Nine Pillars of Engineering DevOps

This paper is derived from selections from this book.



The book “Engineering DevOps” can be obtained here: [**mybook.to/engineeringdevops**](http://mybook.to/engineeringdevops)

This white paper and other useful papers, glossary checklists, slides, blueprints, templates and engineering practices can be downloaded from [**https://engineeringdevops.com/documents**](https://engineeringdevops.com/documents)

About the Author:

**Marc Hornbeek** a.k.a. DevOps\_the\_Gray esq.

[linkedin.com/in/marchornbeek](https://www.linkedin.com/in/marchornbeek)

© Engineering DevOps Consulting 2019, 2020

All rights reserved

**What Is Engineering DevOps?**

***What are the Nine Pillars for Engineering DevOps?***

I like to use the pillar to describe major “immovable” structural parts of DevOps. Much like a pillar, a column of a building that helps hold the structure up, a DevOps pillar represents a permanent structural part of any well-engineered DevOps.

In my engineering framework, DevOps is organized in Nine Pillars. These are permanent structural parts of DevOps engineering, no matter where you are trying to do DevOps. As illustrated in Figure 1— Nine Pillars of DevOps, the Nine Pillars of DevOps are Leadership, Collaborative Culture, Design for DevOps, Continuous Integration, Continuous Testing, Elastic Infrastructure, Continuous Monitoring, Continuous Security and Continuous Delivery.



**Figure 1— Nine Pillars of DevOps**

Each DevOps pillar categorizes specific engineering practices that are useful to describe and evaluate DevOps practices an organization needs to engineer and operate DevOps. The tenets of the CALMS model, which is described in my white paper “What is Engineering DevOps?” are relevant to each of the pillars. In this white paper you will find descriptions of each DevOps pillar and examples of associated DevOps engineering practices. I should point out that I have used both seven-pillar and nine-pillar models from time to time. The seven-pillar model incorporates the Leadership pillar into the Collaborative Culture pillar and the Security pillar into the other pillars. I found it better to break out Leadership and Security into separate pillars because they have some distinctive practices that are worth highlighting.

You will notice the Nine Pillars of DevOps diagram includes two horizontal foundational structures (“Orchestration and Automation” and “Governance”) and three roof structures (“Continuous Delivery CI/CD Pipeline,” “Application Release Automation,” and “Value-Stream Management”). Horizontal foundational and roof structures cross the Nine Pillars because they are relevant to all the vertical pillars. “Orchestration and Automation” is the same as the “A” in CALMS.

**Leadership Pillar**

The DevOps Leadership Pillar has to do with the aptitudes, attitudes, and actions of people that have leadership roles over teams and organizations that are on a DevOps journey. The following are example practices that have been shown to correlate to well-engineered DevOps Leadership:

• Leadership demonstrates a vision for organizational direction, team direction, and a three-year horizon for team.

• Leaders intellectually stimulate the team and upend the status quo by encouraging and asking new questions and questioning the basic assumptions about the work.

• Leaders provide inspirational communication that inspires pride in being part of the team, says positive things about the team, inspires passion and motivation, and encourages people to see that change brings opportunities.

• Leaders demonstrate supportive style by considering others’ personal feelings before acting, being thoughtful of others’ personal needs and caring about individuals’ interests.

• Leaders promote personal recognition by commending teams for better-than-average work, acknowledging improvements in the quality of work, and personally complimenting individuals’ outstanding work.

**Collaborative Culture Pillar**

The DevOps Collaborative Culture Pillar has to do with teams and the organization culture within teams. The following are example engineering practices that have been shown to correlate to well-engineered DevOps:

• The culture encourages cross-functional collaboration and shared responsibilities and avoids silos between Dev, Ops and QA, and Product Management.

• The culture encourages learning from failures and cooperation between departments.

• Communication flows fluidly across the end-to-end cross- functional team using collaboration tools where appropriate (e.g., SLACK, HipChat, Yammer, etc.).

• The DevOps system is created by an expert team and reviewed by a coalition of stakeholders including Dev, Ops and QA, and Security.

• Changes to end-to-end DevOps workflows are led by an expert team and reviewed by a coalition of stakeholders including Dev, Ops, and QA.

• DevOps System changes follow a phased process to ensure the changes do not disturb the current DevOps operation. Examples of implementation phases in include: 1) Proof of Concept (POC) phase in a test environment; 2) Limited production; and 3) Deployment to all live environments.

• Key Performance Indicators (KPIs) are set and monitored by the entire team to validate the performance of the end-to-end DevOps system always. KPIs include the time for a new change to be deployed, the frequency of deliveries, and the number of times changes fail to pass the tests for any stage in the DevOps pipeline.

**Design for DevOps Pillar**

The Design for DevOps Pillar has to do with how applications software is designed. The following are example engineering practices that have been shown to correlate to well-engineered DevOps:

• Products are architected to support modular independent packaging, testing, and releases. In other words, the product itself is partitioned into modules with minimal dependencies between modules. In this way, the modules can be built, tested, and released without requiring the entire product to be built, tested, and released all at once.

• Applications are architected as modular, immutable microservices ready for deployment in cloud infrastructures in accordance with the tenets of twelve-factor apps, rather than monolithic, mutable architectures.

• Software source code changes are pre-checked with static analysis tools prior to committing to the integration branch. Static analysis tools are used to ensure the modified source code does not introduce critical software faults such as memory leaks, uninitialized variables, and array-boundary problems.

• Software code changes are pre-checked using peer code reviews prior to committing to the integration/trunk branch.

• Software code changes are pre-checked with dynamic analysis tests prior to committing to the integration/trunk branch to ensure the software performance has not degraded.

• Software changes are integrated in a private environment together with the most recent integration branch version and tested using functional testing prior to committing the software changes to the integration/trunk branch.

• Software features are tagged with software switches (i.e., feature tags or toggles) during check-in to enable selective feature level testing, promotion, and reverts.

• Automated test cases are checked in to the integration branch at the same time code changes are checked in, together with evidence that the tests passed in a pre-flight test environment.

• Developers commit their code changes regularly to trunk—at least once per day.

**Continuous Integration (CI) Pillar**

The Continuous Integration Pillar has to do with how changes to software code and build artifacts are “built” or compiled, assembled, or otherwise packaged into executable artifacts for application releases. The following are example engineering practices that have been shown to correlate to well-engineered DevOps:

• A Software Version Management (SVM) system is used to manage all source code versions (e.g., Git, GitHub, Gitlab, Bitbucket, Perforce, Mercurial, Subversion, etc.).

• An Artifact Repository system is used to manage all versions of code images changes used by the build process (e.g., JFrog Artifactory, Nexus, Helix, Archiva, etc.).

• An SVM system is used to manage all versions of pipeline tools and infrastructure-as-code configurations and tests that are used in the build process (e.g., Git, GitHub, Gitlab, Bitbucket, Perforce, Mercurial, Subversion, etc.).

• All production software changes are maintained in a single trunk or integration branch of the code.

• The software versions for supporting each customer release are maintained in a separate release branch to support software updated specific to the release.

• Every software commit automatically triggers a build process for all components of the module that has code changed by the commit.

• Once triggered, the software build process is fully automated and produces build artifacts provided the build time validations are successful.

• The automated build process checks include unit tests.

• Resources for builds are available on demand and never block a build.

• CI builds are fast enough to complete incremental builds in less than an hour.

• The build process and resources for builds scale up and down automatically according the complexity of the change. If a full build is required, the CI system automatically scales horizontally to ensure the build are completed as quickly as possible.

**Continuous Testing (CT) Pillar**

The Continuous Test Pillar (my personal favorite) has to do with how tests are used to assess software code and build artifacts changes to ensure they meet the requirements for release. The following are example engineering practices that have been shown to correlate to well-engineered DevOps:

• Development changes are “Pre-Flight” tested in a clone of the production environment prior to being integrated to the trunk branch. (Note: “production environment” means “variations of customer configurations of a product.”)

• New unit and functional regression tests that are necessary to test a software change are created together with the code and integrated into the trunk branch at the same time the code is. The new tests are then used to test the code after integration.

• “Green/Blue Deployment” methods are used to verify deployments in a staging environment before activating the environment to live.

• Release regression tests are automated. At least 85% of the tests are fully automated, and the remaining are auto-assisted if portions must be performed manually.

• Release performance tests are automated to verify that no unacceptable degradations are released.

• “Canary testing” methods are used to trial new code versions on selected live environments.

• The entire testing lifecycle, which may include Pre-Flight, Integration, Regression, Performance, and Release Acceptance tests are automatically orchestrated across the DevOps pipeline. The test suites for each phase include a pre-defined set of tests that may be selected automatically according predefined criteria.

• Test resources are scaled automatically according to the resource requirements of specific tests selected and the available time for testing.

• “A/B testing” methods are used together with Feature Toggles to trial different versions of code with customers in separate live environments.

**Elastic Infrastructure (EI) Pillar**

The Elastic Infrastructure Pillar has to do with how resource requirements (i.e., computing machines, storage and networks”) for builds and testing environments vary in near real time depending on the workload requirements to support specific changes and the variable demand of a constantly changing number of users that need to use resources. The following are example engineering practices that have been shown to correlate to well-engineered DevOps:

• The data and executable files needed for building and testing software builds and Infrastructure-as-Code are automatically archived frequently and can be reinstated on demand.

• Archives include all release and integration repositories. If an older version of a build needs to be updated, then the environment for building and testing that version can be retrieved and reinstated on demand and can be accomplished in a short time (minutes to hours).

• Build and test processes are flexible enough to automatically handle a wide variety of exceptions gracefully.

• If the build or test process for a software change is component is unable to complete, then the process for that failed change is reported and automatically scheduled for analysis but build and test processes for other changes continue.

• The reasons for and infrastructure component failure are automatically analyzed and rescheduled if the reason for the failure can be corrected by the system, but if not, then it is reported and suspended.

• System configuration management and system inventory is stored and maintained in a version-managed Configuration Management Database (CMDB).

• Infrastructure changes are managed and automated using configuration management tools that assure idempotency.

• Automated tools are used to support immutable infrastructure deployments.

 • The user performance experience of the build and test processes by different teams are consistent for all users independent of factors such as location or other factors. Monitoring tools measure user performance experience is consistent for all users in accordance with Service Level Agreements (SLAs).

• Fault recovery mechanisms are provided. Build and test system fault monitoring, fault detection, system and data monitoring, and recovery mechanisms that exist. These mechanisms are automated and are consistently verified through simulated failure conditions.

• Infrastructure failure modes are frequently tested using Chaos engineering.

• Disaster Recovery (DR) procedures are automated and periodically tested.

**Continuous Monitoring (CM) Pillar**

The Continuous Monitoring pillar refers to instrumenting, collecting, and analyzing data needed to manage health and performance of applications, databases, pipelines, and infrastructures and to engineer improvements. The following are example practices that have been shown to correlate to well-engineered DevOps:

• Deployment metrics and release gating thresholds continuously monitor all software changes.

• Example deployment metrics include the following: test completion rate 95%; test pass rate of 99%; any open critical severity bugs mitigated; MTBF S-curve is converging and shows consistent improvement over the last three test runs.

• Logging and pro-active alert systems make it easy to detect and correct DevOps system failures. Logs and proactive system alerts are in place for most DevOps component failures and are organized in a manner to quickly identify the highest-priority problems.

• Snapshot and trend results of each metric from each DevOps pipeline stage (e.g., builds, artifacts, tests) are automatically calculated in process and visible to everyone in the Development, QA, and Ops Teams.

• Key Performance Indicators (KPIs) for the DevOps infrastructure components are automatically gathered, calculated, and made visible to anyone on the team that subscribes to them.

• Example metrics for infrastructure are availability (up time) of computing resources for CI, CT, CD processes, time to complete builds, time to complete tests, number of commits that fail, and number of changes that need to be reverted due to serious failures.

• Metrics and thresholds for the DevOps infrastructure components are automatically gathered, calculated, and made visible to anyone on the team that subscribes to them.

• Process Analytics are used to monitor and determine improvements for integration, test, and release processes. Descriptive build and test analytics drive process improvements.

• Predictive analytics are used to dynamically adjust DevOps pipeline configurations. For analysis of test results, data may indicate a need to concentrate more testing in areas that have a higher failure trend.

**Continuous Security Pillar**

The Continuous Security Pillar, part of what is referred to as DevSecOps, refers to practices for integrating security of applications, databases, pipelines, and infrastructures into the continuous delivery pipeline. The following are example practices that have been shown to correlate to well-engineered DevOps.

• Developers are empowered and trained to take personal responsibility for security.

• Security assurance automation and security monitoring practices are embraced by the organization.

• All information security platforms that are in use expose full functionality via APIs for automatability.

• Proven version control practices and tools are used for all application software, scripts, templates, and blueprints that are used in DevOps environments.

• Immutable infrastructure mindset is adopted to ensure production systems are locked down.

• Security controls are automated so as not to impede DevOps agility.

• Security tools are integrated into the CI/CD pipeline.

• Source code for key intellectual property on build or test machines are only accessible by trusted users with credentials. Build and test scripts do not contain credentials to any system that has intellectual property. Intellectual property is divided such that not all of it exists on the same archive, and each archive has different credentials.

**Continuous Delivery (CD) Pillar**

The Continuous Delivery pillar refers to practices for preparing release artifacts for deployment to production. The following are example practices that have been shown to correlate to well-engineered DevOps:

• Delivery and Deployment stages are separate. The Delivery stage precedes the Deployment pipeline.

• All Deliverables that pass the Delivery metrics are packaged and prepared for Deployment using containers.

• Deliverable packages include sufficient configuration and test data to validate each deployment. Configuration Management tools are used to manage configuration information.

• Deliverables from the Delivery pipeline are automatically pushed to the Deployment pipeline once acceptable delivery measures are achieved.

• Deployment decisions are determined according to pre-determined metrics. The entire deployment process may take hours but usually lasts less than a day.

• Deployments to production environments are staged such that failed deployments can be detected early and the impact to customers can be isolated quickly.

• Deployments are arranged with automated recovery and self- healing capabilities in case a deployment fails.

**Summary**

In summary, the answer to the question: “What are the Nine Pillars of Engineering DevOps?” has been explained in this white paper.

**The Nine Pillars of DevOps is a prescriptive framework that categorizes specific engineering practices in a way that makes it very clear to engineer DevOps implementations.**  The Nine Pillars of DevOps are Leadership, Collaborative Culture, Design for DevOps, Continuous Integration, Continuous Testing, Elastic Infrastructure, Continuous Monitoring, Continuous Security and Continuous Delivery.

**Learn More**



The book “Engineering DevOps” can be obtained here: [**mybook.to/engineeringdevops**](http://mybook.to/engineeringdevops)

This white paper and other useful papers, glossary checklists, slides, blueprints, templates and engineering practices can be downloaded from <https://engineeringdevops.com/documents>

About the Author:

**Marc Hornbeek** a.k.a. DevOps\_the\_Gray esq.

[linkedin.com/in/marchornbeek](https://www.linkedin.com/in/marchornbeek)

© EngineeringDevops 2019, 2020

All rights reserved

**What Is Engineering DevOps?**