwroot/app_data/jobs/continuous/<job name>
- **Log files**: `Console.Out` is treated, or marked, as `INFO`. `Console.Error` is treated as `ERROR`. Use the portal to access monitoring and diagnostics information. Download log files directly from the website. Log files are saved in the following locations:
  - For triggered deployment: Vfs/data/jobs/triggered/<job name>
  - For continuous deployment: Vfs/data/jobs/continuous/<job name>
- **Configuration**: Configure WebJobs by using the portal, the REST API, and PowerShell. Use a configuration file named settings.job, which is in the same root directory as the WebJob script, to provide configuration information for a WebJob. For example:

- `{ "stopping_wait_time": 60 }`
- `{ "is_singleton": true }`

Web Apps and WebJobs considerations

- By default, WebJobs scale with the web app. To configure WebJobs to run on a single instance, set the `is_singleton` configuration property to `true`. Single instance WebJobs are useful for tasks that you don't want to scale or run as simultaneous multiple instances, such as reindexing or data analysis.
- To minimize the effect of WebJobs on the performance of the web app, create an empty web app instance in a new App Service plan to host long-running or resource-intensive WebJobs.

Azure Functions

Azure Functions is serverless and is most suitable for event-driven triggers that run for a short period. You can also use Azure Functions to run scheduled jobs via timer triggers if you configure a function to run at specified times.

Use Azure Durable Functions for orchestrating complex workflows and long-running processes. Durable Functions allow you to define stateful workflows in a serverless environment, which can be particularly useful for background jobs that require coordination and state management.

Azure Functions considerations

If you expect the background task to run for a short duration in response to an event, consider running the task in the consumption plan. You can configure the runtime to a maximum time. A function that runs for longer costs more. CPU-intensive jobs that consume more memory can be more expensive. If you use additional triggers for services as part of your task, they're billed separately.

The premium plan is suitable if you have several tasks that are short but they run continuously. This plan is more expensive because it needs more memory and CPU. As a benefit, you can use other features, such as virtual network integration.

The dedicated plan is suitable for background jobs if your workload already runs on the dedicated plan. If you have underutilized VMs, you can run the dedicated plan on the same VM and share compute costs.

For more information, see:

- Azure Functions hosting options
- Timer trigger for Azure Functions

Azure Logic Apps

Azure Logic Apps provide a powerful platform for automating workflows and integrating various services. They are particularly well-suited for hosting background jobs that require orchestration across multiple services and systems.

Azure Logic Apps has a visual designer for creating workflows, a vast library of connectors for integrating various services, event-driven triggers, stateful workflows for complex coordination, and built-in error handling and retries to ensure resilience and recovery from transient failures.

Azure Logic Apps considerations

Logic Apps works best in scenarios that don't require low latency for a response, such as asynchronous or semi long-running API calls. If low latency is required, for example in a call that blocks a user interface, use a different technology like Azure Functions or a web API deployed to Azure App Service.

Azure Logic Apps follow a pay-as-you-go pricing model, where you are charged based on the number of actions executed and the connectors used. Optimize costs by designing efficient workflows and minimizing the number of actions and connectors used in your Logic Apps.

Logic Apps automatically scale based on the number of incoming requests and the complexity of the workflows. However, be aware of the throttling limits and quotas for Logic Apps, especially if your background jobs involve high-frequency triggers or large volumes of data.

For more information, see:

- Logic App Connectors
- Limits and configuration reference for Azure Logic Apps

Virtual Machines

You can implement background tasks so that they're not deployed to Web Apps. For example, you can implement tasks by using Windows services, third-party utilities, or executable programs. You can also use programs that are written for a runtime

environment that's different than the environment that hosts the application. For example, you might use a Unix or Linux program that you want to run from a Windows or .NET application. Choose from several operating systems for an Azure VM, and run your service or executable on that VM.

For more information, see:

- Choose an Azure compute service
- Sizes for VMs in Azure
- Virtual Machines Marketplace

To initiate the background task in a separate VM, you can:

- Send a request to an endpoint that the task exposes to run the task on demand directly from your application. The request transfers data that the task requires. The endpoint invokes the task.
- Use a scheduler or timer from your chosen operating system to configure the task to run on a schedule. For example, on Windows, you can use Windows Task Scheduler to run scripts and tasks. If you have SQL Server installed on the VM, use SQL Server Agent to run scripts and tasks.
- Use Logic Apps to initiate the task by adding a message to a queue that the task monitors or by sending a request to an API that the task exposes.

For more information about how you can initiate background tasks, see the previous Triggers section.

Virtual Machines considerations

Consider the following points when you deploy background tasks in an Azure VM:

- Host background tasks in a separate Azure VM to provide flexibility and precise control over initiation, deployment, scheduling, and resource allocation. However, runtime costs increase if you deploy a VM solely for background tasks.
- There's no facility to monitor the tasks in the portal and no automated restart capability for failed tasks. But you can use the Azure Resource Manager cmdlets to monitor the status of the VM and manage it. There are no facilities to control processes and threads in compute nodes. Typically, if you use a VM, you have to implement a mechanism that collects data from instrumentation in the task, and also from the operating system in the VM. You can use the System Center Management Pack for Azure for this purpose.

- Consider creating monitoring probes that are exposed through HTTP endpoints. You can configure the code for these probes to perform health checks and collect operational information and statistics. You can also use the probes to collate error information and return it to a management application.

For more information, see:

- Health Endpoint Monitoring pattern
- Virtual Machines
- Virtual Machines FAQ

Azure Batch

Consider Batch if you need to run large, parallel high-performance computing (HPC) workloads across tens, hundreds, or thousands of VMs.

Use Batch to prepare the VMs, assign tasks to the VMs, run the tasks, monitor the progress, and automatically scale out the VMs in response to the workload. Batch also provides job scheduling and supports Linux and Windows VMs.

Batch considerations

Batch is suitable for intrinsically parallel workloads. You can use Batch to perform parallel calculations with a reduce step at the end, or run Message Passing Interface (MPI) applications for parallel tasks that require message passing between nodes.

A Batch job runs on a pool of nodes, or VMs. You can allocate a pool only when needed and then delete it after the job finishes. This approach maximizes utilization because nodes aren't idle, but the job must wait for you to allocate nodes. Alternatively, you can create a pool in advance. This approach minimizes the time that it takes for a job to start but can result in nodes that sit idle.

For more information, see:

- Nodes and pools in Batch
- What is Batch?
- Jobs and tasks in Batch
- HPC on Azure
- Batch service workflow and resources

Azure Kubernetes Service

Use AKS to manage your hosted Kubernetes environment so you can easily deploy and manage containerized applications.

Containers are useful for running background jobs. Some of the benefits include:

- Containers support high-density hosting. You can isolate a background task in a container, while placing multiple containers in each VM.
- Use the container orchestrator to perform internal load balancing, configure the internal network, and perform other configuration tasks.
- You can start and stop containers as needed.
- With Azure Container Registry, you can register your containers inside Azure boundaries to provide security, privacy, and proximity benefits.

AKS considerations

AKS requires an understanding of how to use a container orchestrator.

For more information, see:

- Overview of containers in Azure
- Introduction to container registries in Azure

Azure Container Apps

With Container Apps, you can build serverless microservices that are based on containers. Container Apps:

- Is optimized for running general purpose containers, especially applications that span many microservices that are deployed in containers.
- Is powered by Kubernetes and open-source technologies, like Dapr, Kubernetes Event-driven Autoscaling (KEDA), and Envoy.
- Supports Kubernetes-style apps and microservices with features like service discovery and traffic splitting.
- Enables event-driven application architectures by supporting scaling that's based on traffic and pulling from event sources like queues, including scale to zero.
- Supports long-running processes and can run background tasks.

Container Apps considerations

Container Apps doesn't provide direct access to the underlying Kubernetes APIs. If you require access to the Kubernetes APIs and control plane, use AKS. If you want to build Kubernetes-style applications and you don't require direct access to the native Kubernetes APIs and cluster management, use Container Apps for a fully managed experience. Container Apps is best suited for building container microservices.

For more information, see:

- Container Apps overview
- Quickstart: Deploy your first container app

Related links

- Compensating Transaction pattern
- Competing Consumers pattern
- Leader Election pattern
- Pipes and Filters pattern
- Priority Queue pattern
- Queue-Based Load Leveling pattern
- Scheduler Agent Supervisor pattern

Reliability checklist

Refer to the complete set of recommendations.

Reliability checklist

Feedback

Was this page helpful? Yes No

Design a data partitioning strategy

05/29/2025

This guide describes the recommendations for designing a data partitioning strategy for the database and data storage technology that you deploy. This strategy helps you improve the reliability of your data estate.

In many large-scale solutions, *partitions* are used to divide data so that it can be managed and accessed separately. Partitioning data improves scalability, reduces contention, and optimizes performance. Implement data partitioning to divide data by usage pattern. For example, you can archive older data in inexpensive data storage. Choose your partitioning strategy carefully to maximize the benefits and minimize adverse effects.

> **Note**
>
> In this article, the term *partitioning* means the process of physically dividing data into separate data stores. It differs from SQL Server table partitioning.

You can partition data to:

- **Improve scalability.** When you scale up a single database system, the database eventually reaches a physical hardware limit. If you divide data across multiple partitions, with each partition hosted on a separate server, you can scale out the system almost indefinitely.
- **Improve performance.** In each partition, data access operations are performed over a smaller volume of data compared to data that isn't partitioned. Partition data to make your system more efficient. Operations that affect more than one partition can run in parallel.
- **Improve security.** In some cases, you can separate sensitive and nonsensitive data into different partitions, and apply different security controls to the sensitive data.
- **Provide operational flexibility.** You can partition data to fine-tune operations, maximize administrative efficiency, and minimize cost. For example, you can define strategies for management, monitoring, backup and restore, and other administrative tasks based on the importance of the data in each partition.
- **Match the data store to the pattern of use.** You can deploy each partition on a different type of data store based on the cost and the built-in features that the data store offers. For example, you can store large binary data in blob storage, and store structured data in a document database. For more information, see Understand data store models.

- **Improve availability.** To avoid a single point of failure, you can separate data across multiple servers. If one instance fails, only the data in that partition is unavailable. Operations continue in other partitions. This consideration is less relevant for managed platform as a service (PaaS) data stores because they have built-in redundancy.

Select the right partitioning strategy

There are three typical strategies for partitioning data:

- **Horizontal partitioning** (often called *sharding*). In this strategy, each partition is a separate data store, but all partitions have the same schema. Each partition is known as a *shard* and holds a subset of the data, such as a set of customer orders.

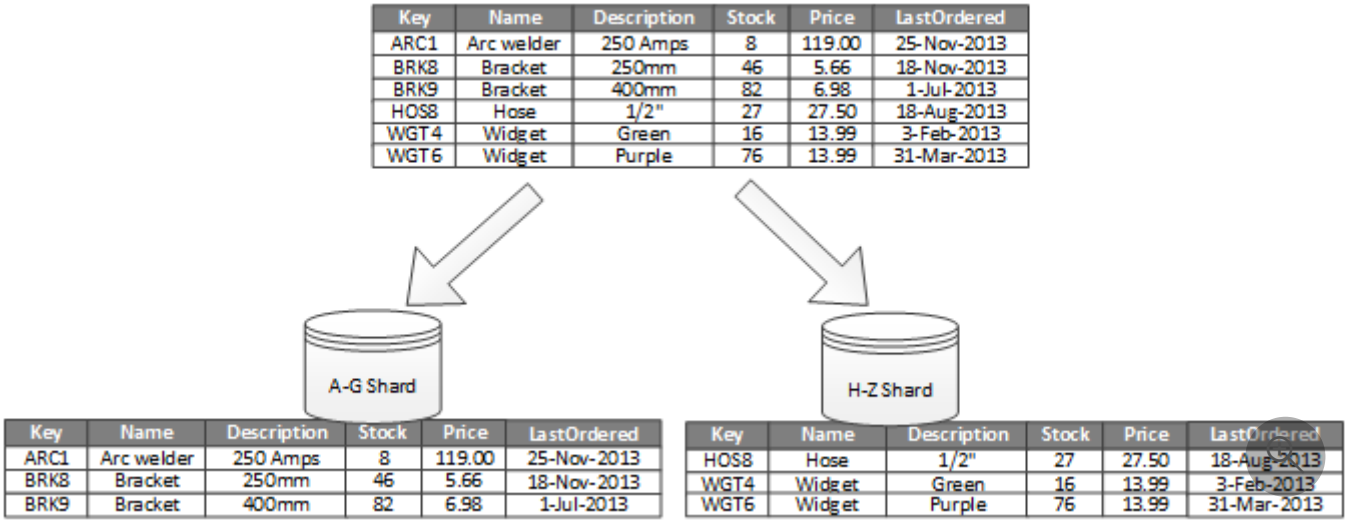
- **Vertical partitioning.** In this strategy, each partition holds a subset of the fields for items in the data store. The fields are divided according to their pattern of use. For example, frequently accessed fields might be placed in one vertical partition and less frequently accessed fields in another.

- **Functional partitioning.** In this strategy, data is aggregated according to how each bounded context in the system uses the data. For example, an e-commerce system might store invoice data in one partition and product inventory data in another.

Consider combining these strategies when you design a partitioning scheme. For example, you might divide data into shards and then use vertical partitioning to further subdivide the data in each shard.

Horizontal partitioning (sharding)

The following image shows an example of horizontal partitioning, or sharding. This example divides product inventory data into shards that are based on the product key. Each shard holds the data for a contiguous range of shard keys (A-G and H-Z), organized alphabetically. When you perform sharding, it spreads the load over more computers, which reduces contention and improves performance.



The most important factor is the sharding key that you choose. It can be difficult to change the key after the system is in operation. The key must ensure that data is partitioned to spread the workload as evenly as possible across the shards.

The shards don't have to be the same size. It's more important to balance the number of requests. Some shards might be large, but each item in the shard has a low number of access operations. Other shards might be smaller, but each item in the shard is accessed more frequently. It's also important to ensure that a single shard doesn't exceed the scale limits, in terms of capacity and processing resources, of the data store.

Avoid creating *hot* partitions that can affect performance and availability. For example, if you use the first letter of a customer's name, it can create an unbalanced distribution because some letters are more common than others. Instead, use a customer identifier hash to distribute data evenly across partitions.

Choose a sharding key that minimizes the future need to split large shards, combine small shards into larger partitions, or change the schema. These operations are time-consuming and might require you to take one or more shards offline.

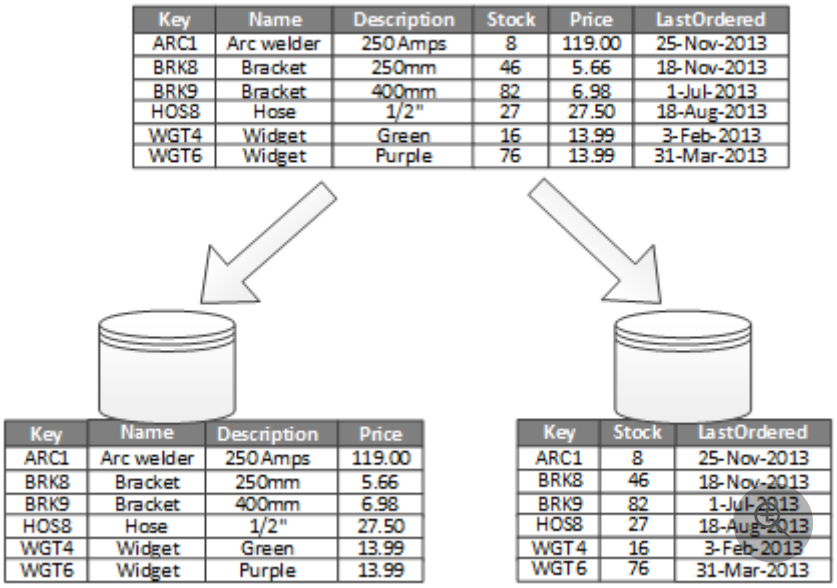
If shards are replicated, you can keep some of the replicas online while others are split, merged, or reconfigured. However, the system might limit the operations that can be performed during the reconfiguration. For example, the data in the replicas might be marked as read-only to prevent data inconsistences.

For more information, see Sharding pattern.

Vertical partitioning

The most common use for vertical partitioning is to reduce the I/O and performance costs that are associated with fetching frequently accessed items. The following image shows an example of vertical partitioning. In this example, different properties of an item are stored in different

partitions. One partition holds data that's accessed more frequently, including product name, description, and price. Another partition holds inventory data, including the stock count and the last ordered date.



In this example, the application regularly queries the product name, description, and price when it displays the product details to customers. The stock count and last ordered date are in a separate partition because these two items are commonly used together.

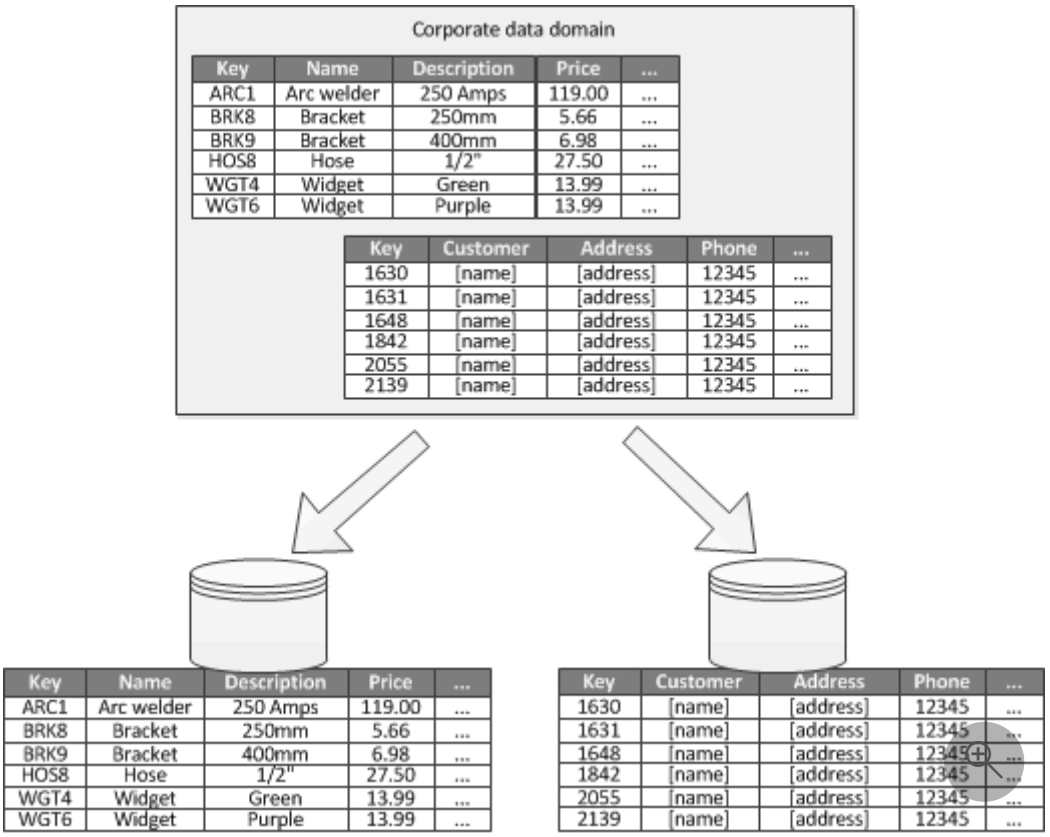
See the following advantages of vertical partitioning:

- You can separate relatively slow-moving data (product name, description, and price) from more dynamic data (stock level and last ordered date). Slow-moving data is a good candidate for an application to cache in memory.
- You can store sensitive data in a separate partition with added security controls.
- Vertical partitioning can reduce the amount of concurrent access that's needed.

Vertical partitioning operates at the entity level within a data store, partially normalizing an entity to break it down from a *wide* item to a set of *narrow* items. It's ideally suited for column-oriented data stores, such as HBase and Cassandra. If the data in a collection of columns is unlikely to change, consider using column stores in SQL Server.

Functional partitioning

When a bounded context can be identified for each distinct business area in an application, functional partitioning can improve isolation and data access performance. Another common use for functional partitioning is to separate read-write data from read-only data. The following image shows an overview of functional partitioning that has inventory data separated from customer data.



This partitioning strategy can help reduce data access contention across different parts of a system.

Design partitions for scalability

It's vital to consider the size and workload for each partition. Balance them so that data is distributed to achieve maximum scalability. However, you must also partition the data so that it doesn't exceed the scaling limits of a single partition store.

Follow these steps when you design partitions for scalability:

1. Analyze the application to understand the data access patterns, such as the size of the result set that each query returns, frequency of access, inherent latency, and server-side compute processing requirements. In many cases, a few major entities demand most of the processing resources.

2. Use this analysis to determine the current and future scalability targets, such as the data size and workload. Then distribute the data across the partitions to meet the scalability target. For horizontal partitioning, choose the right shard key to ensure even distribution. For more information, see Sharding pattern.

3. Ensure that each partition has enough resources to handle the scalability requirements in terms of the data size and throughput. Depending on the data store, there might be a limit for each partition on the amount of storage space, processing power, or network bandwidth. If the requirements are likely to exceed these limits, you might need to refine

your partitioning strategy or split data out further. You might need to combine two or more strategies.

4. Monitor the system to verify that data is distributed as expected and that the partitions can handle the load. Actual usage doesn't always match what an analysis predicts. You might have to rebalance the partitions or redesign some parts of the system to yield the required balance.

Some cloud environments allocate resources based on infrastructure boundaries. Ensure that the limits of your selected boundary provide enough room for anticipated growth in data volume, data storage, processing power, and bandwidth.

For example, if you use Azure Table Storage, there's a limit to the volume of requests that a single partition can handle in a particular period of time. For more information, see Scalability and performance targets for standard storage accounts. A busy shard might require more resources than a single partition can handle. You might need to repartition the shard to spread the load. If the total size or throughput of these tables exceeds the capacity of a storage account, you might need to create more storage accounts and spread the tables across these accounts.

Design partitions for query performance

You can boost query performance by using small datasets and running parallel queries. Each partition should contain a small proportion of the entire dataset. This reduction in volume can improve the performance of queries. However, partitioning isn't an alternative to appropriate database design and configuration. Ensure that you implement the necessary indexes.

Follow these steps when you design partitions for query performance:

1. Examine the application requirements and performance.

    - Use business requirements to determine the critical queries that must always perform quickly.
    - Monitor the system to identify queries that perform slowly.
    - Determine the queries that perform most frequently. Even if a single query has a minimal cost, cumulative resource consumption can be significant.

2. Partition the data that's causing slow performance.

    - Limit the size of each partition so that the query response time is within target.
    - If you use horizontal partitioning, design the shard key so that the application can easily select the appropriate partition. This specification prevents the query from

scanning every partition.

- Consider the location of a partition. Try to keep data in partitions that are geographically close to the applications and users that access it.

3. If an entity has throughput and query performance requirements, use functional partitioning that's based on that entity. If this allocation still doesn't satisfy the requirements, you can add horizontal partitioning. A single partitioning strategy is usually adequate, but in some cases, it's more efficient to combine both strategies.

4. Run queries in parallel across partitions to improve performance.

Design partitions for availability

Partition data to improve the availability of applications. Partitioning ensures that the entire dataset doesn't have a single point of failure, and you can independently manage individual subsets of the dataset.

Consider the following factors that affect availability:

Determine the criticality of the data. Identify the critical business data, such as transactions, and the less critical operational data, such as log files.

- Store critical data in highly available partitions, and create an appropriate backup plan.
- Establish separate management and monitoring procedures for different datasets.
- Place data that has the same level of criticality in the same partition so that it can be backed up at the same frequency. For example, you might need to back up partitions that hold transaction data more often than partitions that hold logging or trace information.

Manage individual partitions. Design partitions to support independent management and maintenance. This practice provides several advantages, for example:

- If a partition fails, it can be recovered independently without applications that access data in other partitions.
- Partitioning data by geographic area allows scheduled maintenance tasks to occur at off-peak hours for each location. Ensure that partitions aren't so large that they prevent planned maintenance from finishing during this period.

Replicate critical data across partitions. This strategy improves availability and performance but can also introduce consistency issues. It takes time to synchronize changes with every replica. During synchronization, different partitions contain different data values.

Optimize application code to use partitions

Partitioning adds complexity to the design and development of your system. Partition data as a fundamental part of your system design even if the system initially only contains a single partition. If you address partitioning as an afterthought, it's challenging because you already have a live system to maintain. You might:

- Have to modify data access logic.
- Have to migrate large quantities of existing data to distribute it across partitions.
- Run into challenges because users expect to continue using the system during the migration.

In some cases, partitioning isn't important because the initial dataset is small and a single server can easily handle it. Some workloads can go without partitions, but many commercial systems need to expand as the number of users increase.

Some small data stores also benefit from partitioning. For example, hundreds of concurrent clients might access a small data store. If you partition the data in this situation, it can help to reduce contention and improve throughput.

Consider the following points when you design a data partitioning scheme:

Minimize cross-partition data access operations. Try to keep data for the most common database operations together in a partition to minimize cross-partition data access operations. It can be more time-consuming to query across partitions rather than querying within a single partition. But optimizing partitions for one set of queries might adversely affect other sets of queries. If you must query across partitions, minimize query time by running parallel queries and aggregating the results within the application. In some cases, you can't use this approach, for example if the result from one query is used in the next query.

Replicate static reference data. If queries use relatively static reference data, such as postal code tables or product lists, consider replicating this data in all the partitions to reduce separate lookup operations in different partitions. This approach can also reduce the likelihood of the reference data becoming a *hot* dataset with heavy traffic from across the entire system. There are extra costs associated with synchronizing changes to the reference data.

Minimize cross-partition joins. Where possible, minimize requirements for referential integrity across vertical and functional partitions. In these schemes, the application is responsible for maintaining referential integrity across partitions. Queries that join data across multiple partitions are inefficient because the application typically performs consecutive queries that are based on a key and then a foreign key. Instead, consider replicating or de-normalizing the

relevant data. If cross-partition joins are necessary, run parallel queries over the partitions and join the data within the application.

Embrace eventual consistency. Evaluate whether strong consistency is a requirement. A common approach in distributed systems is to implement eventual consistency. The data in each partition is updated separately, and the application logic ensures that the updates finish successfully. The application logic also handles the inconsistencies that arise from querying data while an eventually consistent operation runs.

Consider how queries locate the correct partition. If a query must scan all partitions to locate the required data, it significantly affects performance, even when multiple parallel queries run. With vertical and functional partitioning, queries can specify the partition. On the other hand, horizontal partitioning can make locating an item difficult because every shard has the same schema. A typical solution is to maintain a map that's used to look up the shard location of items. Implement this map in the sharding logic of the application. It can also be maintained by the data store if the data store supports transparent sharding.

Rebalance shards periodically. With horizontal partitioning, rebalancing shards can help evenly distribute the data by size and workload. Rebalance shards to minimize hotspots, maximize query performance, and work around physical storage limitations. This task is complex and often requires a custom tool or process.

Replicate partitions. Replicate each partition to provide added protection against failure. If a single replica fails, queries are directed to a working copy.

Extend scalability to a different level. If you reach the physical limits of a partitioning strategy, you might need to extend the scalability to a different level. For example, if partitioning is at the database level, you might need to locate or replicate partitions in multiple databases. If partitioning is already at the database level, and there are physical limitations, you might need to locate or replicate partitions in multiple hosting accounts.

Avoid transactions that access data in multiple partitions. Some data stores implement transactional consistency and integrity for operations that modify data but only when the data is located in a single partition. If you need transactional support across multiple partitions, implement it as part of your application logic because most partitioning systems don't provide native support.

All data stores require some operational management and monitoring activity. These tasks include loading data, backing up and restoring data, reorganizing data, and ensuring that the system performs correctly and efficiently.

Consider the following factors that affect operational management:

- **Implement appropriate management and operational tasks when the data is partitioned.** These tasks might include backup and restore, archiving data, monitoring the system, and other administrative tasks. For example, it can be challenging to maintain logical consistency during backup and restore operations.
- **Load data into multiple partitions, and add new data that comes from other sources.** Some tools and utilities might not support sharded data operations, such as loading data into the correct partition.
- **Archive and delete data regularly.** To prevent the excessive growth of partitions, archive and delete data every month. You might need to transform the data to match a different archive schema.
- **Locate data integrity problems.** Consider running a periodic process to locate data integrity problems, such as data in one partition that references missing information in another. The process can either automatically attempt to fix these issues or generate a report for manual review.

Rebalance partitions

As a system matures, you might have to adjust the partitioning scheme. For example, individual partitions might start receiving a disproportionate volume of traffic and become hot, leading to excessive contention. Or you might have underestimated the volume of data in some partitions, which causes the partitions to approach capacity limits.

Some data stores, such as Azure Cosmos DB, can automatically rebalance partitions. In other cases, you can rebalance partitions in two stages:

1. Determine a new partitioning strategy.
    - Which partitions need to be split or combined?
    - What's the new partition key?
2. Migrate data from the old partitioning scheme to the new set of partitions.

You might need to make partitions unavailable while you relocate data, which is called *offline migration*. Depending on the data store, you might migrate data between partitions while they're in use. This technique is called *online migration*.

Offline migration

Offline migration reduces the chance of contention occurring. To perform offline migration:

1. Mark the partition as offline. You can mark a partition as read-only so that applications can still read the data while you move it.
2. Split-merge and move the data to the new partitions.
3. Verify the data.
4. Bring the new partitions online.
5. Remove the old partition.

Online migration

Online migration is more complex but less disruptive compared to offline migration. The process is similar to offline migration, but you don't mark the original partition as offline. Depending on the granularity of the migration process, for example item by item versus shard by shard, the data access code in the client applications might have to read and write data that's in two locations, the original partition and the new partition.

Azure facilitation

The following sections describe recommendations for partitioning data that's stored in Azure services.

Partition in Azure SQL Database

A single SQL database has a limit to the volume of data that it can contain. Throughput is constrained by architectural factors and the number of concurrent connections that it supports.

Elastic pools support horizontal scaling for a SQL database. Use elastic pools to partition your data into shards that are spread across multiple SQL databases. You can also add or remove shards as the volume of data grows and shrinks. Elastic pools can also help reduce contention by distributing the load across databases.

Each shard is implemented as a SQL database. A shard can hold more than one dataset. Each dataset is called a *shardlet*. Each database has metadata that describes the shardlets that it contains. A shardlet can be a single data item or a group of items that share the same shardlet key. For example, in a multitenant application, the shardlet key can be the tenant ID, and all data for a tenant can be in the same shardlet.

Applications are responsible for associating a dataset with a shardlet key. A separate SQL database acts as a global shard map manager. This database has a list of all the shards and shardlets in the system. The application connects to the shard map manager database to

obtain a copy of the shard map. It caches the shard map locally and uses the map to route data requests to the appropriate shard. This functionality is hidden behind a series of APIs that are contained in the client library of the Elastic Database feature of SQL Database, which is available for Java and .NET.

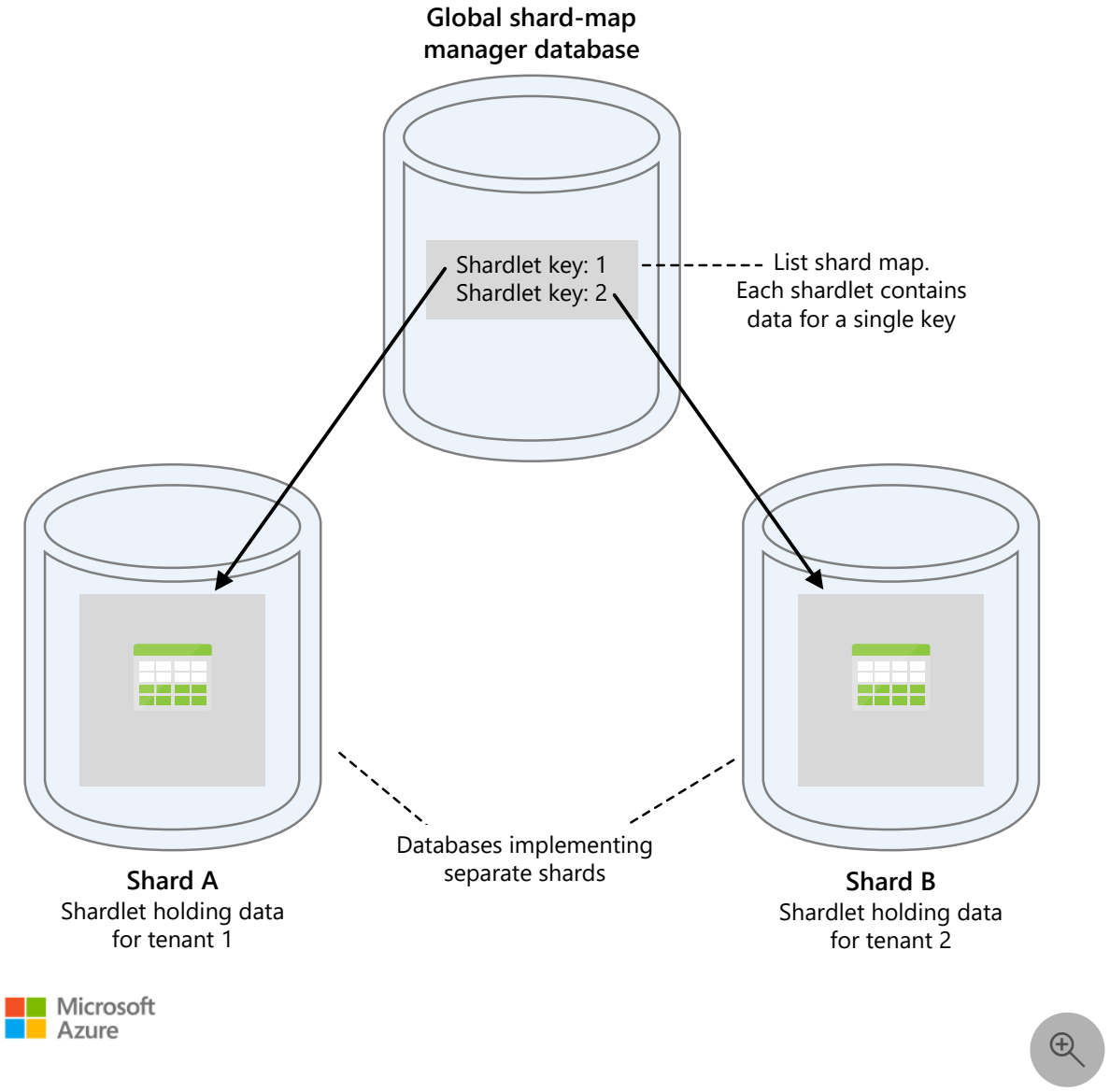
For more information about elastic pools, see Scaling out with SQL Database.

To reduce latency and improve availability, you can replicate the global shard map manager database. With the premium pricing tiers, you can configure active geo-replication to continuously copy data to databases in different regions.

Alternatively, use SQL Data Sync for SQL Database or Azure Data Factory to replicate the shard map manager database across regions. This form of replication runs periodically and is more suitable if the shard map changes infrequently and doesn't require the premium tier.

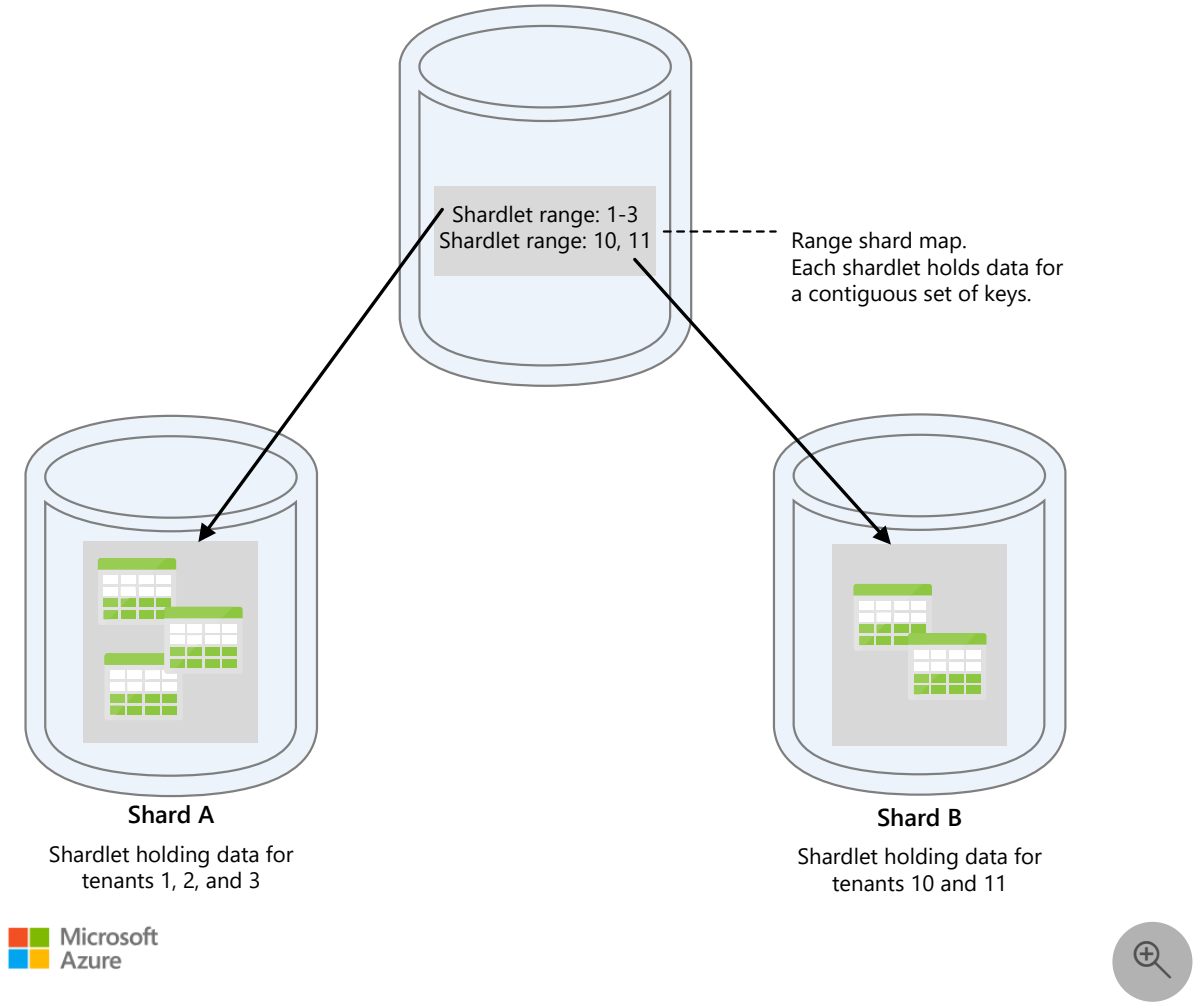
Elastic Database provides two schemes for mapping data to shardlets and storing them in shards:

- A **list shard map** associates a single key with a shardlet. For example, in a multitenant system, the data for each tenant can be associated with a unique key and stored in its own shardlet. To guarantee isolation, each shardlet can be held within its own shard.



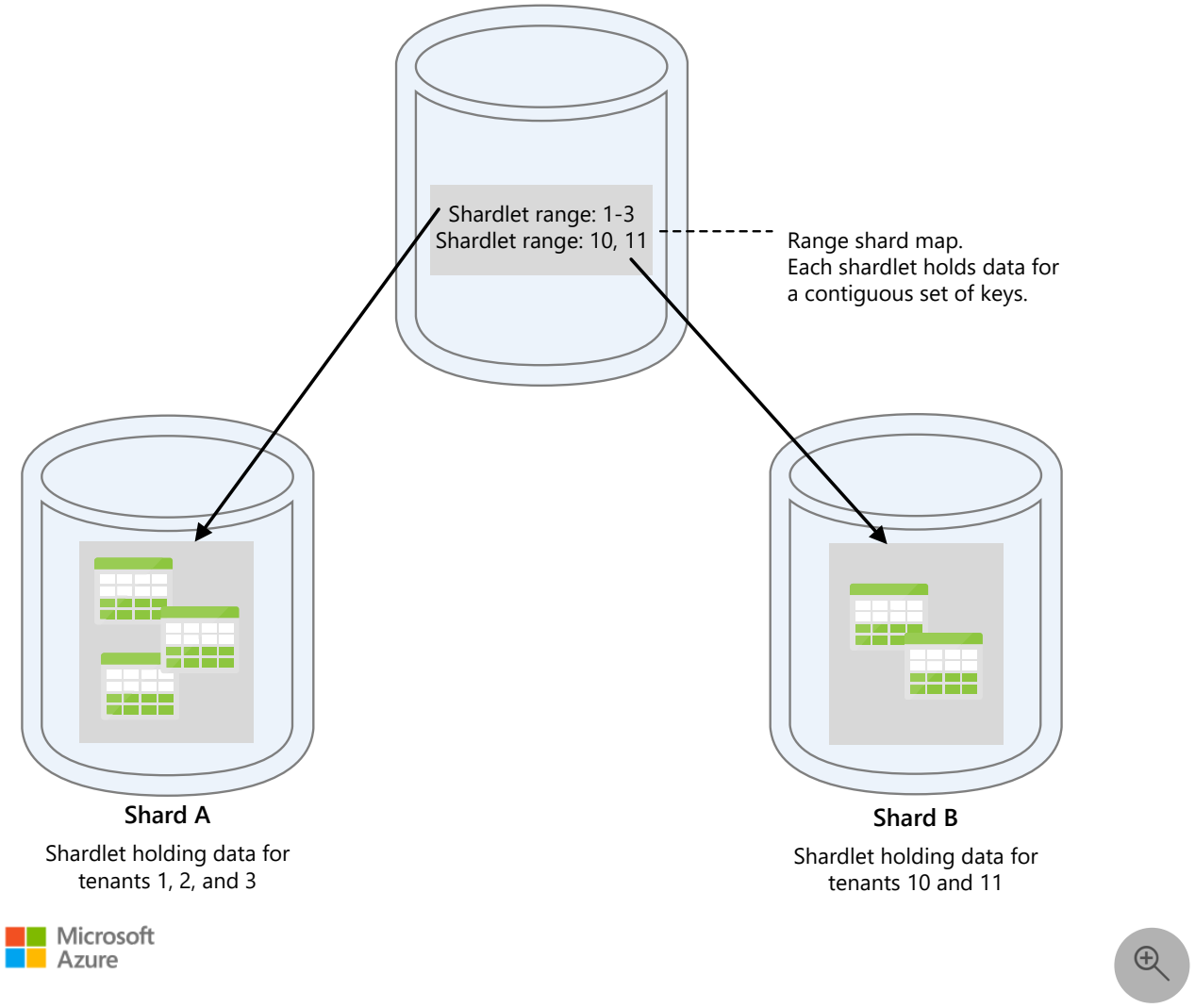
Download a Visio file of this diagram.

- A **range shard map** associates a set of contiguous key values with a shardlet. For example, you can group the data for a set of tenants, each with their own key, within the same shardlet. This scheme is less expensive than a list shard map because tenants share data storage, but it provides less isolation.



Download a Visio file of this diagram

A single shard can contain the data for several shardlets. For example, you can use list shardlets to store data for different non-contiguous tenants in the same shard. You can also mix range shardlets and list shardlets in the same shard, but then they are addressed via different maps. The following diagram shows this approach:



Download a Visio file of this diagram.

With elastic pools, you can add and remove shards as the volume of data grows and shrinks. Client applications can create and delete shards dynamically and transparently update the shard map manager. However, removing a shard is a destructive operation that also requires deleting all the data in that shard.

If an application needs to split a shard into two separate shards or combine shards, use the split-merge tool. This tool runs as an Azure web service and migrates data safely between shards.

The partitioning scheme can significantly affect the performance of your system. It can also affect the rate at which shards have to be added or removed, or that data must be repartitioned across shards. Consider the following points:

- Group data that's used together in the same shard, and avoid operations that access data from multiple shards. A shard is a SQL database in its own right, and cross-database joins must be performed on the client side when operations access multiple shards.

Although SQL Database doesn't support cross-database joins, you can use Elastic Database tools to perform multi-shard queries. A multi-shard query sends individual queries to each database and merges the results.

- Design a system that doesn't have dependencies between shards. Referential integrity constraints, triggers, and stored procedures in one database can't reference objects in another.
- Consider replicating data across shards if you have reference data that's frequently used by queries. This approach can eliminate the need to join data across databases. Ideally, such data should be static or slow-moving to minimize the replication effort and reduce the chance of it becoming stale.
- Use the same schema for shardlets that belong to the same shard map. This guidance isn't enforced by SQL Database, but data management and querying is complex if each shardlet has a different schema. Instead, create separate shard maps for each schema. You can store data that belongs to different shardlets in the same shard.
- Store data in the same shard or implement eventual consistency if your business logic needs to perform transactions. Transactional operations are only supported for data that's in a shard, and not across shards. Transactions can span shardlets if they're part of the same shard.
- Place shards close to the users that access the data in those shards. This strategy helps reduce latency.
- Avoid having a combination of highly active and relatively inactive shards. Try to spread the load evenly across shards. You might have to hash the sharding keys. If you're geo-locating shards, ensure that the hashed keys map to shardlets held in shards that are stored close to the users that access that data.

Partition in Azure Blob Storage

With Blob Storage, you can store large binary objects. Use block blobs in scenarios that require you to quickly upload or download large volumes of data. Use page blobs for applications that require random, rather than serial, access to parts of the data.

Each block blob or page blob is held in a container in an Azure storage account. Use containers to group related blobs that have the same security requirements. This grouping is logical rather than physical. Inside a container, each blob has a unique name.

The partition key for a blob is the account name, the container name, and the blob name. The partition key is used to partition data into ranges. These ranges are load balanced across the

system. Blobs can be distributed across many servers to scale out access to them. A single blob can only be served by a single server.

If your naming scheme uses timestamps or numerical identifiers, it can lead to excessive traffic to one partition. It prevents the system from effectively load balancing. For instance, if you have daily operations that use a blob object with a timestamp, such as *yyyy-mm-dd*, all the traffic for that operation goes to a single partition server. Instead, prefix the name with a three-digit hash. For more information, see Partition naming convention.

The actions of writing a single block or page are atomic, but operations that span blocks, pages, or blobs aren't. If you need to ensure consistency when write operations are performed across blocks, pages, and blobs, take out a write lock by using a blob lease.

Considerations

Data partitioning introduces some challenges and complexities that you need to consider.

- Data synchronization between the partitions might become a challenge. Ensure that updates or changes to one partition are propagated to the other partitions in a timely and consistent manner.
- Failover and disaster recovery processes become complex when you need to coordinate the backup and restore of multiple partitions. Data integrity issues can arise if some partitions or their backups are corrupted or unavailable.
- Data partitioning can affect performance and reliability if you need to query across partitions, and when you rebalance the partitions if the data grows unevenly.

Related links

- Building scalable cloud databases
- Data Factory
- Index Table pattern
- Materialized View pattern
- Moving data between scaled-out cloud databases
- Multi-shard querying using elastic database tools
- Partition naming
- Review your data options
- Scalability and performance targets for standard storage accounts
- Scaling out with SQL Database
- Sharding pattern
- Understand data store models

- Use elastic pools to manage and scale multiple databases in SQL Database
- What is SQL Data Sync for Azure?

Modernize your software development with continuous integration

06/11/2025

As code is developed, updated, or even removed, having an intuitive and safe method to integrate these changes into the main code branch enables developers to provide value.

As a developer, you can make small code changes, push these changes to a code repository, and get almost instantaneous feedback on the quality, test coverage, and introduced bugs. This process lets you work faster and with more confidence and less risk.

Continuous integration (CI) is a practice where source control systems and software deployment pipelines are integrated to provide automated build, test, and feedback mechanisms for software development teams.

The continuous integration process starts when an engineer creates a GitHub pull request to signal to the CI system that code changes are ready to be integrated. Ideally, the integration process validates the code against several baselines and tests. It then provides feedback to the requesting engineer on the status of these tests.

If baseline checks and testing go well, the integration process produces and stages assets that will deploy the updated software. These assets include compiled code and container images.

Continuous integration can help you deliver high-quality software more quickly by performing the following actions:

- Run automated tests against the code to provide early detection of breaking changes.
- Run code analysis to ensure code standards, quality, and configuration.
- Run compliance and security checks to ensure that the software has no known vulnerabilities.
- Run acceptance or functional tests to ensure that the software operates as expected.
- Provide quick feedback on detected problems.
- Where applicable, produce deployable assets or packages that include the updated code.

Automate continuous integration with pipelines

To achieve continuous integration, use software solutions to manage, integrate, and automate the process. A common practice is to use a continuous integration pipeline.

A continuous integration pipeline involves a piece of software (often cloud hosted) that provides:

- A platform for running automated tests.
- Compliance scans.
- Reporting.
- All other components that make up the continuous integration process.

In most cases, the pipeline software is attached to source control such that when pull requests are created or software is merged into a specific branch, the continuous integration pipeline is run. Source control integration also provides the opportunity to give CI feedback directly on pull requests.

Many solutions, like Azure Pipelines or GitHub Actions, provide the capabilities of continuous integration pipelines.

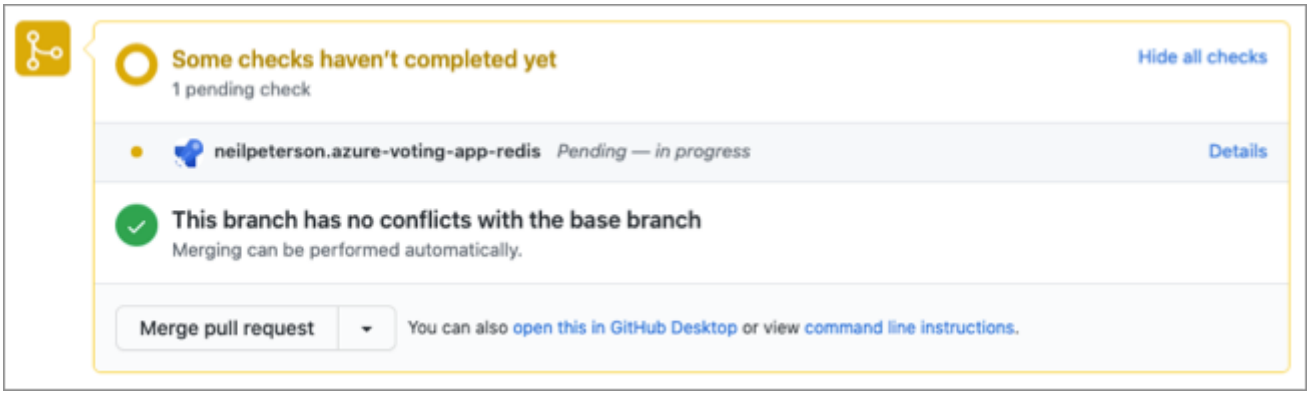
Integrate pipelines with source control

The integration of your continuous integration pipeline with your source control system is key to enabling fast, self-service code contributions.

The CI pipeline runs on a newly created pull request. The pipeline includes all tests, security assessments, and other checks. CI test results appear directly in the pull request to allow for almost real-time feedback on quality.

Another popular practice is building small reports or badges that can be presented in source control to make the current build states visible.

The following image shows the integration between GitHub and an Azure DevOps pipeline. In this example, the creation of a pull request triggers an Azure DevOps pipeline. The pipeline status appears in the pull request.



Incorporate automated tests

A key element of continuous integration is the continual building and testing of code as developers make code contributions. Testing pull requests as they're created gives quick

feedback that the commit hasn't introduced breaking changes. The advantage is that the tests in the continuous integration pipeline can be the same tests that run during test-driven development.

The following code snippet shows a test step from an Azure DevOps pipeline. The step has two tasks:

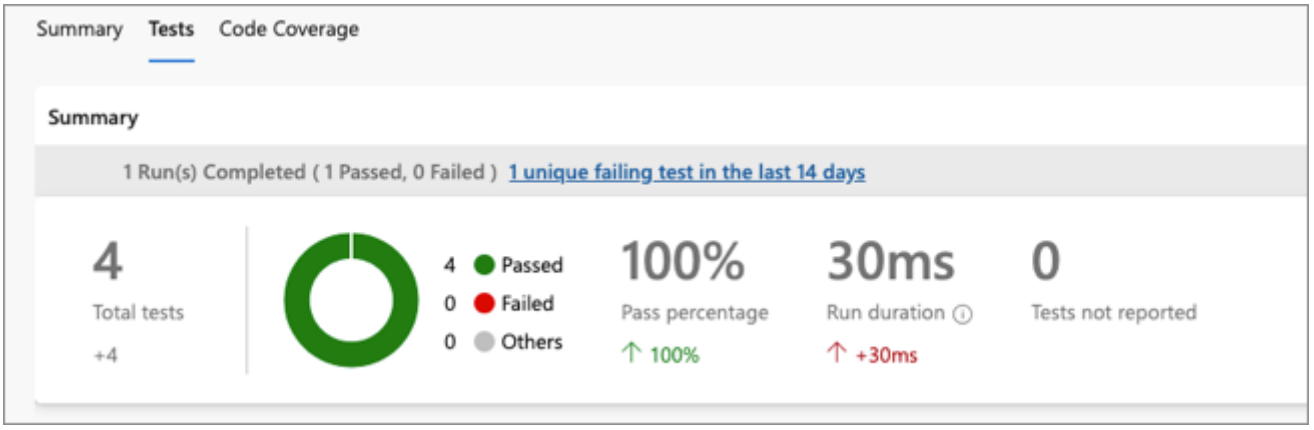
- The first task uses a popular Python testing framework to run CI tests. These tests reside in source control alongside the Python code. The test results go to a file named *test-results.xml*.
- The second task consumes the test results and publishes them to the Azure DevOps pipeline as an integrated report.

YAML

```yaml
- script: |
    pip3 install pytest
    pytest azure-vote/azure-vote/tests/ --junitxml=junit/test-results.xml
    continueOnError: true

- task: PublishTestResults@2
    displayName: 'Publish Test Results'
    inputs:
    testResultsFormat: 'JUnit'
    testResultsFiles: '**/test-results.xml'
    failTaskOnFailedTests: true
    testRunTitle: 'Python $(python.version)'
```

The following image shows test results that appear in the Azure DevOps portal.

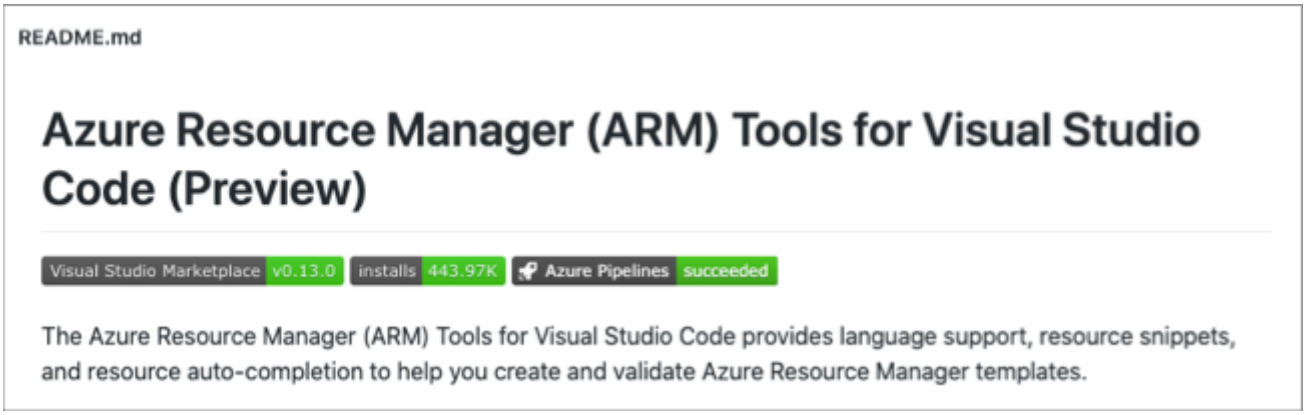


Failed tests

Failed tests should temporarily block a deployment and lead to a deeper analysis of what happened. Failed tests should also lead to either a refinement of the tests or an improvement in the change that caused the tests to fail.

Publish build status

Many developers show that their code quality is high by displaying a status badge in their repository. The following image shows an Azure Pipelines badge displayed on the readme file for an open-source project in GitHub.



Optimize build times

To perform faster builds, you can:

- **Choose agents that meet your performance requirements**: Speed up your builds by selecting the right build machines. Fast machines can make the difference between hours and minutes. If your pipelines are in Azure Pipelines, you can run your jobs by using a Microsoft-hosted agent. When you use Microsoft-hosted agents, maintenance and upgrades are taken care of for you. For more information, see Microsoft-hosted agents.

- **Optimize the build server location**: When you're building your code, data is sent across the wire. Inputs to the builds are fetched from a source control repository and the artifact repository. The output from the build process needs to be copied, including the compiled artifacts, test reports, code coverage results, and debug symbols. It's important that these copy actions are run quickly. If you use your own build server, ensure that the build server is located near the sources and a target location. Fast uploads and downloads can reduce the overall build time.

- **Scale out build servers**: A single build server might be sufficient for a small product. As the size and scope of the product and the number of teams working on the product increases, a single server might not be enough. Scale your infrastructure horizontally over multiple machines when you reach the limit. For more information, see Create and manage agent pools.

- **Optimize the build**:

- Add parallel jobs to speed up the build process. For more information, see Configure and pay for parallel jobs.
- Enable parallel test suite runs, which often save a large amount of time, especially when running integration and UI tests. For more information, see Run tests in parallel for any test runner.
- Use the notion of a multiplier, where you can scale out your builds over multiple build agents. For more information, see Specify jobs in your pipeline.
- Consider moving integration, UI, and smoke tests to a release pipeline. Moving to a release pipeline improves the build speed and the speed of the build feedback loop.
- Publish the build artifacts to a package management solution, such as NuGet or Maven. Publishing to a package management solution lets you reuse your build artifact more easily.

Implement build types to fit your workflows

Your organization might choose to create several different kinds of builds to optimize build times. Possible builds include:

- **Continuous integration (CI) build**: The purpose of this build is to ensure code is compiled and unit tests are run. This build gets triggered at each commit. It serves as the heartbeat of the project and provides quality feedback to the team im_imagestely. For more information, see Specify events that trigger pipelines.
- **Nightly build**: The purpose of a nightly build isn't only to compile the code, but also to ensure any larger test suites that are inefficient run on a regular cadence for each build. Usually, these tests include integration, UI, or smoke tests. For more information, see Configure schedules for pipelines.
- **Release build**: In addition to compiling and running tests, this build also compiles the API documentation, compliance reports, code signing, and other steps that aren't required every time the code is built. This build provides the golden copy that's pushed to the release pipeline to finally deploy in the production environment.

The types of builds needed by your organization depend on factors including your team's and organization's maturity, the kind of product you're working on, and your deployment strategy.

Related links

Learn how to create a continuous integration pipeline by using either GitHub or Azure DevOps:

- Create your first pipeline
- Use starter workflows

Learn how to display badges in your repositories:

- Add an Azure Pipelines status badge to your repository
- Add a GitHub workflow status badge to your repository