Contents

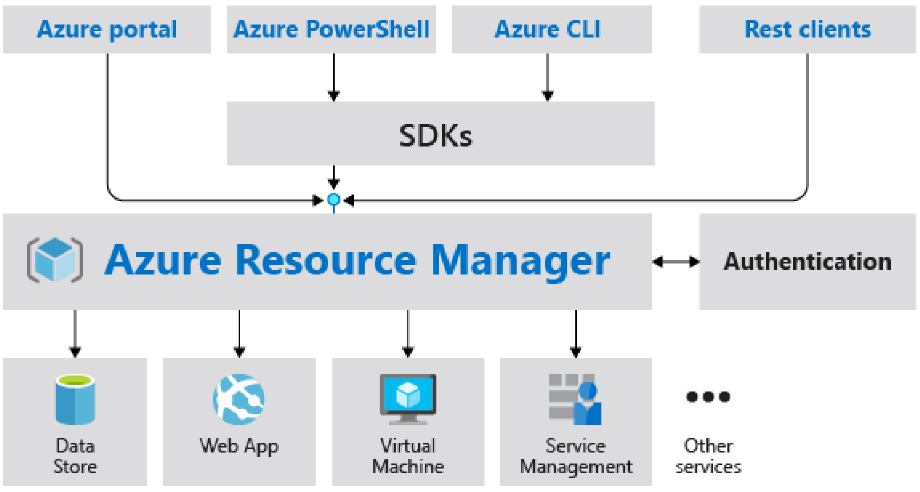
Azure Resource Manager overview

6/18/2019 • 12 minutes to read • Edit Online

Azure Resource Manager is the deployment and management service for Azure. It provides a consistent management layer that enables you to create, update, and delete resources in your Azure subscription. You can use its access control, auditing, and tagging features to secure and organize your resources after deployment.

When you take actions through the portal, PowerShell, Azure CLI, REST APIs, or client SDKs, the Azure Resource Manager API handles your request. Because all requests are handled through the same API, you see consistent results and capabilities in all the different tools. All capabilities that are available in the portal are also available through PowerShell, Azure CLI, REST APIs, and client SDKs. Functionality initially released through APIs will be represented in the portal within 180 days of initial release.

The following image shows how all the tools interact with the Azure Resource Manager API. The API passes requests to the Resource Manager service, which authenticates and authorizes the requests. Resource Manager then routes the requests to the appropriate service.



Terminology

If you're new to Azure Resource Manager, there are some terms you might not be familiar with.

- **resource** - A manageable item that is available through Azure. Virtual machines, storage accounts, web apps, databases, and virtual networks are examples of resources.
- **resource group** - A container that holds related resources for an Azure solution. The resource group includes those resources that you want to manage as a group. You decide how to allocate resources to resource groups based on what makes the most sense for your organization. See Resource groups.
- **resource provider** - A service that supplies Azure resources. For example, a common resource provider is **Microsoft.Compute**, which supplies the virtual machine resource. **Microsoft.Storage** is another common resource provider. See Resource providers.
- **Resource Manager template** - A JavaScript Object Notation (JSON) file that defines one or more resources to deploy to a resource group or subscription. The template can be used to deploy the resources consistently and repeatedly. See Template deployment.
- **declarative syntax** - Syntax that lets you state "Here is what I intend to create" without having to write the sequence of programming commands to create it. The Resource Manager template is an example of declarative syntax. In the file, you define the properties for the infrastructure to deploy to Azure.

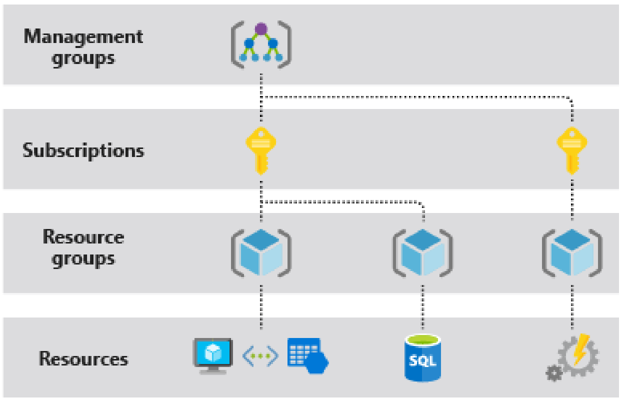
The benefits of using Resource Manager

Resource Manager provides several benefits:

- You can deploy, manage, and monitor all the resources for your solution as a group, rather than handling these resources individually.
- You can repeatedly deploy your solution throughout the development lifecycle and have confidence your resources are deployed in a consistent state.
- You can manage your infrastructure through declarative templates rather than scripts.
- You can define the dependencies between resources so they're deployed in the correct order.
- You can apply access control to all services in your resource group because Role-Based Access Control (RBAC) is natively integrated into the management platform.
- You can apply tags to resources to logically organize all the resources in your subscription.
- You can clarify your organization's billing by viewing costs for a group of resources sharing the same tag.

Understand scope

Azure provides four levels of scope: management groups, subscriptions, resource groups, and resources. The following image shows an example of these layers.



You apply management settings at any of these levels of scope. The level you select determines how widely the setting is applied. Lower levels inherit settings from higher levels. For example, when you apply a policy to the subscription, the policy is applied to all resource groups and resources in your subscription. When you apply a policy on the resource group, that policy is applied the resource group and all its resources. However, another resource group doesn't have that policy assignment.

You can deploy templates to management groups, subscriptions, or resource groups.

Guidance

The following suggestions help you take full advantage of Resource Manager when working with your solutions.

- Define and deploy your infrastructure through the declarative syntax in Resource Manager templates, rather than through imperative commands.
- Define all deployment and configuration steps in the template. You should have no manual steps for setting up your solution.
- Run imperative commands to manage your resources, such as to start or stop an app or machine.
- Arrange resources with the same lifecycle in a resource group. Use tags for all other organizing of resources.

For guidance on how enterprises can use Resource Manager to effectively manage subscriptions, see Azure enterprise scaffold - prescriptive subscription governance.

For recommendations on creating Resource Manager templates, see Azure Resource Manager template best practices.

Resource groups

There are some important factors to consider when defining your resource group:

- All the resources in your group should share the same lifecycle. You deploy, update, and delete them together. If one resource, such as a database server, needs to exist on a different deployment cycle it should be in another resource group.
- Each resource can only exist in one resource group.
- You can add or remove a resource to a resource group at any time.
- You can move a resource from one resource group to another group. For more information, see Move resources to new resource group or subscription.
- A resource group can contain resources that are located in different regions.
- A resource group can be used to scope access control for administrative actions.
- A resource can interact with resources in other resource groups. This interaction is common when the two resources are related but don't share the same lifecycle (for example, web apps connecting to a database).

When creating a resource group, you need to provide a location for that resource group. You may be wondering, "Why does a resource group need a location? And, if the resources can have different locations than the resource group, why does the resource group location matter at all?" The resource group stores metadata about the resources. Therefore, when you specify a location for the resource group, you're specifying where that metadata is stored. For compliance reasons, you may need to ensure that your data is stored in a particular region.

If the resource group's region is temporarily unavailable, you can't update resources in the resource group because the metadata is unavailable. The resources in other regions will still function as expected, but you can't update them. For more information about building reliable applications, see Designing reliable Azure applications.

Resource providers

Each resource provider offers a set of resources and operations for working with those resources. For example, if you want to store keys and secrets, you work with the **Microsoft.KeyVault** resource provider. This resource provider offers a resource type called **vaults** for creating the key vault.

The name of a resource type is in the format: **{resource-provider}/{resource-type}**. The resource type for a key vault is **Microsoft.KeyVault/vaults**.

Before getting started with deploying your resources, you should gain an understanding of the available resource providers. Knowing the names of resource providers and resources helps you define resources you want to deploy to Azure. Also, you need to know the valid locations and API versions for each resource type. For more information, see Resource providers and types.

For all the operations offered by resource providers, see the Azure REST APIs.

Template deployment

With Resource Manager, you can create a template (in JSON format) that defines the infrastructure and configuration of your Azure solution. By using a template, you can repeatedly deploy your solution throughout its lifecycle and have confidence your resources are deployed in a consistent state.

To learn about the format of the template and how you construct it, see Understand the structure and syntax of Azure Resource Manager Templates. To view the JSON syntax for resources types, see Define resources in

Azure Resource Manager templates.

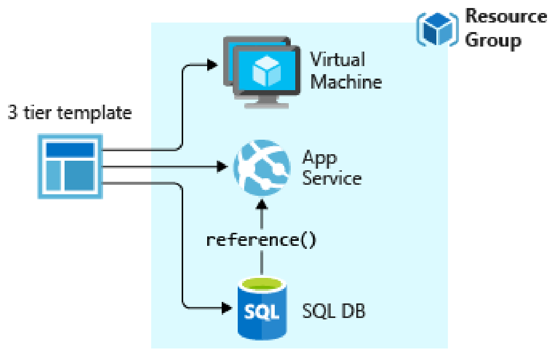
Resource Manager processes the template like any other request. It parses the template and converts its syntax into REST API operations for the appropriate resource providers. For example, when Resource Manager receives a template with the following resource definition:

```
"resources": [
  {
    "apiVersion": "2016-01-01",
    "type": "Microsoft.Storage/storageAccounts",
    "name": "mystorageaccount",
    "location": "westus",
    "sku": {
      "name": "Standard_LRS"
    },
    "kind": "Storage",
    "properties": {
    }
  }
]
```

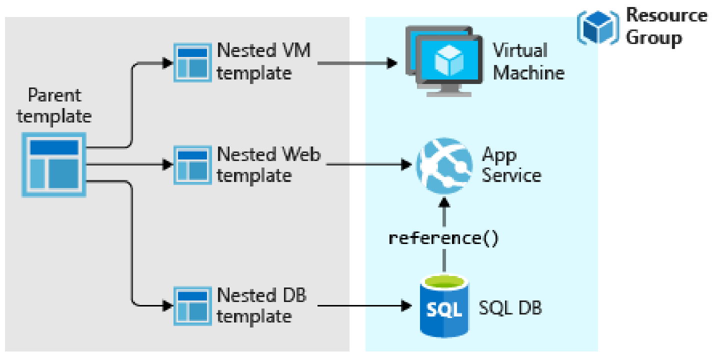
It converts the definition to the following REST API operation, which is sent to the Microsoft.Storage resource provider:

```
PUT
https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Mic
rosoft.Storage/storageAccounts/mystorageaccount?api-version=2016-01-01
REQUEST BODY
{
  "location": "westus",
  "properties": {
  }
  "sku": {
    "name": "Standard_LRS"
  },
  "kind": "Storage"
}
```

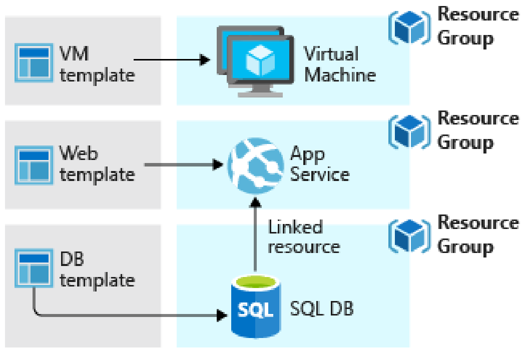
How you define templates and resource groups is entirely up to you and how you want to manage your solution. For example, you can deploy your three tier application through a single template to a single resource group.



But, you don't have to define your entire infrastructure in a single template. Often, it makes sense to divide your deployment requirements into a set of targeted, purpose-specific templates. You can easily reuse these templates for different solutions. To deploy a particular solution, you create a master template that links all the required templates. The following image shows how to deploy a three tier solution through a parent template that includes three nested templates.



If you envision your tiers having separate lifecycles, you can deploy your three tiers to separate resource groups. Notice the resources can still be linked to resources in other resource groups.



For information about nested templates, see Using linked templates with Azure Resource Manager.

Azure Resource Manager analyzes dependencies to ensure resources are created in the correct order. If one resource relies on a value from another resource (such as a virtual machine needing a storage account for disks), you set a dependency. For more information, see Defining dependencies in Azure Resource Manager templates.

You can also use the template for updates to the infrastructure. For example, you can add a resource to your solution and add configuration rules for the resources that are already deployed. If the template defines a resource that already exists, Resource Manager updates the existing resource instead of creating a new one.

Resource Manager provides extensions for scenarios when you need additional operations such as installing particular software that isn't included in the setup. If you're already using a configuration management service, like DSC, Chef or Puppet, you can continue working with that service by using extensions. For information about virtual machine extensions, see About virtual machine extensions and features.

When you create a solution from the portal, the solution automatically includes a deployment template. You don't have to create your template from scratch because you can start with the template for your solution and customize it to meet your specific needs. For a sample, see Quickstart: Create and deploy Azure Resource Manager templates by using the Azure portal. You can also retrieve a template for an existing resource group by either exporting the current state of the resource group, or viewing the template used for a particular deployment. Viewing the exported template is a helpful way to learn about the template syntax.

Finally, the template becomes part of the source code for your app. You can check it in to your source code repository and update it as your app evolves. You can edit the template through Visual Studio.

After defining your template, you're ready to deploy the resources to Azure. To deploy the resources, see:

- Deploy resources with Resource Manager templates and Azure PowerShell
- Deploy resources with Resource Manager templates and Azure CLI
- Deploy resources with Resource Manager templates and Azure portal
- Deploy resources with Resource Manager templates and Resource Manager REST API

Safe deployment practices

When deploying a complex service to Azure, you might need to deploy your service to multiple regions, and check its health before proceeding to the next step. Use Azure Deployment Manager to coordinate a staged rollout of the service. By staging the rollout of your service, you can find potential problems before it has been deployed to all regions. If you don't need these precautions, the deployment operations in the preceding section are the better option.

Deployment Manager is currently in public preview.

Resiliency of Azure Resource Manager

The Azure Resource Manager service is designed for resiliency and continuous availability. Resource Manager and control plane operations (requests sent to management.azure.com) in the REST API are:

- Distributed across regions. Some services are regional.
- Distributed across Availability Zones (as well regions) in locations that have multiple Availability Zones.
- Not dependent on a single logical data center.
- Never taken down for maintenance activities.

This resiliency applies to services that receive requests through Resource Manager. For example, Key Vault benefits from this resiliency.

Quickstarts and tutorials

Use the following quickstarts and tutorials to learn how to develop resource manager templates:

- Quickstarts

TITLE	DESCRIPTION
Use the Azure portal	Generate a template using the portal, and understand the process of editing and deploying the template.
Use Visual Studio Code	Use Visual Studio Code to create and edit templates, and how to use the Azure Cloud shell to deploy templates.
Use Visual Studio	Use Visual Studio to create, edit, and deploy templates.

- Tutorials

TITLE	DESCRIPTION
Utilize template reference	Utilize the template reference documentation to develop templates. In the tutorial, you find the storage account schema, and use the information to create an encrypted storage account.
Create multiple instances	Create multiple instances of Azure resources. In the tutorial, you create multiple instances of storage account.

TITLE	DESCRIPTION
Set resource deployment order	Define resource dependencies. In the tutorial, you create a virtual network, a virtual machine, and the dependent Azure resources. You learn how the dependencies are defined.
Use conditions	Deploy resources based on some parameter values. In the tutorial, you define a template to create a new storage account or use an existing storage account based on the value of a parameter.
Integrate key vault	Retrieve secrets/passwords from Azure Key Vault. In the tutorial, you create a virtual machine. The virtual machine administrator password is retrieved from a Key Vault.
Create linked templates	Modularize templates, and call other templates from a template. In the tutorial, you create a virtual network, a virtual machine, and the dependent resources. The dependent storage account is defined in a linked template.
Deploy virtual machine extensions	Perform post-deployment tasks by using extensions. In the tutorial, you deploy a customer script extension to install web server on the virtual machine.
Deploy SQL extensions	Perform post-deployment tasks by using extensions. In the tutorial, you deploy a customer script extension to install web server on the virtual machine.
Secure artifacts	Secure the artifacts needed to complete the deployments. In the tutorial, you learn how to secure the artifact used in the Deploy SQL extensions tutorial.
Use safe deployment practices	Use Azure Deployment manager.
Tutorial: Troubleshoot Resource Manager template deployments	Troubleshoot template deployment issues.

These tutorials can be used individually, or as a series to learn the major Resource Manager template development concepts.

Next steps

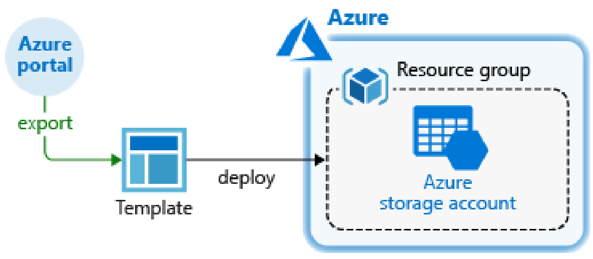
In this article, you learned how to use Azure Resource Manager for deployment, management, and access control of resources on Azure. Proceed to the next article to learn how to create your first Azure Resource Manager template.

Quickstart: Create and deploy Azure Resource Manager templates by using the Azure portal

Quickstart: Create and deploy Azure Resource Manager templates by using the Azure portal

6/12/2019 • 6 minutes to read • Edit Online

Learn how to generate a Resource Manager template using the Azure portal, and the process of editing and deploying the template from the portal. Resource Manager templates are JSON files that define the resources you need to deploy for your solution. To understand the concepts associated with deploying and managing your Azure solutions, see Azure Resource Manager overview.



After completing the tutorial, you deploy an Azure Storage account. The same process can be used to deploy other Azure resources.

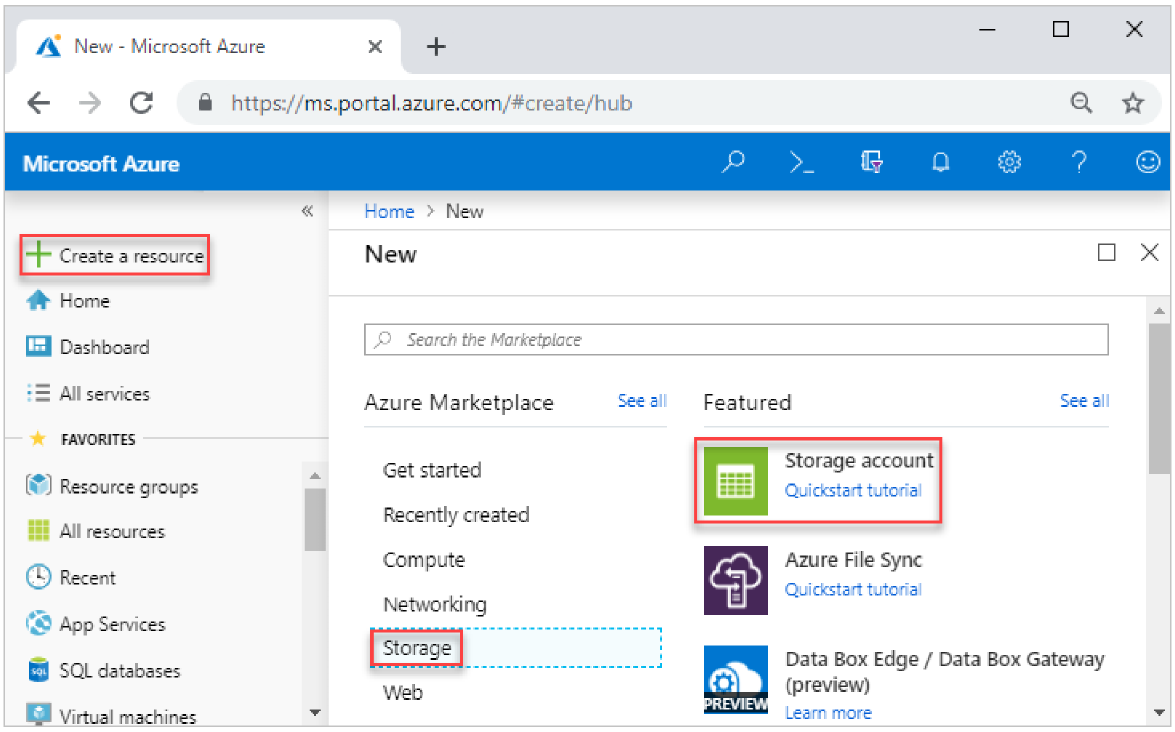
If you don't have an Azure subscription, create a free account before you begin.

Generate a template using the portal

Creating a Resource Manager template from scratch is not an easy task, especially if you are new to Azure deployment and you are not familiar with the JSON format. Using the Azure portal, you can configure a resource, for example an Azure Storage account. Before you deploy the resource, you can export your configuration into a Resource Manager template. You can save the template and reuse it in the future.

Many experienced template developers use this method to generate templates when they try to deploy Azure resources that they are not familiar with. For more information about exporting templates by using the portal, see Export resource groups to templates. The other way to find a working template is from Azure Quickstart templates.

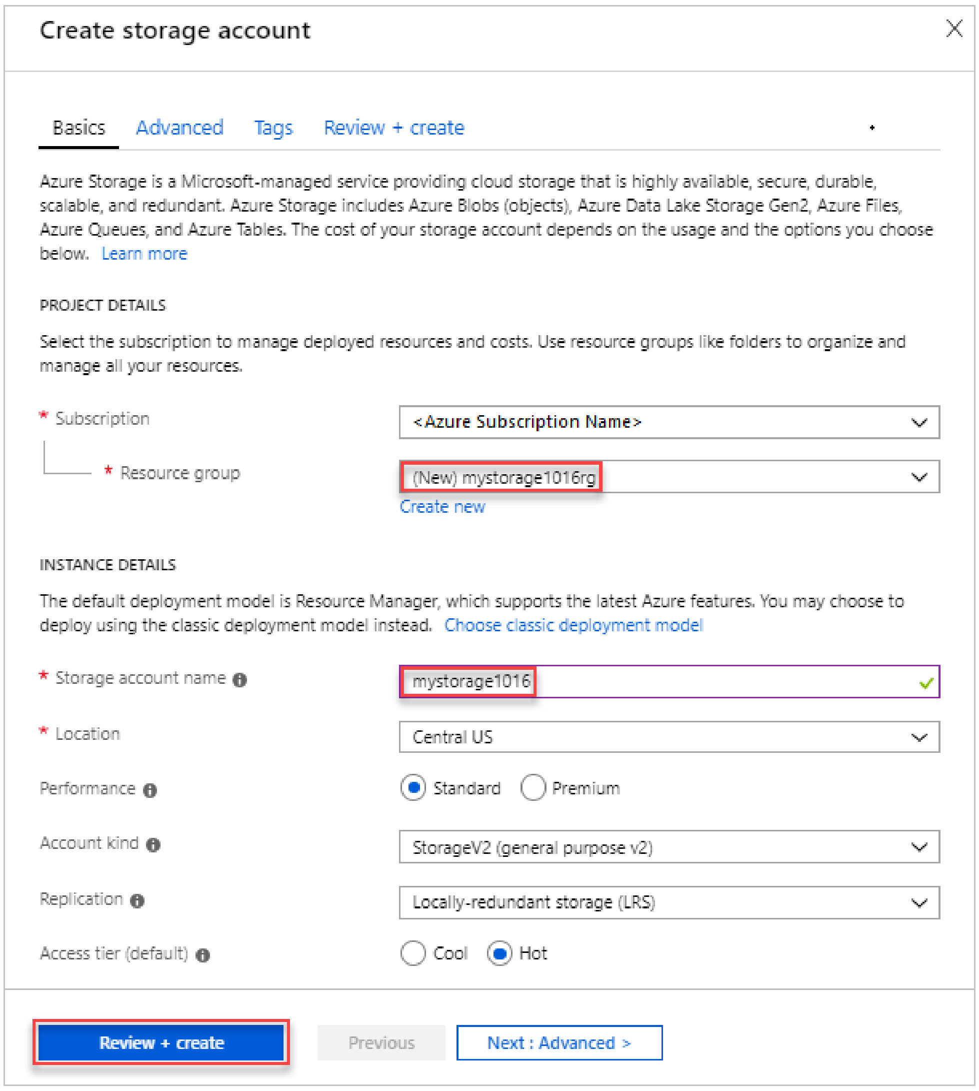
1. Sign in to the Azure portal.
2. Select **Create a resource** > **Storage** > **Storage account - blob, file, table, queue**.



3. Enter the following information:

NAME	VALUE
Resource group	Select **Create new**, and specify a resource group name of your choice. On the screenshot, the resource group name is *mystorage1016rg*. Resource group is a container for Azure resources. Resource group makes it easier to manage Azure resources.
Name	Give your storage account a unique name. The storage account name must be unique across all of Azure, and it contain only lowercase letters and numbers. Name must be between 3 and 24 characters. If you get an error message saying "The storage account name 'mystorage1016' is already taken", try using **<your name>storage<Today's date in MMDD>**, for example **johndolestorage1016**. For more information, see Naming rules and restrictions.

You can use the default values for the rest of the properties.


Create storage account

Basics | Advanced | Tags | Review + create

Azure Storage is a Microsoft-managed service providing cloud storage that is highly available, secure, durable, scalable, and redundant. Azure Storage includes Azure Blobs (objects), Azure Data Lake Storage Gen2, Azure Files, Azure Queues, and Azure Tables. The cost of your storage account depends on the usage and the options you choose below. Learn more

PROJECT DETAILS

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

* Subscription	<Azure Subscription Name>
* Resource group	(New) mystorage1016rg
	Create new

INSTANCE DETAILS

The default deployment model is Resource Manager, which supports the latest Azure features. You may choose to deploy using the classic deployment model instead. Choose classic deployment model

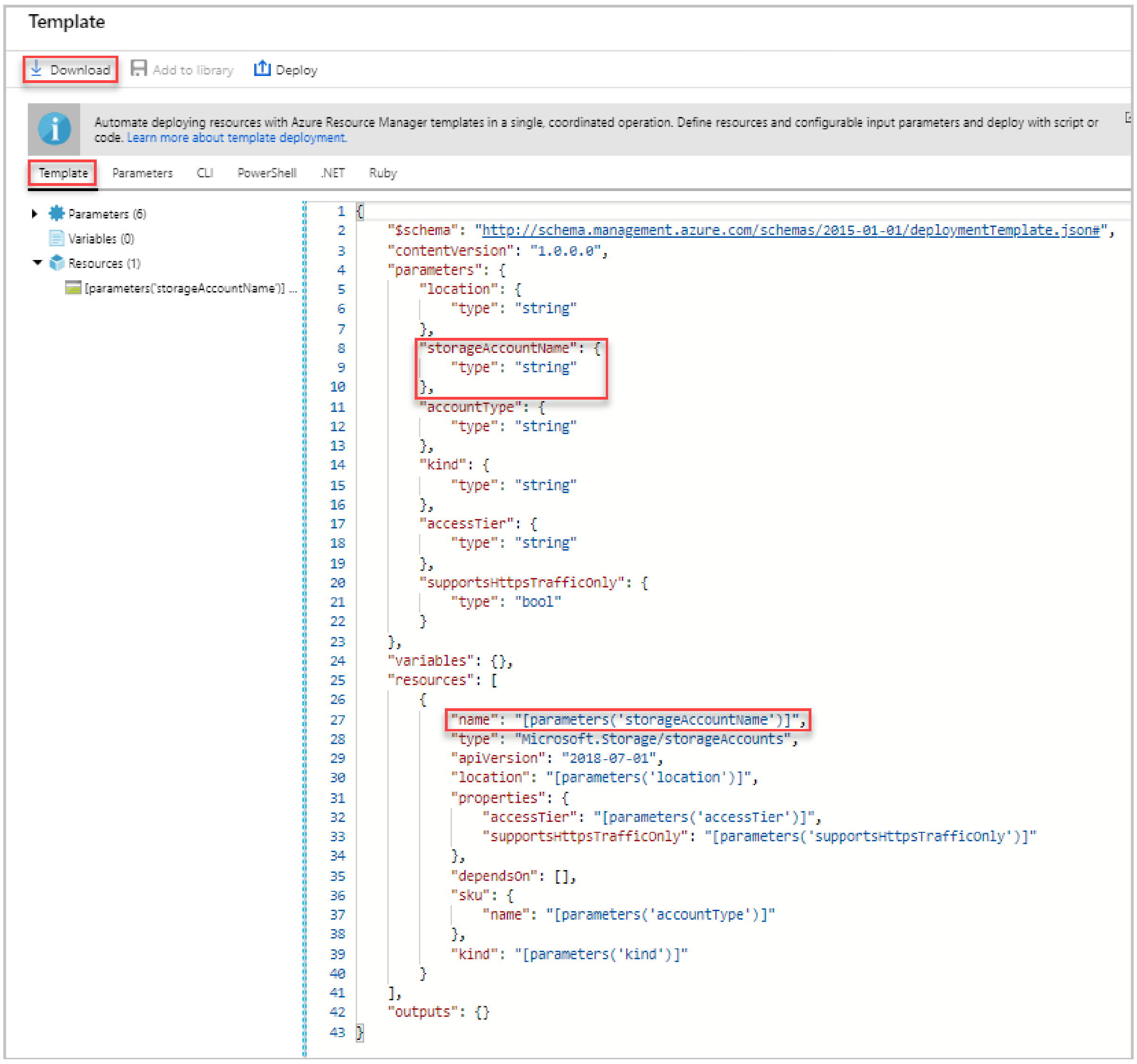
* Storage account name	mystorage1016
* Location	Central US
Performance	[✓ Standard] Premium
Account kind	StorageV2 (general purpose v2)
Replication	Locally-redundant storage (LRS)
Access tier (default)	Cool [✓ Hot]

NOTE

Some of the exported templates require some edits before you can deploy them.

4. Select **Review + create** on the bottom of the screen. Do not select **Create** in the next step.

5. Select **Download a template for automation** on the bottom of the screen. The portal shows the generated template:



Automate deploying resources with Azure Resource Manager templates in a single, coordinated operation. Define resources and configurable input parameters and deploy with script or code. Learn more about template deployment.

```json
{
    "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "location": {
            "type": "string"
        },
        "storageAccountName": {
            "type": "string"
        },
        "accountType": {
            "type": "string"
        },
        "kind": {
            "type": "string"
        },
        "accessTier": {
            "type": "string"
        },
        "supportsHttpsTrafficOnly": {
            "type": "bool"
        }
    },
    "variables": {},
    "resources": [
        {
            "name": "[parameters('storageAccountName')]",
            "type": "Microsoft.Storage/storageAccounts",
            "apiVersion": "2018-07-01",
            "location": "[parameters('location')]",
            "properties": {
                "accessTier": "[parameters('accessTier')]",
                "supportsHttpsTrafficOnly": "[parameters('supportsHttpsTrafficOnly')]"
            },
            "dependsOn": [],
            "sku": {
                "name": "[parameters('accountType')]"
            },
            "kind": "[parameters('kind')]"
        }
    ],
    "outputs": {}
}
```

The main pane shows the template. It is a JSON file with six top-level elements - `schema`, `contentVersion`, `parameters`, `variables`, `resources`, and `output`. For more information, see Understand the structure and syntax of Azure Resource Manager Templates

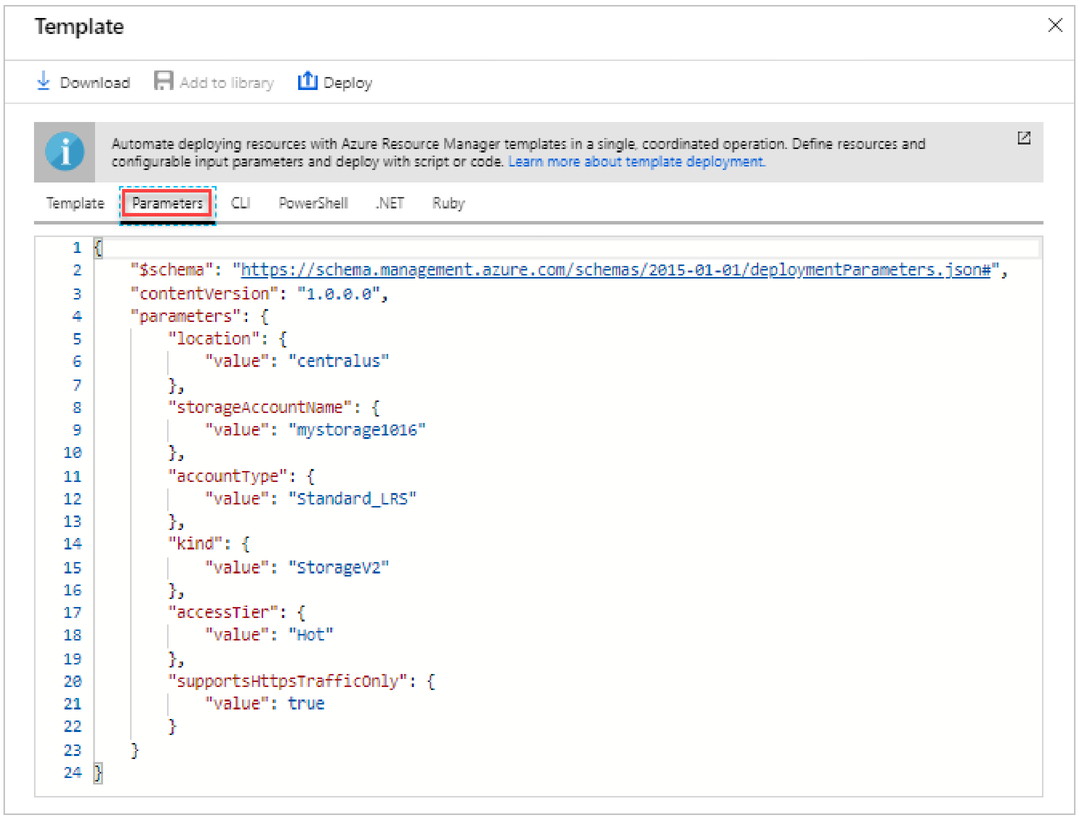
There are six parameters defined. One of them is called **storageAccountName**. The second highlighted part on the previous screenshot shows how to reference this parameter in the template. In the next section, you edit the template to use a generated name for the storage account.

In the template, one Azure resource is defined. The type is `Microsoft.Storage/storageAccounts`. Take a look of how the resource is defined, and the definition structure.

6. Select **Download** from the top of the screen.

7. Open the downloaded zip file, and then save **template.json** to your computer. In the next section, you use a template deployment tool to edit the template.

8. Select the **Parameter** tab to see the values you provided for the parameters. Write down these values, you need them in the next section when you deploy the template.


Template
Download
Add to library
Deploy
Automate deploying resources with Azure Resource Manager templates in a single, coordinated operation. Define resources and configurable input parameters and deploy with script or code. Learn more about template deployment.
Template
Parameters
CLI
PowerShell
.NET
Ruby

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "location": {
            "value": "centralus"
        },
        "storageAccountName": {
            "value": "mystorage1016"
        },
        "accountType": {
            "value": "Standard_LRS"
        },
        "kind": {
            "value": "StorageV2"
        },
        "accessTier": {
            "value": "Hot"
        },
        "supportsHttpsTrafficOnly": {
            "value": true
        }
    }
}
```

Using both the template file and the parameters file, you can create a resource, in this tutorial, an Azure storage account.

Edit and deploy the template

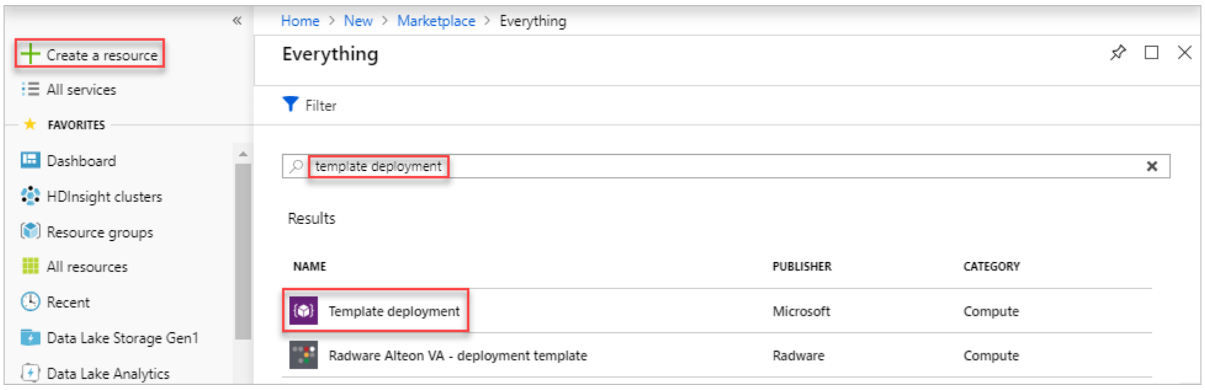
The Azure portal can be used to perform some basic template editing. In this quickstart, you use a portal tool called *Template Deployment*. *Template Deployment* is used in this tutorial so you can complete the whole tutorial using one interface - the Azure portal. To edit a more complex template, consider using Visual Studio Code, which provides richer edit functionalities.

IMPORTANT

Template Deployment provides an interface for testing simple templates. It is not recommended to use this feature in production. Instead, store your templates in an Azure storage account, or a source code repository like GitHub.

Azure requires that each Azure service has a unique name. The deployment could fail if you entered a storage account name that already exists. To avoid this issue, you modify the template to use a template function call `uniquestring()` to generate a unique storage account name.

1. In the Azure portal, select **Create a resource**.
2. In **Search the Marketplace**, type **template deployment**, and then press **ENTER**.
3. Select **Template deployment**.



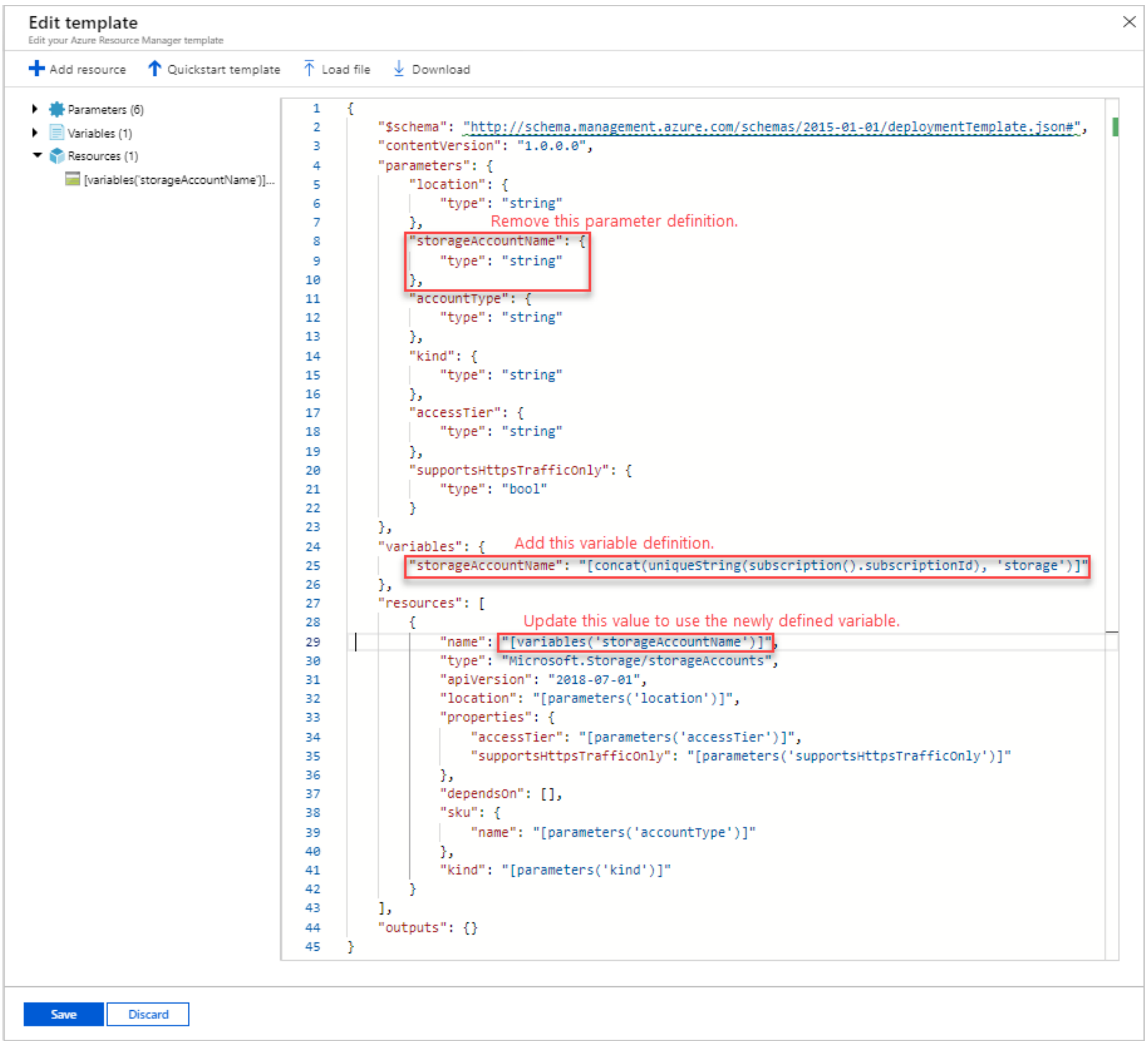
NAME	PUBLISHER	CATEGORY
Template deployment	Microsoft	Compute
Radware Alteon VA - deployment template	Radware	Compute

4. Select **Create**.

5. Select **Build your own template in the editor**.

6. Select **Load file**, and then follow the instructions to load template.json you downloaded in the last section.

7. Make the following three changes to the template:


Edit template
Edit your Azure Resource Manager template
Add resource
Quickstart template
Load file
Download
Parameters (6)
Variables (1)
Resources (1)
[variables('storageAccountName')]...
Remove this parameter definition.
Add this variable definition.
Update this value to use the newly defined variable.
Save
Discard

```
{
    "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "location": {
            "type": "string"
        },
        "storageAccountName": {
            "type": "string"
        },
        "accountType": {
            "type": "string"
        },
        "kind": {
            "type": "string"
        },
        "accessTier": {
            "type": "string"
        },
        "supportsHttpsTrafficOnly": {
            "type": "bool"
        }
    },
    "variables": {
        "storageAccountName": "[concat(uniqueString(subscription().subscriptionId), 'storage')]"
    },
    "resources": [
        {
            "name": "[variables('storageAccountName')]",
            "type": "Microsoft.Storage/storageAccounts",
            "apiVersion": "2018-07-01",
            "location": "[parameters('location')]",
            "properties": {
                "accessTier": "[parameters('accessTier')]",
                "supportsHttpsTrafficOnly": "[parameters('supportsHttpsTrafficOnly')]"
            },
            "dependsOn": [],
            "sku": {
                "name": "[parameters('accountType')]"
            },
            "kind": "[parameters('kind')]"
        }
    ],
    "outputs": {}
}
```

- Remove the **storageAccountName** parameter as shown in the previous screenshot.
- Add one variable called **storageAccountName** as shown in the previous screenshot:

```
"storageAccountName": "[concat(uniqueString(subscription().subscriptionId), 'storage')]"
```

Two template functions are used here: `concat()` and `uniqueString()`.

- Update the name element of the **Microsoft.Storage/storageAccounts** resource to use the newly defined variable instead of the parameter:

  ```
  "name": "[variables('storageAccountName')]",
  ```

  The final template shall look like:

  ```
  {
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "location": {
            "type": "string"
        },
        "accountType": {
            "type": "string"
        },
        "kind": {
            "type": "string"
        },
        "accessTier": {
            "type": "string"
        },
        "supportsHttpsTrafficOnly": {
            "type": "bool"
        }
    },
    "variables": {
        "storageAccountName": "[concat(uniqueString(subscription().subscriptionId), 'storage')]"
    },
    "resources": [
        {
            "name": "[variables('storageAccountName')]",
            "type": "Microsoft.Storage/storageAccounts",
            "apiVersion": "2018-07-01",
            "location": "[parameters('location')]",
            "properties": {
                "accessTier": "[parameters('accessTier')]",
                "supportsHttpsTrafficOnly": "[parameters('supportsHttpsTrafficOnly')]"
            },
            "dependsOn": [],
            "sku": {
                "name": "[parameters('accountType')]"
            },
            "kind": "[parameters('kind')]"
        }
    ],
    "outputs": {}
  }
  ```

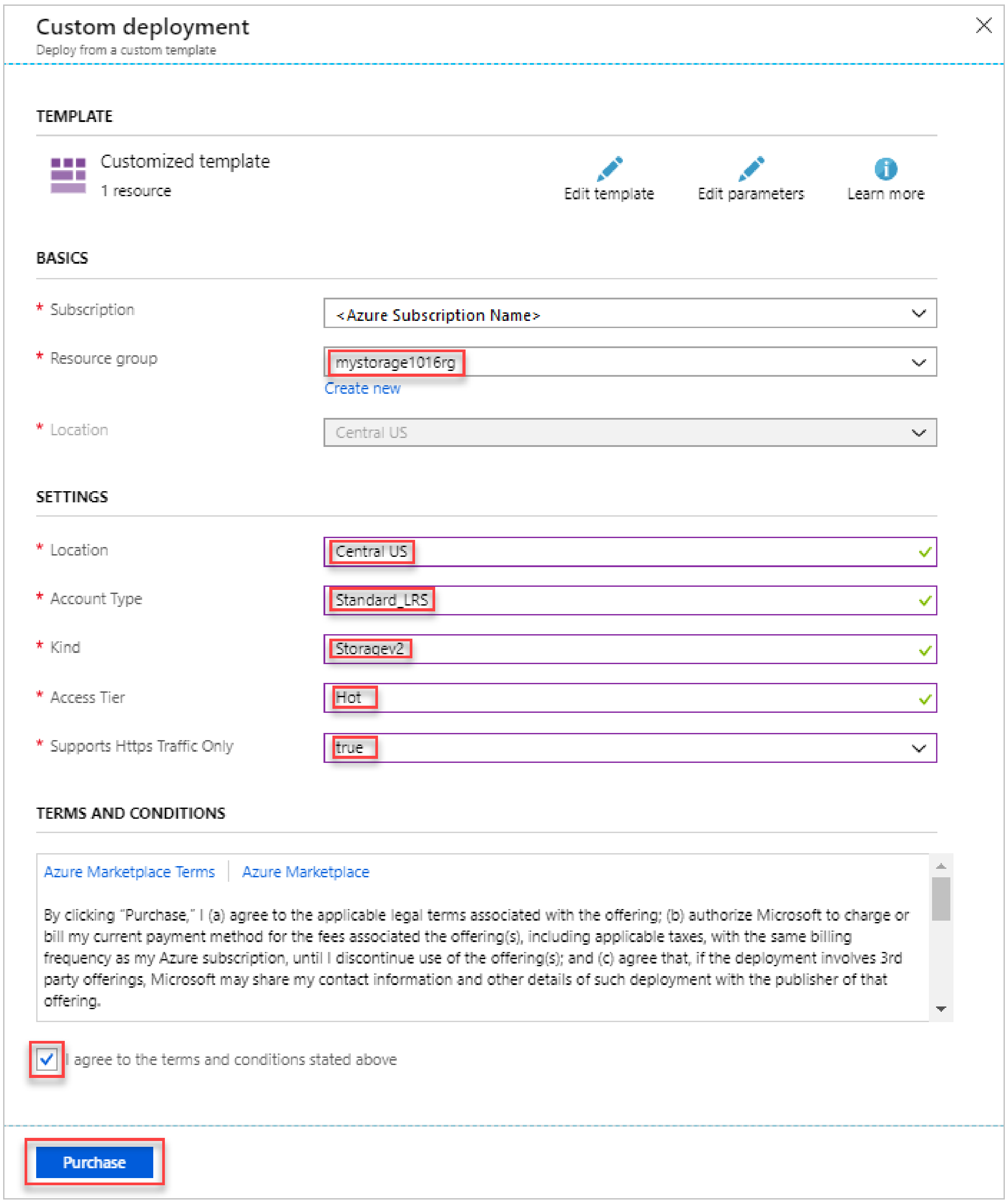
8. Select **Save**.

9. Enter the following values:

NAME	VALUE
Resource group	Select the resource group name you created in the last section.
Location	Select a location for the storage account. For example, **Central US**.

NAME	VALUE
Account Type	Enter **Standard_LRS** for this quickstart.
Kind	Enter **StorageV2** for this quickstart.
Access Tier	Enter **Hot** for this quickstart.
Https Traffic Only Enabled	Select **true** for this quickstart.
I agree to the terms and conditions stated above	(select)

Here is a screenshot of a sample deployment:



Custom deployment
Deploy from a custom template

TEMPLATE

Customized template
1 resource

BASICS

* Subscription	<Azure Subscription Name>
* Resource group	mystorage1016rg
	Create new
* Location	Central US

SETTINGS

* Location	Central US
* Account Type	Standard_LRS
* Kind	Storagev2
* Access Tier	Hot
* Supports Https Traffic Only	true

TERMS AND CONDITIONS

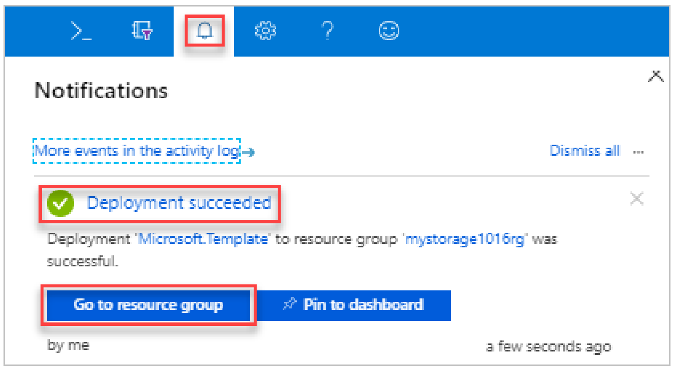
Azure Marketplace Terms | Azure Marketplace

By clicking "Purchase," I (a) agree to the applicable legal terms associated with the offering; (b) authorize Microsoft to charge or bill my current payment method for the fees associated the offering(s), including applicable taxes, with the same billing frequency as my Azure subscription, until I discontinue use of the offering(s); and (c) agree that, if the deployment involves 3rd party offerings, Microsoft may share my contact information and other details of such deployment with the publisher of that offering.

[✓] I agree to the terms and conditions stated above

10. Select **Purchase**.

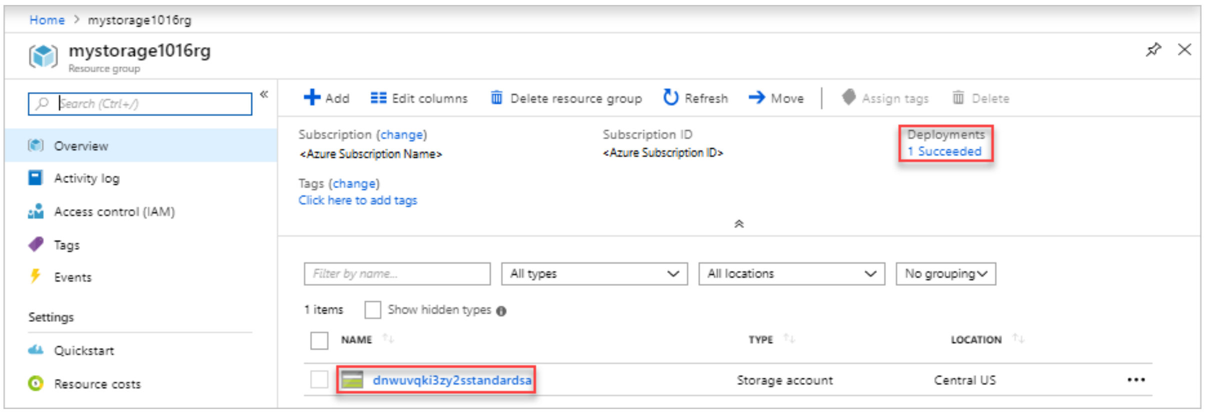
11. Select the bell icon (notifications) from the top of the screen to see the deployment status. You shall see **Deployment in progress**. Wait until the deployment is completed.



Deployment 'Microsoft.Template' to resource group 'mystorage1016rg' was successful.

by me a few seconds ago

12. Select **Go to resource group** from the notification pane. You shall see a screen similar to:



Subscription (change)
<Azure Subscription Name>

Subscription ID
<Azure Subscription ID>

Deployments
1 Succeeded

Tags (change)
Click here to add tags

1 items

NAME	TYPE	LOCATION
dnwuvqki3zy2sstandardsa	Storage account	Central US

You can see the deployment status was successful, and there is only one storage account in the resource group. The storage account name is a unique string generated by the template. To learn more about using Azure storage accounts, see Quickstart: Upload, download, and list blobs using the Azure portal.

Clean up resources

When the Azure resources are no longer needed, clean up the resources you deployed by deleting the resource group.

1. In the Azure portal, select **Resource group** on the left menu.
2. Enter the resource group name in the **Filter by name** field.
3. Select the resource group name. You shall see the storage account in the resource group.
4. Select **Delete resource group** in the top menu.

Next steps

In this tutorial, you learned how to generate a template from the Azure portal, and how to deploy the template using the portal. The template used in this Quickstart is a simple template with one Azure resource. When the template is complex, it is easier to use Visual Studio Code or Visual Studio to develop the template. The next quickstart also shows you how to deploy templates using Azure PowerShell and Azure Command-line Interface (CLI).

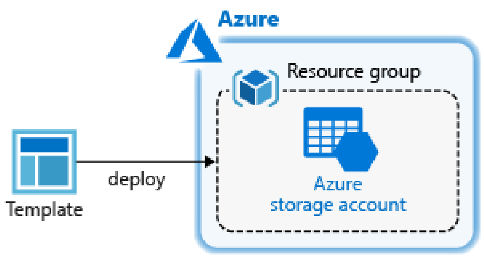
Create templates by using Visual Studio Code

Quickstart: Create Azure Resource Manager templates by using Visual Studio Code

3/4/2019 • 4 minutes to read • Edit Online

Learn how to use Visual Studio code and the Azure Resource Manager Tools extension to create and edit Azure Resource Manager templates. You can create Resource Manager templates in Visual Studio Code without the extension, but the extension provides autocomplete options that simplify template development. To understand the concepts associated with deploying and managing your Azure solutions, see Azure Resource Manager overview.

In this tutorial, you deploy a storage account:



If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

To complete this article, you need:

- Visual Studio Code.
- Resource Manager Tools extension. To install, use these steps:
  1. Open Visual Studio Code.
  2. Press **CTRL+SHIFT+X** to open the Extensions pane
  3. Search for **Azure Resource Manager Tools**, and then select **Install**.
  4. Select **Reload** to finish the extension installation.

Open a Quickstart template

Instead of creating a template from scratch, you open a template from Azure Quickstart Templates. Azure QuickStart Templates is a repository for Resource Manager templates.

The template used in this quickstart is called Create a standard storage account. The template defines an Azure Storage account resource.

1. From Visual Studio Code, select **File**>**Open File**.
2. In **File name**, paste the following URL:

```
https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-storage-account-
create/azuredeploy.json
```

3. Select **Open** to open the file.

4. Select **File**>**Save As** to save the file as **azuredeploy.json** to your local computer.

Edit the template

To experience how to edit a template using Visual Studio Code, you add one more element into the `outputs` section to show the storage URI.

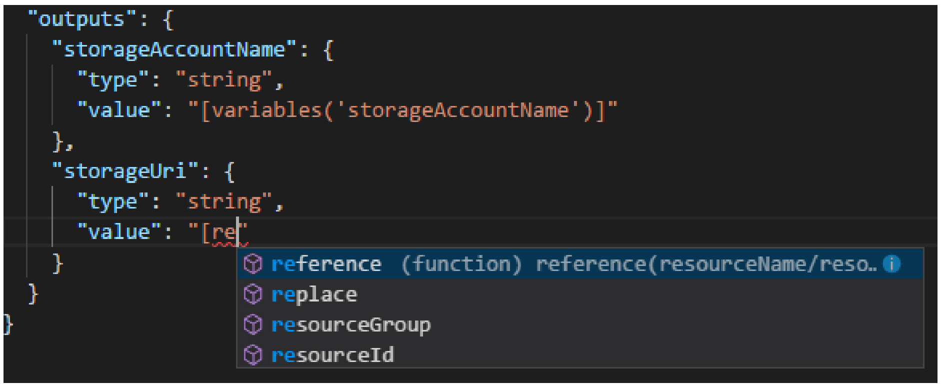
1. Add one more output to the exported template:

```
"storageUri": {
  "type": "string",
  "value": "[reference(variables('storageAccountName')).primaryEndpoints.blob]"
}
```

When you are done, the outputs section looks like:

```
"outputs": {
  "storageAccountName": {
    "type": "string",
    "value": "[variables('storageAccountName')]"
  },
  "storageUri": {
    "type": "string",
    "value": "[reference(variables('storageAccountName')).primaryEndpoints.blob]"
  }
}
```

If you copied and pasted the code inside Visual Studio Code, try to retype the **value** element to experience the IntelliSense capability of the Resource Manager Tools extension.



```
"outputs": {
  "storageAccountName": {
    "type": "string",
    "value": "[variables('storageAccountName')]"
  },
  "storageUri": {
    "type": "string",
    "value": "[re"
  }
}
}
```

2. Select **File**>**Save** to save the file.

Deploy the template

There are many methods for deploying templates. Azure Cloud shell is used in this quickstart. The cloud shell supports both Azure CLI and Azure PowerShell. Use the tab selector to choose between CLI and PowerShell.

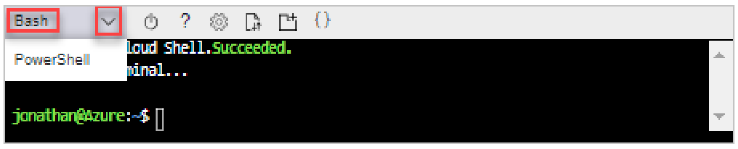
NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

1. Sign in to the Azure Cloud shell

2. Choose your preferred environment by selecting either **PowerShell** or **Bash**(CLI) on the upper left corner. Restarting the shell is required when you switch.

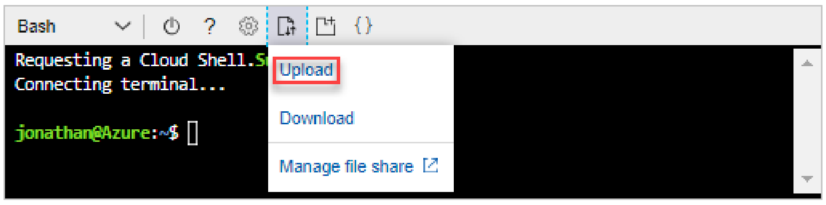
   - CLI
   - PowerShell



```
loud Shell.Succeeded.
minal...

jonathan@Azure:~$
```

3. Select **Upload/download files**, and then select **Upload**.

   - CLI
   - PowerShell

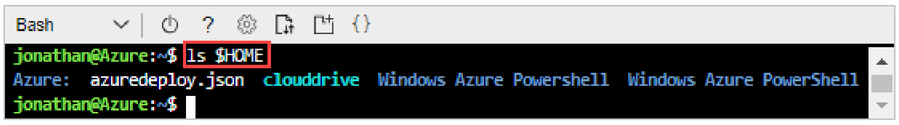


```
Requesting a Cloud Shell.S
Connecting terminal...

jonathan@Azure:~$
```

   Select the file you saved in the previous section. The default name is **azuredeploy.json**. The template file must be accessible from the shell.

   You can optionally use the **ls** command and the **cat** command to verify the file is uploaded successfully.

   - CLI
   - PowerShell



```
jonathan@Azure:~$ ls $HOME
Azure:  azuredeploy.json  clouddrive  Windows Azure Powershell  Windows Azure PowerShell
jonathan@Azure:~$
```

4. From the Cloud shell, run the following commands. Select the tab to show the PowerShell code or the CLI code.

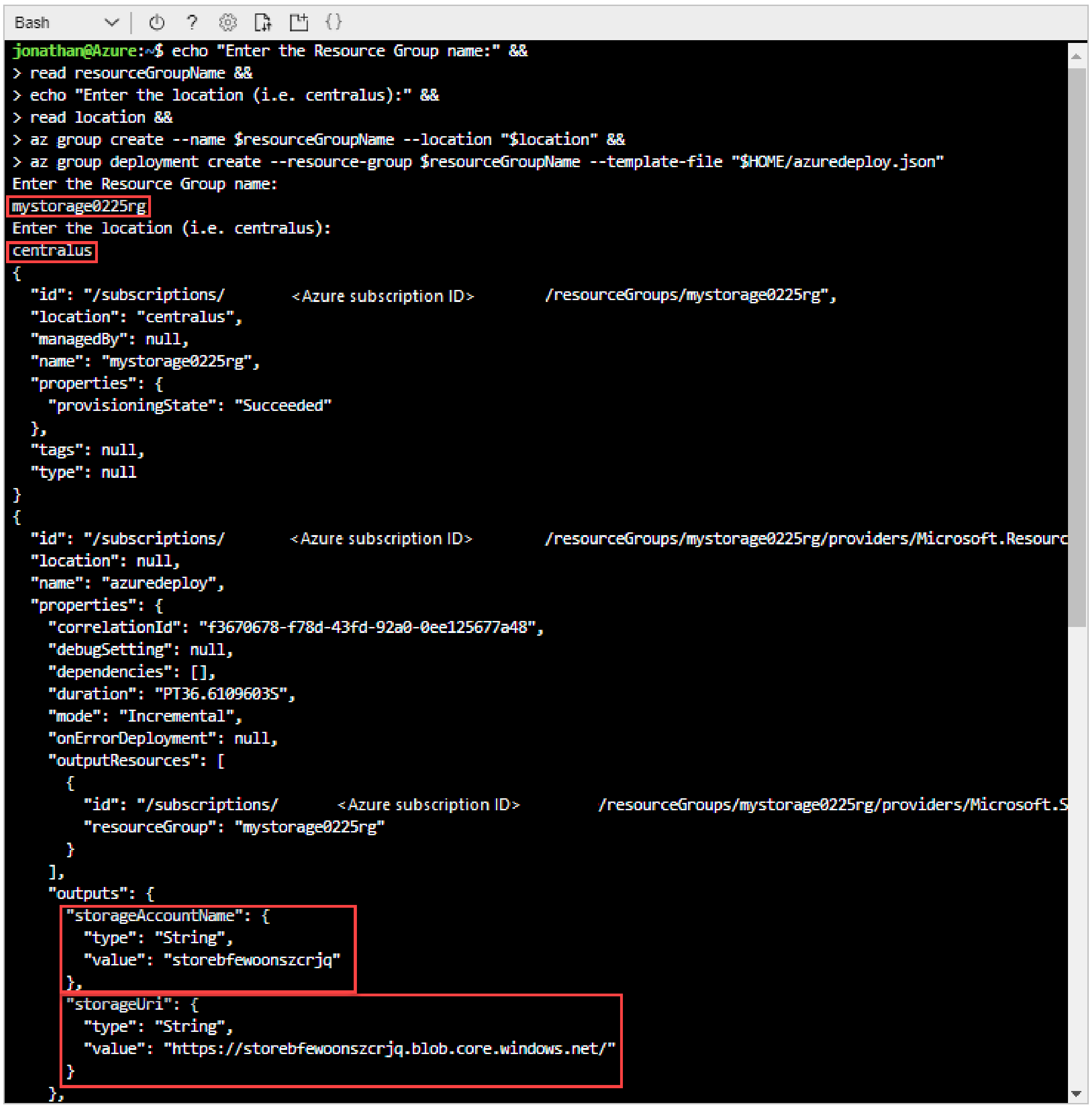
   - CLI
   - PowerShell

```
echo "Enter the Resource Group name:" &&
read resourceGroupName &&
echo "Enter the location (i.e. centralus):" &&
read location &&
az group create --name $resourceGroupName --location "$location" &&
az group deployment create --resource-group $resourceGroupName --template-file "$HOME/azuredeploy.json"
```

   Update the template file name if you save the file to a name other than **azuredeploy.json**.

   The following screenshot shows a sample deployment:

   - CLI
   - PowerShell



```
jonathan@Azure:~$ echo "Enter the Resource Group name:" &&
> read resourceGroupName &&
> echo "Enter the location (i.e. centralus):" &&
> read location &&
> az group create --name $resourceGroupName --location "$location" &&
> az group deployment create --resource-group $resourceGroupName --template-file "$HOME/azuredeploy.json"
Enter the Resource Group name:
mystorage0225rg
Enter the location (i.e. centralus):
centralus
{
  "id": "/subscriptions/      <Azure subscription ID>       /resourceGroups/mystorage0225rg",
  "location": "centralus",
  "managedBy": null,
  "name": "mystorage0225rg",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null,
  "type": null
}
{
  "id": "/subscriptions/      <Azure subscription ID>       /resourceGroups/mystorage0225rg/providers/Microsoft.Resourc
  "location": null,
  "name": "azuredeploy",
  "properties": {
    "correlationId": "f3670678-f78d-43fd-92a0-0ee125677a48",
    "debugSetting": null,
    "dependencies": [],
    "duration": "PT36.6109603S",
    "mode": "Incremental",
    "onErrorDeployment": null,
    "outputResources": [
      {
        "id": "/subscriptions/      <Azure subscription ID>       /resourceGroups/mystorage0225rg/providers/Microsoft.S
        "resourceGroup": "mystorage0225rg"
      }
    ],
    "outputs": {
      "storageAccountName": {
        "type": "String",
        "value": "storebfewoonszcrjq"
      },
      "storageUri": {
        "type": "String",
        "value": "https://storebfewoonszcrjq.blob.core.windows.net/"
      }
    },
```

The storage account name and the storage URL in the outputs section are highlighted on the screenshot. You need the storage account name in the next step.

5. Run the following CLI or PowerShell command to list the newly created storage account:

- CLI
- PowerShell

```
echo "Enter the Resource Group name:" &&
read resourceGroupName &&
echo "Enter the Storage Account name:" &&
read storageAccountName &&
az storage account show --resource-group $resourceGroupName --name $storageAccountName
```

To learn more about using Azure storage accounts, see Quickstart: Upload, download, and list blobs using the Azure portal.

Clean up resources

When the Azure resources are no longer needed, clean up the resources you deployed by deleting the resource group.

1. From the Azure portal, select **Resource group** from the left menu.

2. Enter the resource group name in the **Filter by name** field.
3. Select the resource group name. You shall see a total of six resources in the resource group.
4. Select **Delete resource group** from the top menu.

Next steps

The main focus of this quickstart is to use Visual Studio Code to edit an existing template from Azure Quickstart templates. You also learned how to deploy the template using either CLI or PowerShell from the Azure Cloud shell. The templates from Azure Quickstart templates might not give you everything you need. The next tutorial shows you how to find the information from template reference so you can create an encrypted Azure Storage account.

Create an encrypted storage account

Creating and deploying Azure resource groups through Visual Studio

6/21/2019 • 10 minutes to read • Edit Online

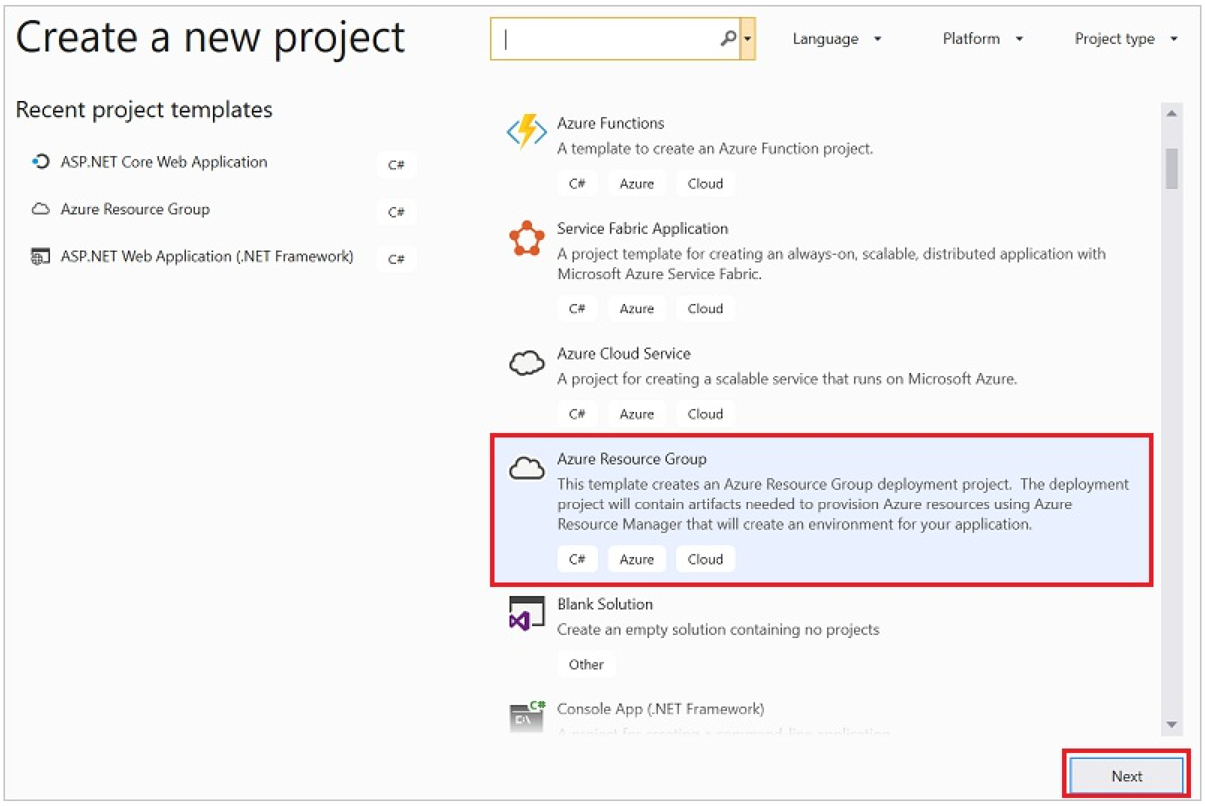
With Visual Studio, you can create a project that deploys your infrastructure and code to Azure. For example, you can deploy the web host, web site, and code for the web site. Visual Studio provides many different starter templates for deploying common scenarios. In this article, you deploy a web app.

This article shows how to use Visual Studio 2019 or later with the Azure development and ASP.NET workloads installed. If you use Visual Studio 2017, your experience is largely the same.

Create Azure Resource Group project

In this section, you create an Azure Resource Group project with a **Web app** template.

1. In Visual Studio, choose **File**, **New**, and **Project**. Select the **Azure Resource Group** project template and **Next**.



Create a new project

Recent project templates

- ASP.NET Core Web Application C#
- Azure Resource Group C#
- ASP.NET Web Application (.NET Framework) C#

Azure Functions
A template to create an Azure Function project.
C# Azure Cloud

Service Fabric Application
A project template for creating an always-on, scalable, distributed application with Microsoft Azure Service Fabric.
C# Azure Cloud

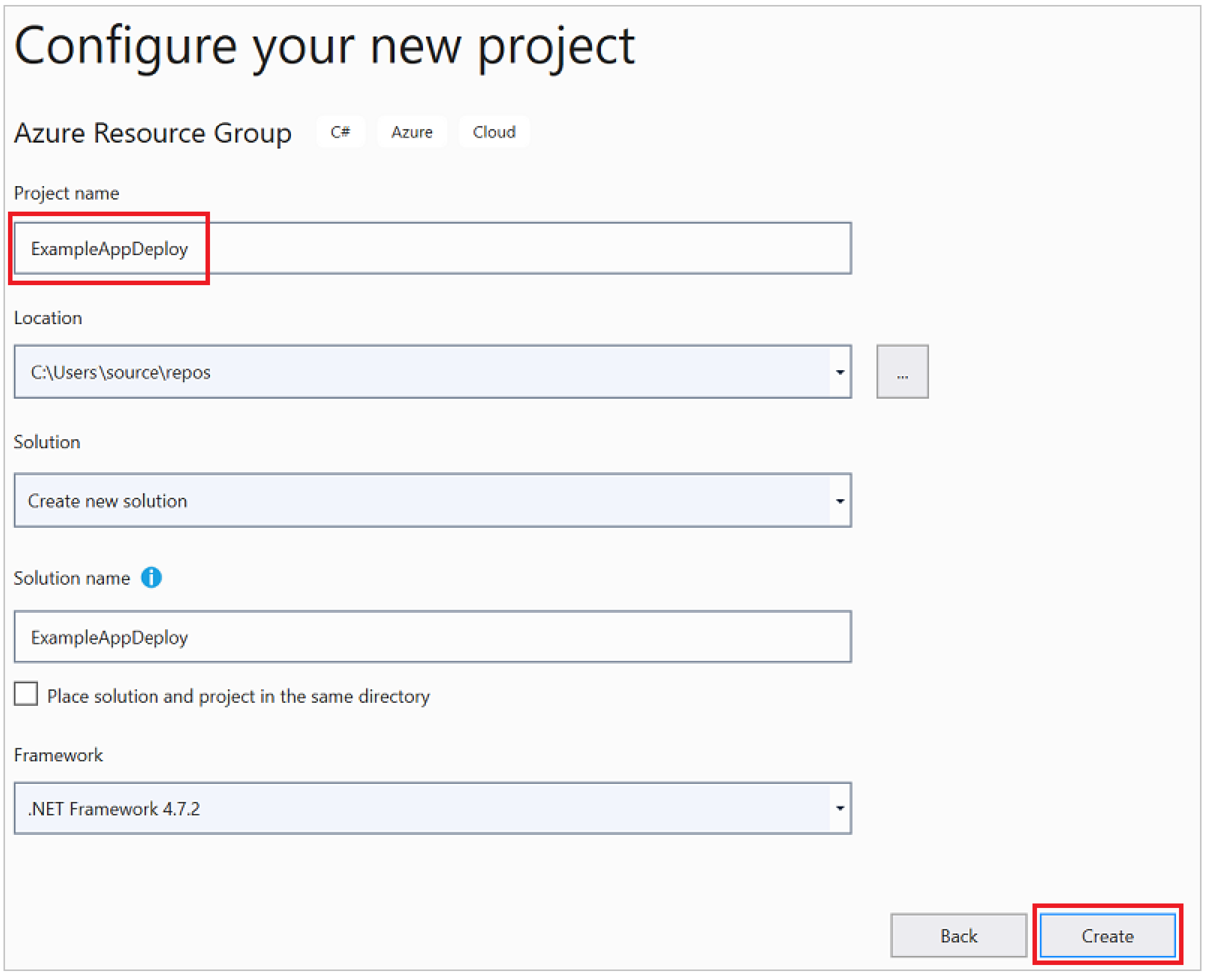
Azure Cloud Service
A project for creating a scalable service that runs on Microsoft Azure.
C# Azure Cloud

Azure Resource Group
This template creates an Azure Resource Group deployment project. The deployment project will contain artifacts needed to provision Azure resources using Azure Resource Manager that will create an environment for your application.
C# Azure Cloud

Blank Solution
Create an empty solution containing no projects
Other

Console App (.NET Framework)

2. Give your project a name. The other default settings are probably fine, but review them to make they work for your environment. When done, select **Create**.



Configure your new project

Azure Resource Group   C#   Azure   Cloud

Project name: ExampleAppDeploy

Location: C:\Users\source\repos

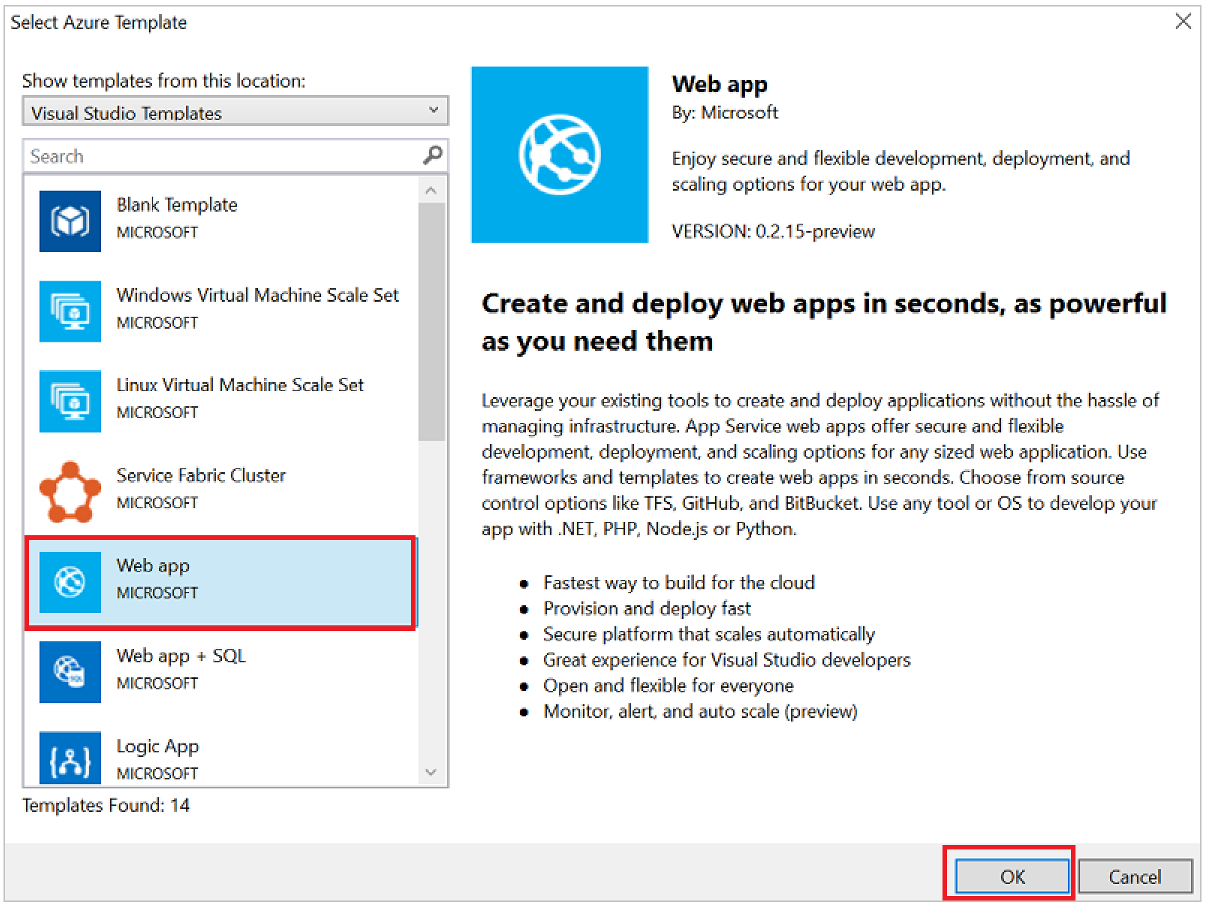
Solution: Create new solution

Solution name: ExampleAppDeploy

☐ Place solution and project in the same directory

Framework: .NET Framework 4.7.2

3. Choose the template that you want to deploy to Azure Resource Manager. Notice there are many different options based on the type of project you wish to deploy. For this article, choose the **Web app** template and **OK**.



Select Azure Template

Show templates from this location: Visual Studio Templates

Blank Template
MICROSOFT

Windows Virtual Machine Scale Set
MICROSOFT

Linux Virtual Machine Scale Set
MICROSOFT

Service Fabric Cluster
MICROSOFT

Web app
MICROSOFT

Web app + SQL
MICROSOFT

Logic App
MICROSOFT

Templates Found: 14

Web app
By: Microsoft

Enjoy secure and flexible development, deployment, and scaling options for your web app.

VERSION: 0.2.15-preview

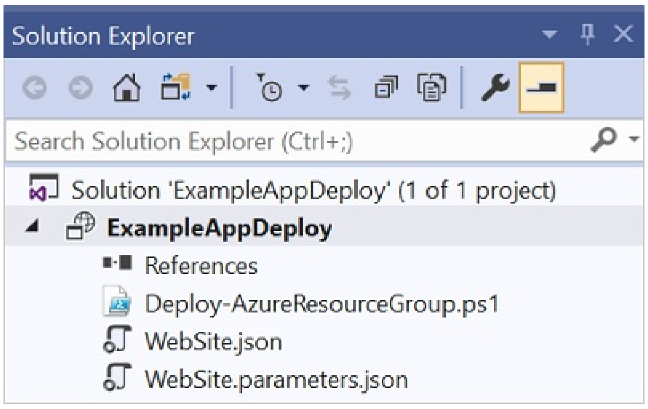
Create and deploy web apps in seconds, as powerful as you need them

Leverage your existing tools to create and deploy applications without the hassle of managing infrastructure. App Service web apps offer secure and flexible development, deployment, and scaling options for any sized web application. Use frameworks and templates to create web apps in seconds. Choose from source control options like TFS, GitHub, and BitBucket. Use any tool or OS to develop your app with .NET, PHP, Node.js or Python.

- Fastest way to build for the cloud
- Provision and deploy fast
- Secure platform that scales automatically
- Great experience for Visual Studio developers
- Open and flexible for everyone
- Monitor, alert, and auto scale (preview)

The template you pick is just a starting point; you can add and remove resources to fulfill your scenario.

4. Visual Studio creates a resource group deployment project for the web app. To see the files for your project, look at the node in the deployment project.


Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'ExampleAppDeploy' (1 of 1 project)
ExampleAppDeploy
References
Deploy-AzureResourceGroup.ps1
WebSite.json
WebSite.parameters.json

Since you chose the Web app template, you see the following files:

FILE NAME	DESCRIPTION
Deploy-AzureResourceGroup.ps1	A PowerShell script that runs PowerShell commands to deploy to Azure Resource Manager. Visual Studio uses this PowerShell script to deploy your template.
WebSite.json	The Resource Manager template that defines the infrastructure you want deploy to Azure, and the parameters you can provide during deployment. It also defines the dependencies between the resources so Resource Manager deploys the resources in the correct order.
WebSite.parameters.json	A parameters file that has values needed by the template. You pass in parameter values to customize each deployment.

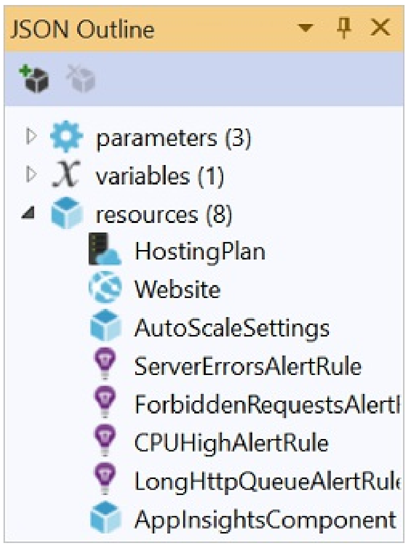
All resource group deployment projects have these basic files. Other projects may have additional files to support other functionality.

Customize Resource Manager template

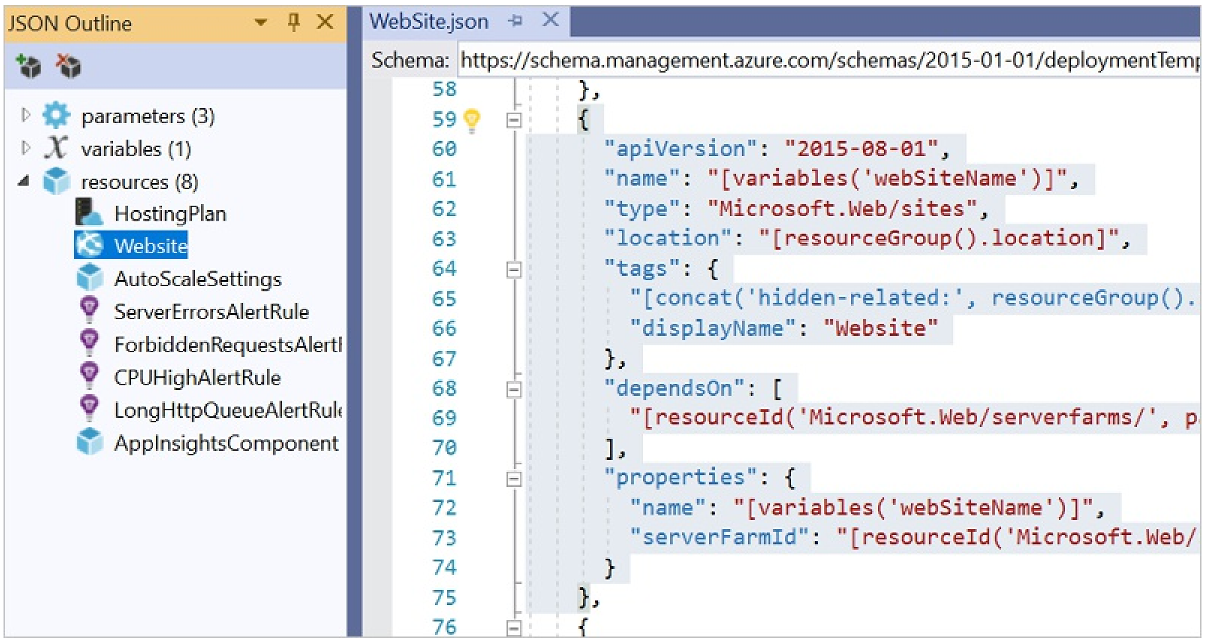
You can customize a deployment project by modifying the Resource Manager template that describes the resources you want to deploy. To learn about the elements of the Resource Manager template, see Authoring Azure Resource Manager templates.

1. To work on your template, open **WebSite.json**.

2. The Visual Studio editor provides tools to assist you with editing the Resource Manager template. The **JSON Outline** window makes it easy to see the elements defined in your template.



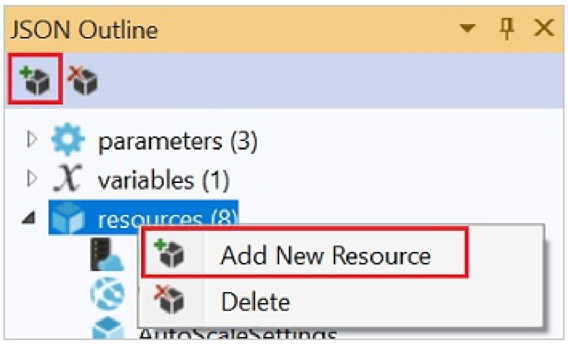
3. Select an element in the outline to go to that part of the template.



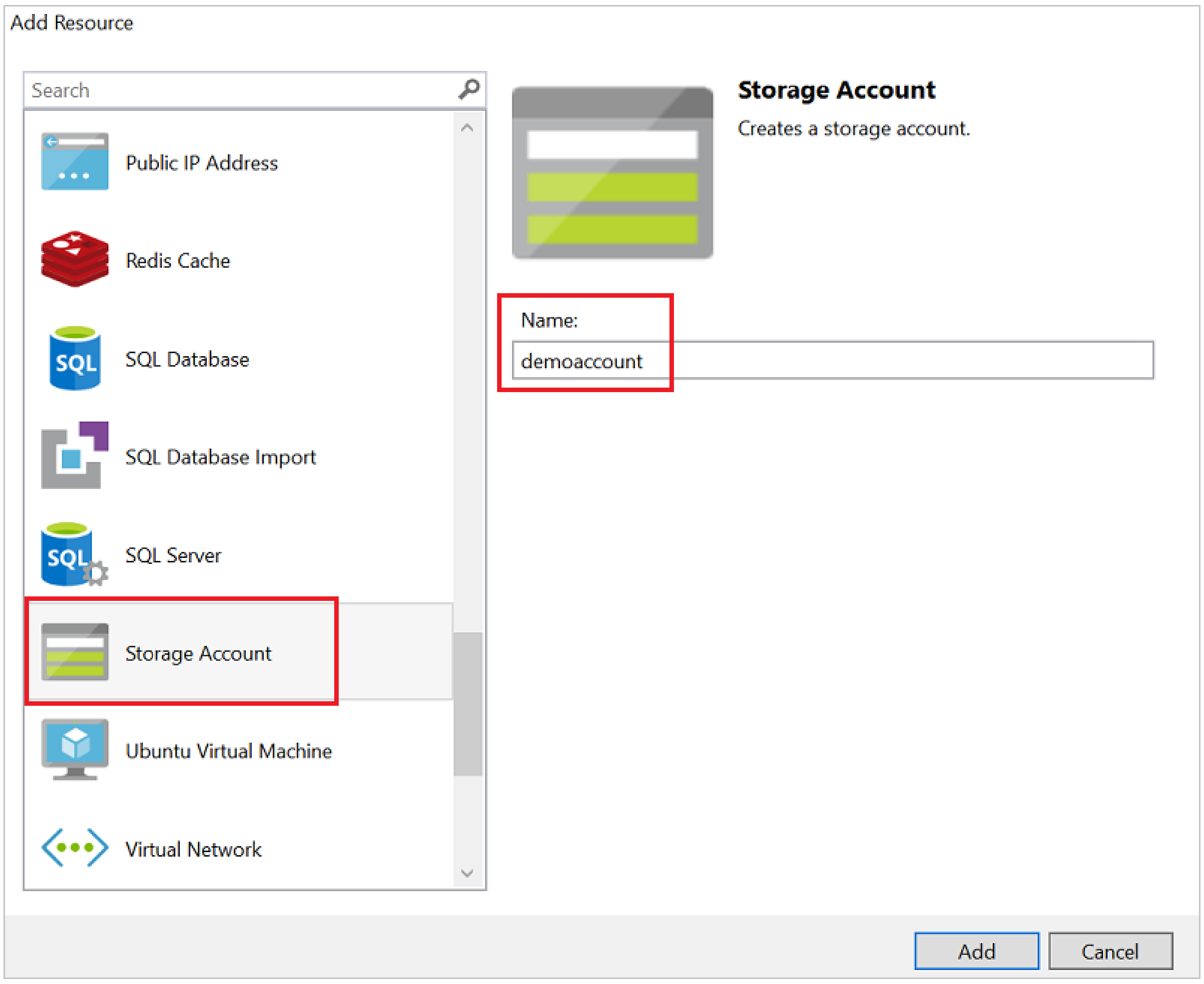
Schema: https://schema.management.azure.com/schemas/2015-01-01/deploymentTemp

```
},
{
  "apiVersion": "2015-08-01",
  "name": "[variables('webSiteName')]",
  "type": "Microsoft.Web/sites",
  "location": "[resourceGroup().location]",
  "tags": {
    "[concat('hidden-related:', resourceGroup().
    "displayName": "Website"
  },
  "dependsOn": [
    "[resourceId('Microsoft.Web/serverfarms/', p
  ],
  "properties": {
    "name": "[variables('webSiteName')]",
    "serverFarmId": "[resourceId('Microsoft.Web/
  }
},
{
```

4. You can add a resource by either selecting the **Add Resource** button at the top of the JSON Outline window, or by right-clicking **resources** and selecting **Add New Resource**.



5. Select **Storage Account** and give it a name. Provide a name that is no more than 11 characters, and only contains numbers and lower-case letters.


Add Resource

Search

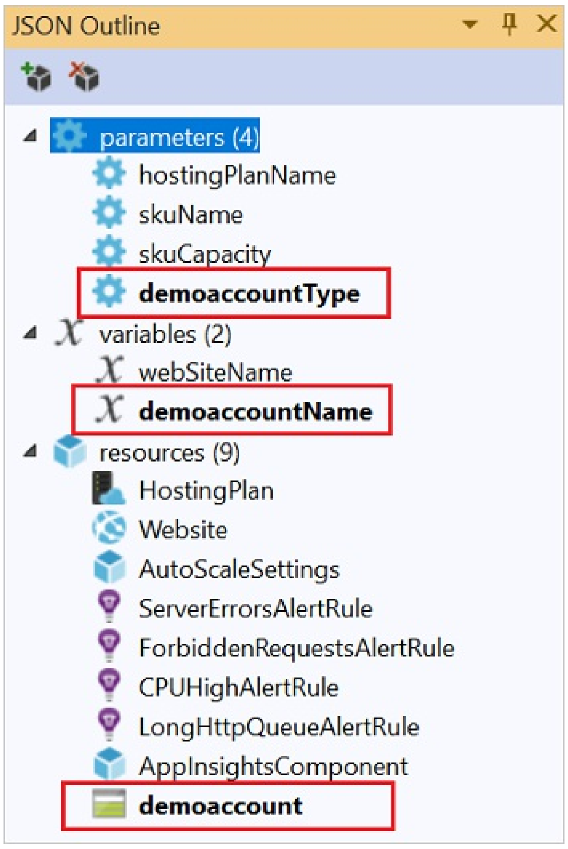
- Public IP Address
- Redis Cache
- SQL Database
- SQL Database Import
- SQL Server
- Storage Account
- Ubuntu Virtual Machine
- Virtual Network

Storage Account

Creates a storage account.

Name: demoaccount

6. Notice that not only was the resource added, but also a parameter for the type storage account, and a variable for the name of the storage account.


JSON Outline

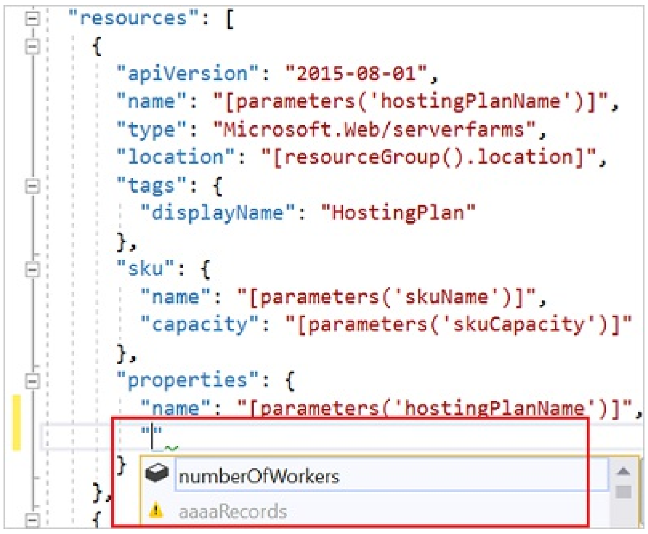
- parameters (4)
  - hostingPlanName
  - skuName
  - skuCapacity
  - **demoaccountType**
- variables (2)
  - webSiteName
  - **demoaccountName**
- resources (9)
  - HostingPlan
  - Website
  - AutoScaleSettings
  - ServerErrorsAlertRule
  - ForbiddenRequestsAlertRule
  - CPUHighAlertRule
  - LongHttpQueueAlertRule
  - AppInsightsComponent
  - **demoaccount**

7. The parameter for the type of storage account is pre-defined with allowed types and a default type. You can leave these values or edit them for your scenario. If you don't want anyone to deploy a **Premium_LRS**

storage account through this template, remove it from the allowed types.

```
"demoaccountType": {
  "type": "string",
  "defaultValue": "Standard_LRS",
  "allowedValues": [
    "Standard_LRS",
    "Standard_ZRS",
    "Standard_GRS",
    "Standard_RAGRS"
  ]
}
```

8. Visual Studio also provides intellisense to help you understand the properties that are available when editing the template. For example, to edit the properties for your App Service plan, navigate to the **HostingPlan** resource, and add a value for the **properties**. Notice that intellisense shows the available values and provides a description of that value.



```
"resources": [
  {
    "apiVersion": "2015-08-01",
    "name": "[parameters('hostingPlanName')]",
    "type": "Microsoft.Web/serverfarms",
    "location": "[resourceGroup().location]",
    "tags": {
      "displayName": "HostingPlan"
    },
    "sku": {
      "name": "[parameters('skuName')]",
      "capacity": "[parameters('skuCapacity')]"
    },
    "properties": {
      "name": "[parameters('hostingPlanName')]",
      ""
    }
  },
  {
```

You can set **numberOfWorkers** to 1, and save the file.

```
"properties": {
  "name": "[parameters('hostingPlanName')]",
  "numberOfWorkers": 1
}
```

9. Open the **WebSite.parameters.json** file. You use the parameters file to pass in values during deployment that customize the resource being deployed. Give the hosting plan a name, and save the file.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "hostingPlanName": {
      "value": "demoHostPlan"
    }
  }
}
```

Deploy project to Azure

You're now ready to deploy your project to a resource group.

By default, the PowerShell script (Deploy-AzureResourceGroup.ps1) in the project uses the AzureRM module. If you still have the AzureRM module installed and want to continue using it, you can use this default script. With this script, you can use the Visual Studio interface to deploy your solution.

However, if you've migrated to the new Az module, you need to add a new script to your project. To add a script that uses the Az module, copy the Deploy-AzTemplate.ps1 script and add it to your project. To use this script for deployment, you must run it from a PowerShell console, rather than using Visual Studio's deployment interface.

Both approaches are shown in this article. This article refers to the default script as the AzureRM module script, and the new script as the Az module script.

Az module script

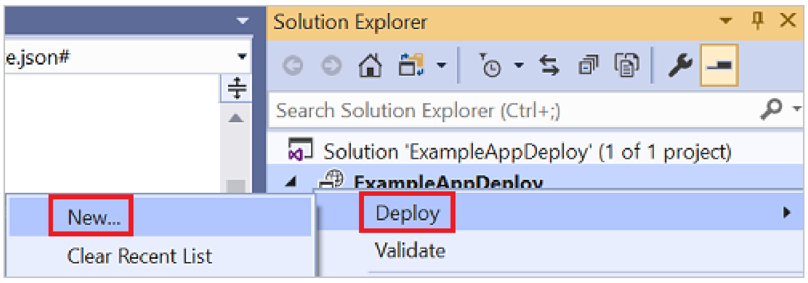
For the Az module script, open a PowerShell console and run:

```
.\Deploy-AzTemplate.ps1 -ArtifactStagingDirectory . -Location centralus -TemplateFile WebSite.json -
TemplateParametersFile WebSite.parameters.json
```

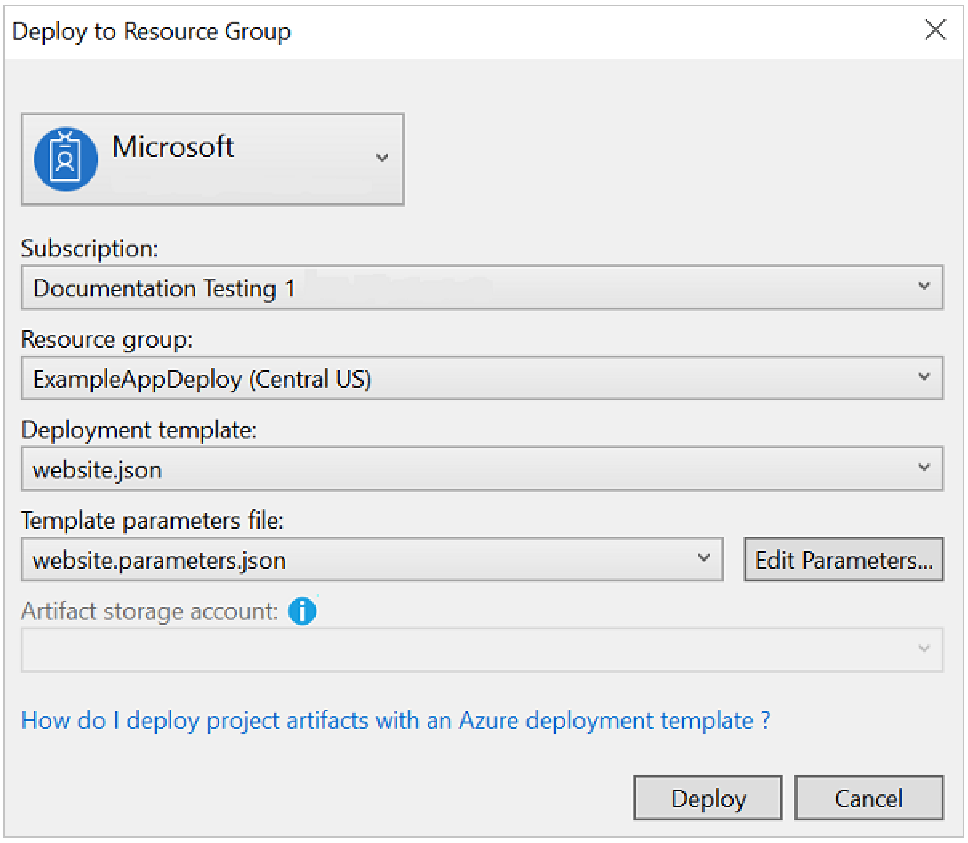
AzureRM module script

For the AzureRM module script, use Visual Studio:

1. On the shortcut menu of the deployment project node, choose **Deploy** > **New**.


e.json#
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'ExampleAppDeploy' (1 of 1 project)
ExampleAppDeploy
New...
Clear Recent List
Deploy
Validate

2. The **Deploy to Resource Group** dialog box appears. In the **Resource group** dropdown box, choose an existing resource group or create a new one. Select **Deploy**.


Deploy to Resource Group

Microsoft

Subscription: Documentation Testing 1

Resource group: ExampleAppDeploy (Central US)

Deployment template: website.json

Template parameters file: website.parameters.json

Artifact storage account:

How do I deploy project artifacts with an Azure deployment template ?

3. In the **Output** windows, you see the status of the deployment. When the deployment has finished, the last message indicates a successful deployment with something similar to:

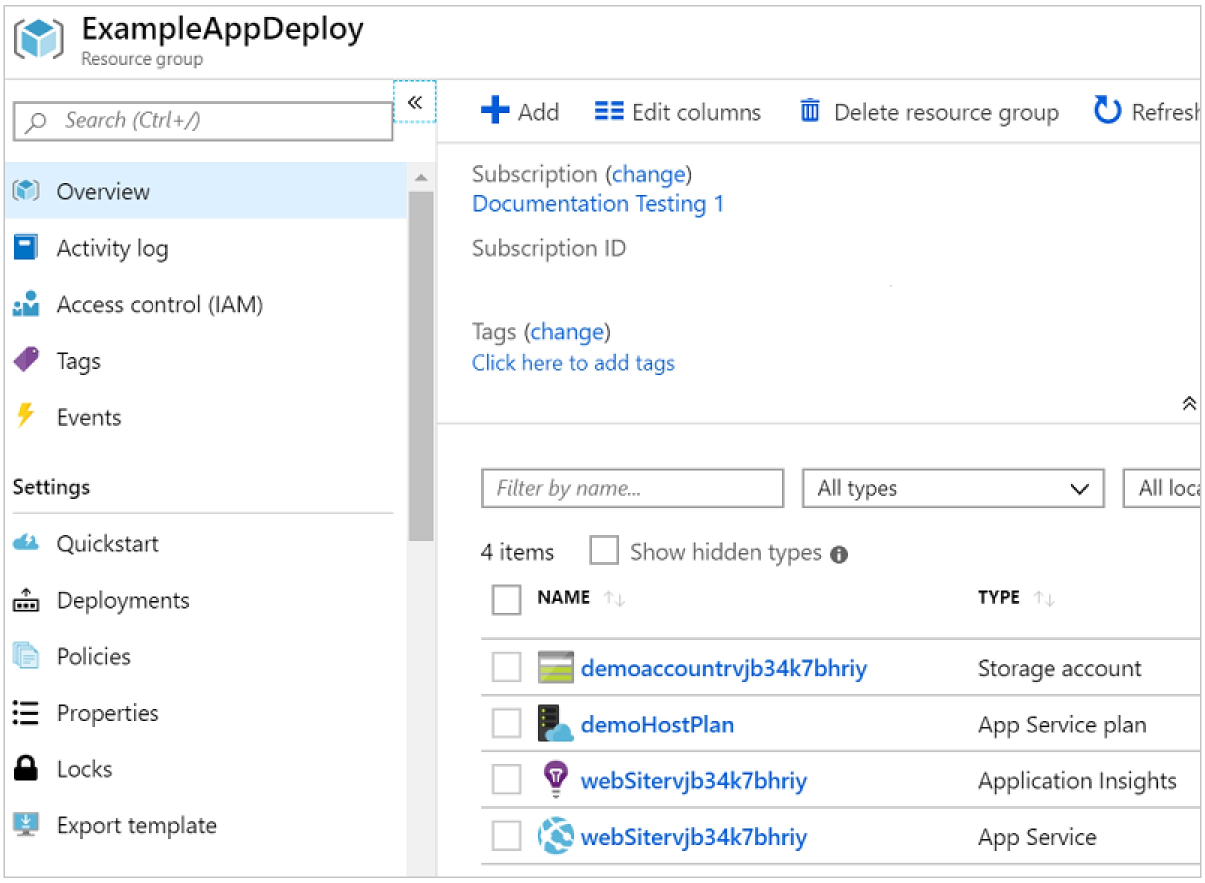
```
18:00:58 - Successfully deployed template 'website.json' to resource group 'ExampleAppDeploy'.
```

View deployed resources

Let's check the results.

1. In a browser, open the Azure portal and sign in to your account. To see the resource group, select **Resource groups** and the resource group you deployed to.

2. You see all the deployed resources. Notice that the name of the storage account isn't exactly what you specified when adding that resource. The storage account must be unique. The template automatically adds a string of characters to the name you provided to create a unique name.



ExampleAppDeploy
Resource group

Subscription (change)
Documentation Testing 1

Subscription ID

Tags (change)
Click here to add tags

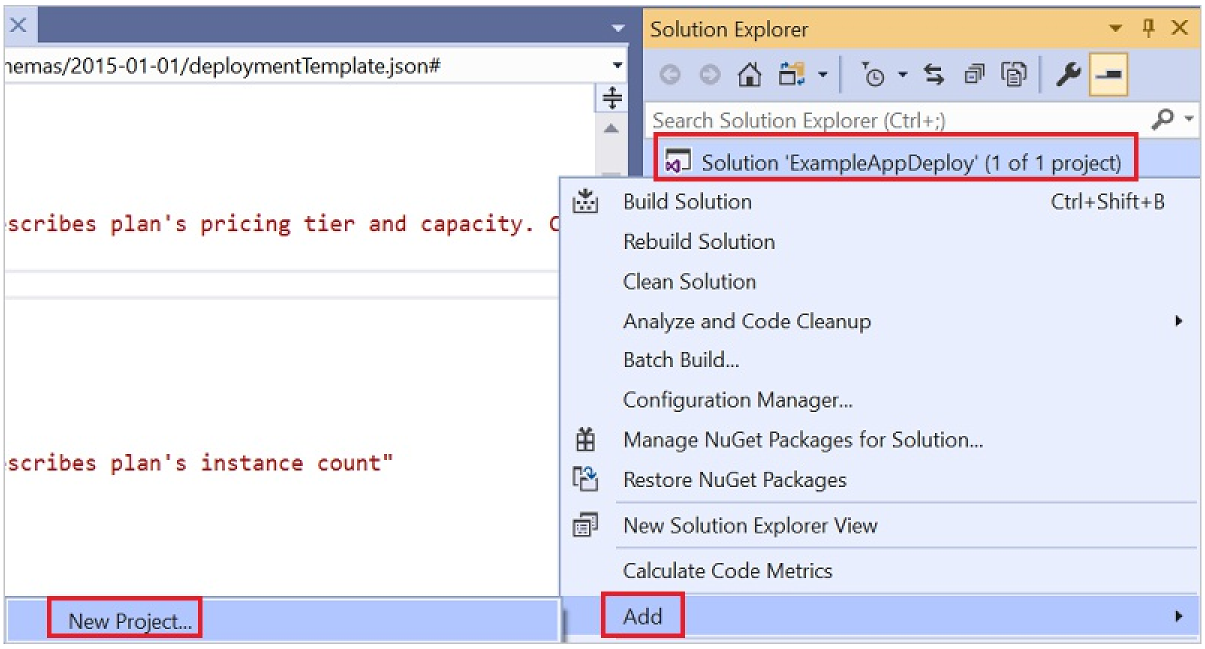
4 items

NAME	TYPE
demoaccountrvjb34k7bhriy	Storage account
demoHostPlan	App Service plan
webSitervjb34k7bhriy	Application Insights
webSitervjb34k7bhriy	App Service

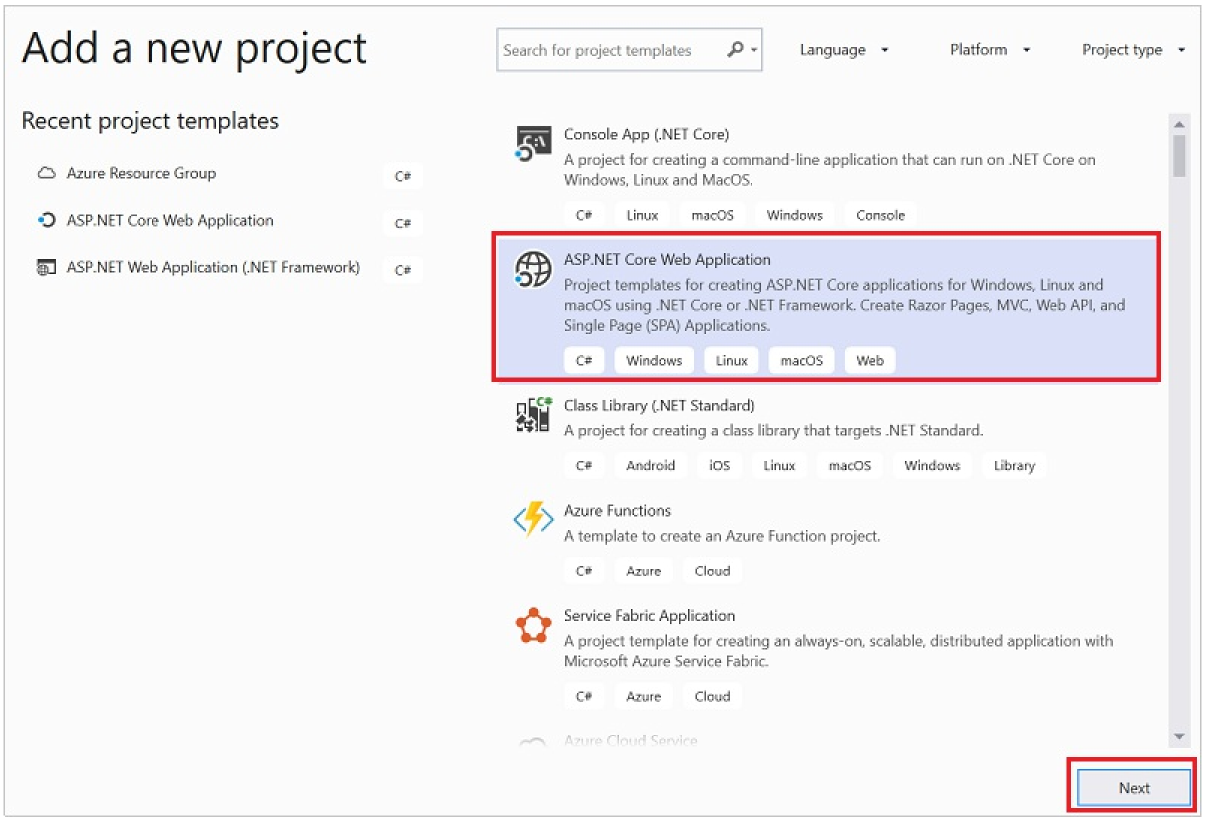
Add code to project

At this point, you've deployed the infrastructure for your app, but there's no actual code deployed with the project.

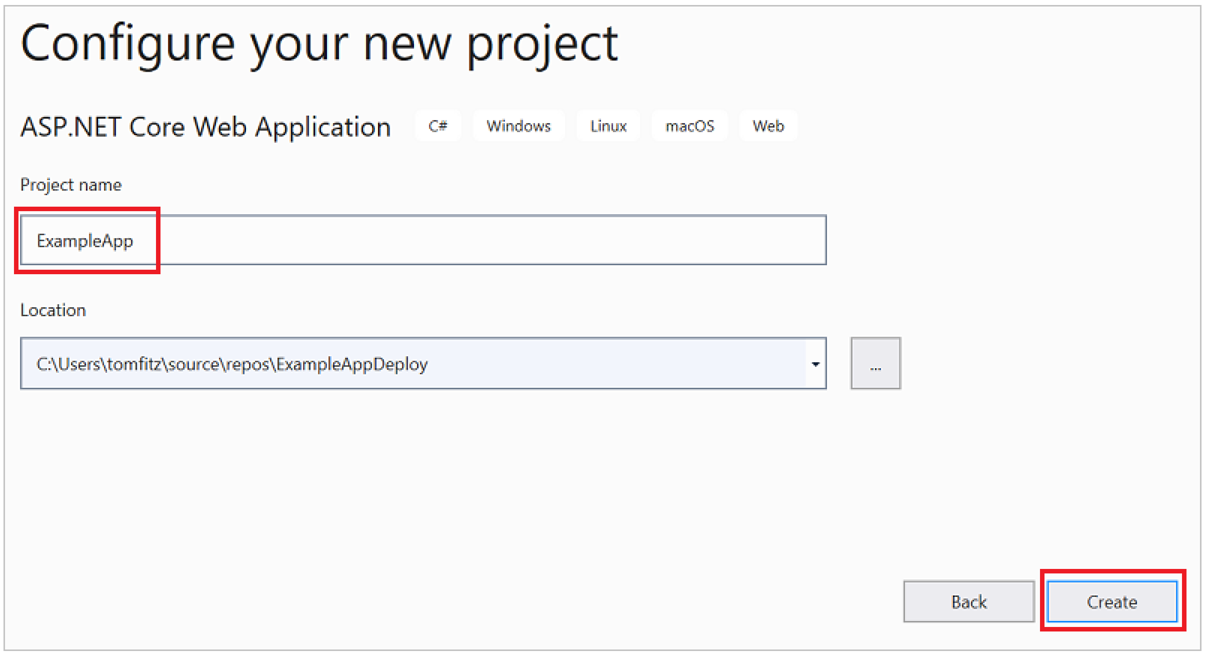
1. Add a project to your Visual Studio solution. Right-click the solution, and select **Add** > **New Project**.


hemas/2015-01-01/deploymentTemplate.json#
scribes plan's pricing tier and capacity. C
scribes plan's instance count"
New Project...
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'ExampleAppDeploy' (1 of 1 project)
Build Solution Ctrl+Shift+B
Rebuild Solution
Clean Solution
Analyze and Code Cleanup
Batch Build...
Configuration Manager...
Manage NuGet Packages for Solution...
Restore NuGet Packages
New Solution Explorer View
Calculate Code Metrics
Add

2. Add an **ASP.NET Core Web Application**.


Add a new project
Search for project templates
Language
Platform
Project type
Recent project templates
Azure Resource Group C#
ASP.NET Core Web Application C#
ASP.NET Web Application (.NET Framework) C#
Console App (.NET Core)
A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.
C# Linux macOS Windows Console
ASP.NET Core Web Application
Project templates for creating ASP.NET Core applications for Windows, Linux and macOS using .NET Core or .NET Framework. Create Razor Pages, MVC, Web API, and Single Page (SPA) Applications.
C# Windows Linux macOS Web
Class Library (.NET Standard)
A project for creating a class library that targets .NET Standard.
C# Android iOS Linux macOS Windows Library
Azure Functions
A template to create an Azure Function project.
C# Azure Cloud
Service Fabric Application
A project template for creating an always-on, scalable, distributed application with Microsoft Azure Service Fabric.
C# Azure Cloud
Azure Cloud Service
Next

3. Give your web app a name, and select **Create**.

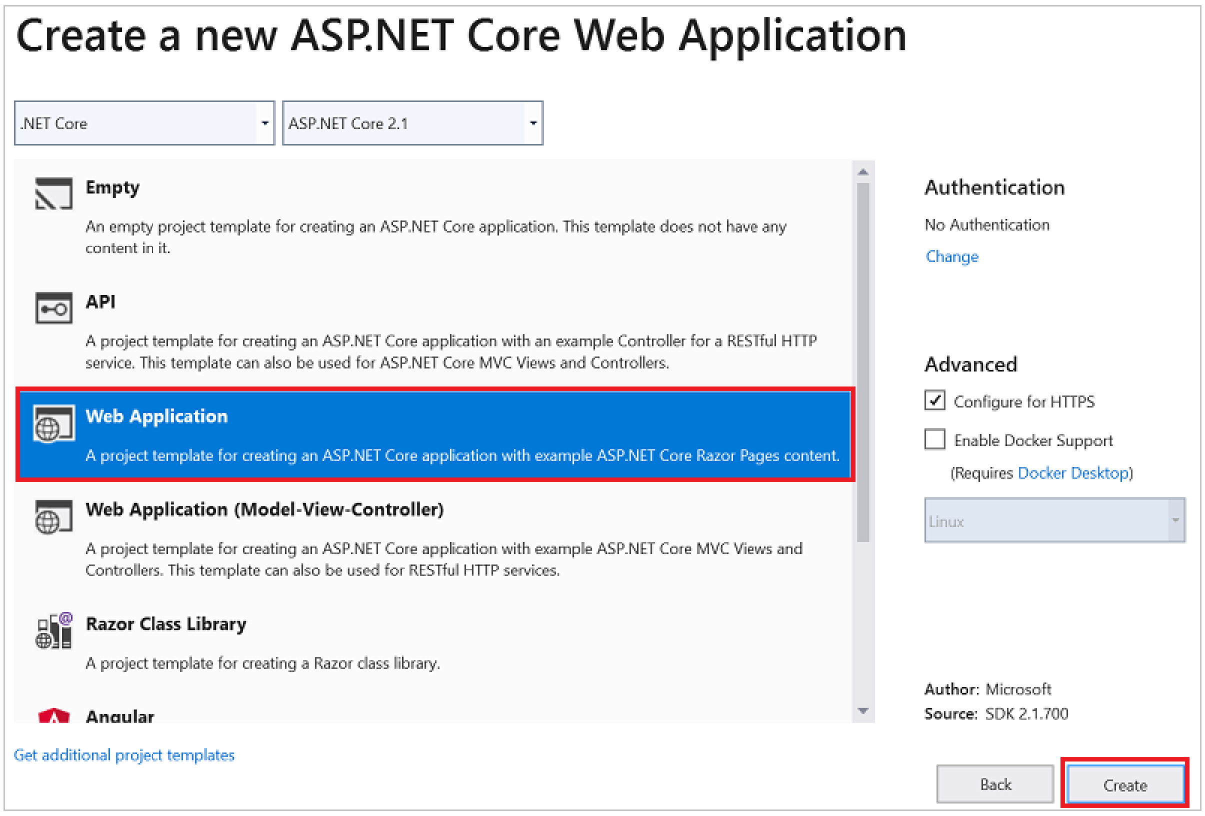

Configure your new project

ASP.NET Core Web Application

Project name: ExampleApp

Location: C:\Users\tomfitz\source\repos\ExampleAppDeploy

4. Select **Web Application** and **Create**.


Create a new ASP.NET Core Web Application

.NET Core
ASP.NET Core 2.1

Empty
An empty project template for creating an ASP.NET Core application. This template does not have any content in it.

API
A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.

Web Application
A project template for creating an ASP.NET Core application with example ASP.NET Core Razor Pages content.

Web Application (Model-View-Controller)
A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

Razor Class Library
A project template for creating a Razor class library.

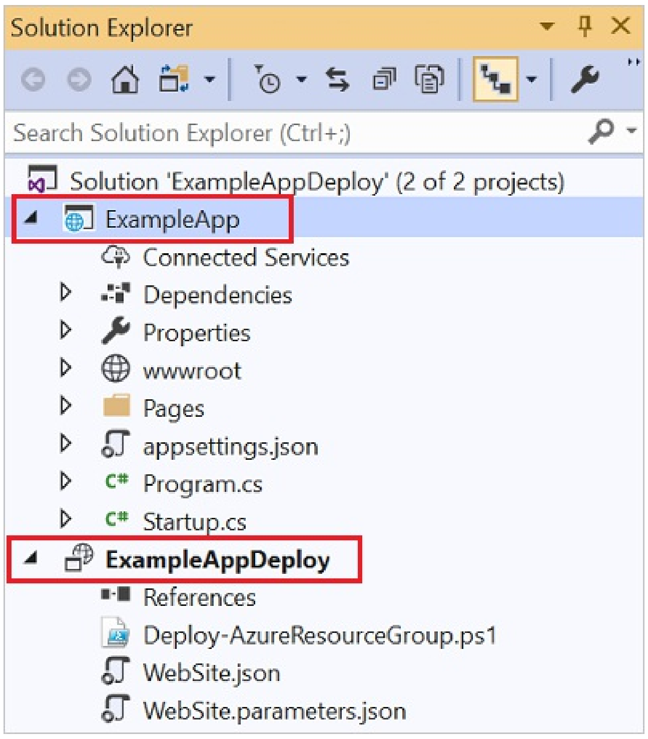
Angular

Authentication
No Authentication
Change

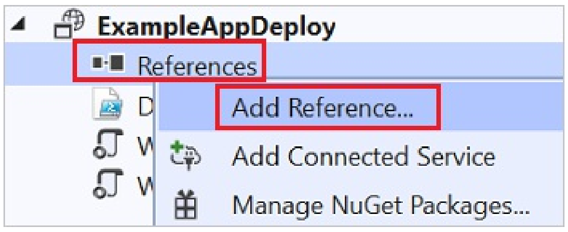
Advanced
[✓] Configure for HTTPS
[ ] Enable Docker Support
(Requires Docker Desktop)
Linux

Author: Microsoft
Source: SDK 2.1.700

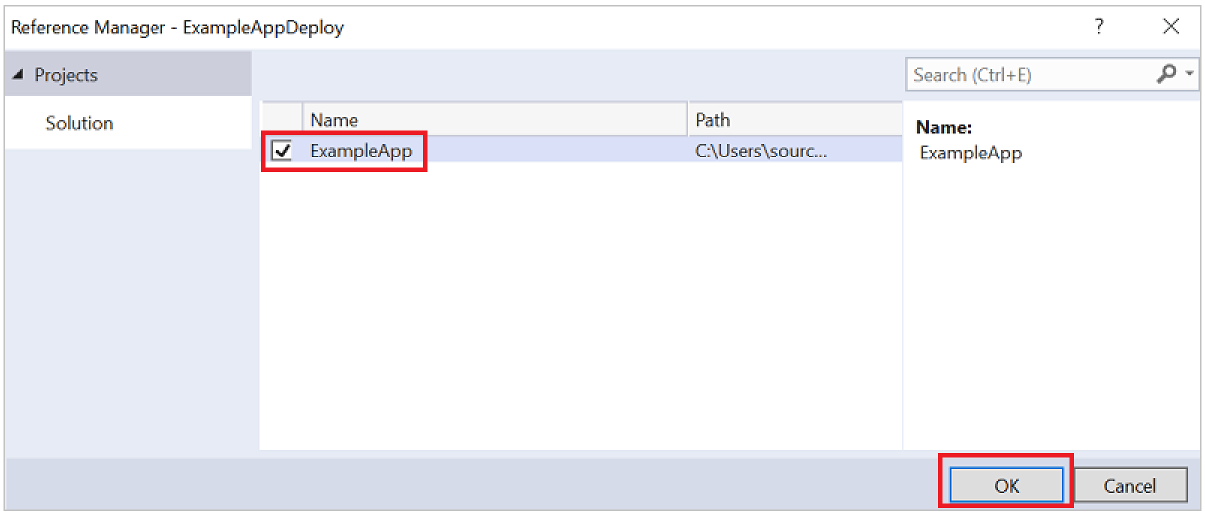
5. After Visual Studio creates your web app, you see both projects in the solution.



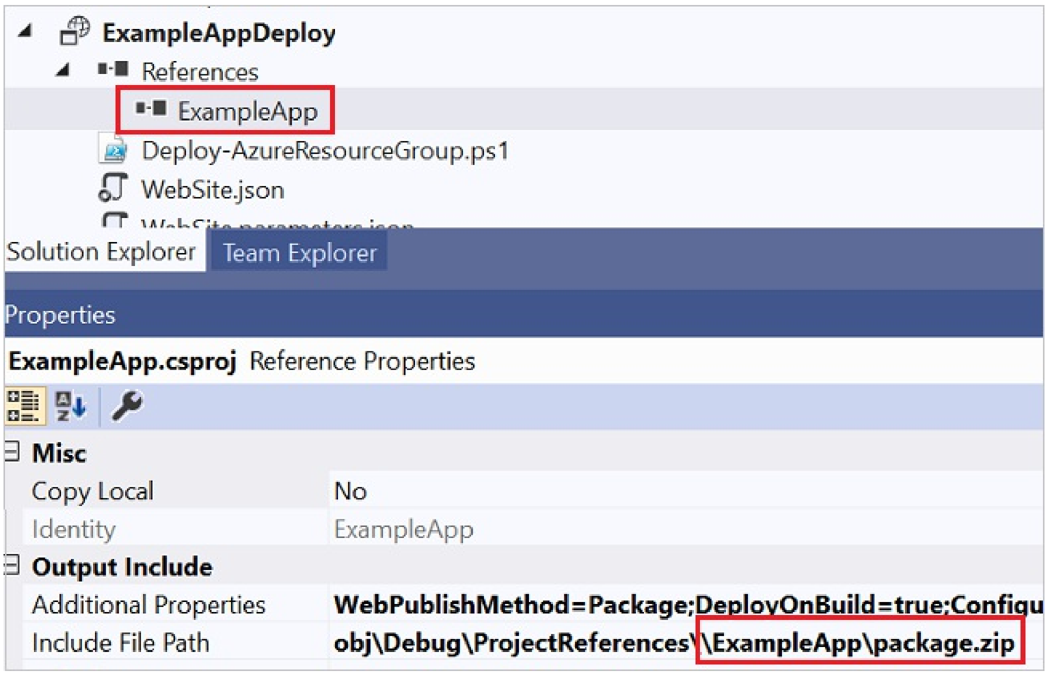
6. Now, you need to make sure your resource group project is aware of the new project. Go back to your resource group project (ExampleAppDeploy). Right-click **References** and select **Add Reference**.



7. Select the web app project that you created.



By adding a reference, you link the web app project to the resource group project, and automatically sets some properties. You see these properties in the **Properties** window for the reference. The **Include File Path** has the path where the package is created. Note the folder (ExampleApp) and file (package.zip). You need to know these values because you provide them as parameters when deploying the app.


ExampleAppDeploy
References
ExampleApp
Deploy-AzureResourceGroup.ps1
WebSite.json

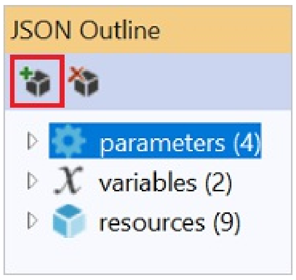
Solution Explorer | Team Explorer

Properties

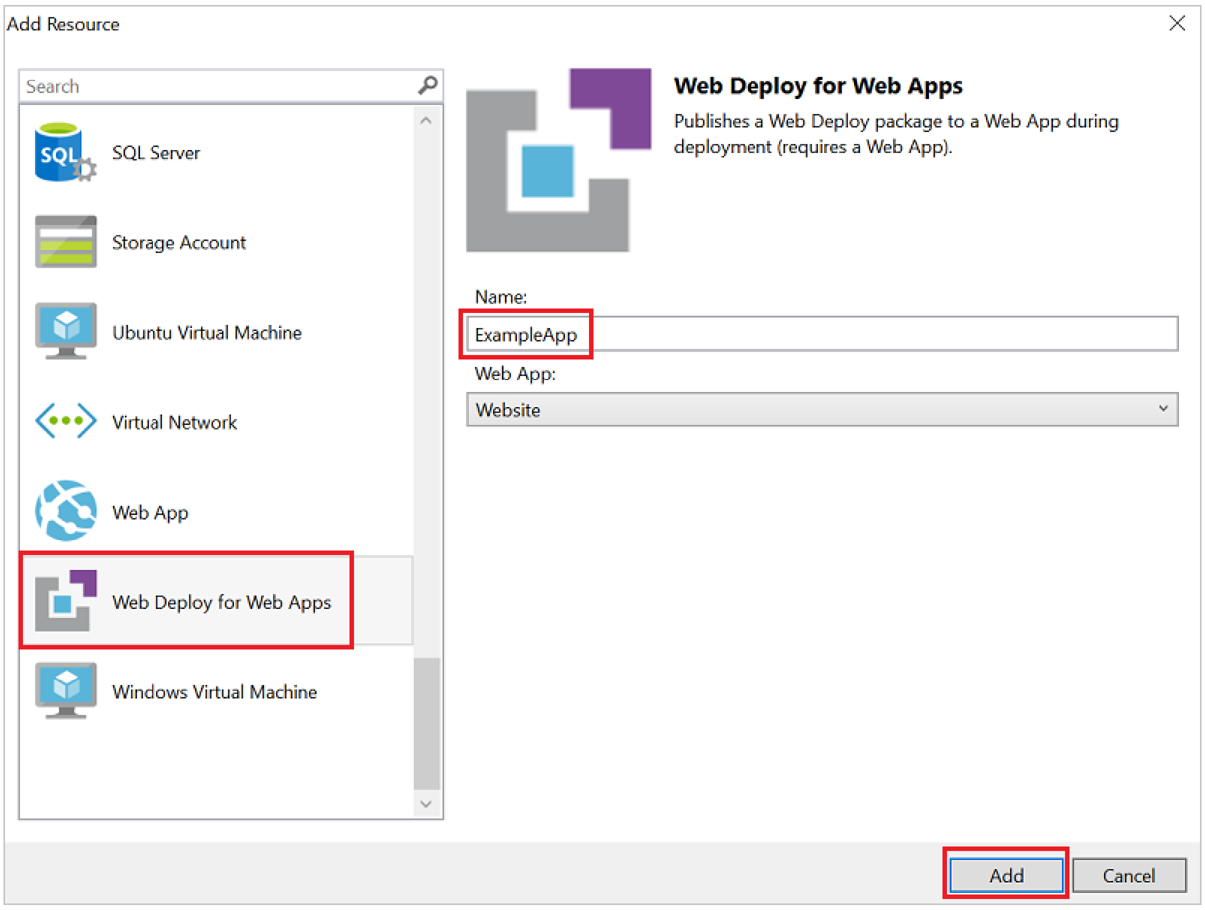
ExampleApp.csproj Reference Properties

Misc	
Copy Local	No
Identity	ExampleApp
Output Include	
Additional Properties	WebPublishMethod=Package;DeployOnBuild=true;Configu
Include File Path	obj\Debug\ProjectReferences\ExampleApp\package.zip

8. Go back to your template (WebSite.json) and add a resource to the template.



9. This time select **Web Deploy for Web Apps**.



Web Deploy for Web Apps

Publishes a Web Deploy package to a Web App during deployment (requires a Web App).

Name: ExampleApp

Web App: Website

Save your template.

10. There are some new parameters in your template. They were added in the previous step. You don't need to provide values for **_artifactsLocation** or **_artifactsLocationSasToken** because those values are automatically generated. However, you have to set the folder and file name to the path that contains the deployment package. The names of these parameters end with **PackageFolder** and **PackageFileName**. The first part of the name is the name of the Web Deploy resource you added. In this article, they're named **ExampleAppPackageFolder** and **ExampleAppPackageFileName**.

    Open **Website.parameters.json** and set those parameters to the values you saw in the reference properties. Set **ExampleAppPackageFolder** to the name of the folder. Set **ExampleAppPackageFileName** to the name of the zip file.

    ```
    {
      "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
      "contentVersion": "1.0.0.0",
      "parameters": {
        "hostingPlanName": {
          "value": "demoHostPlan"
        },
        "ExampleAppPackageFolder": {
          "value": "ExampleApp"
        },
        "ExampleAppPackageFileName": {
          "value": "package.zip"
        }
      }
    }
    ```

Deploy code with infrastructure

Because you added code to the project, your deployment is a little different this time. During deployment, you stage artifacts for your project to a place that Resource Manager can access. The artifacts are staged to a storage account.

Az module script

There's one small change you need to make to your template if you're using the Az module script. This script adds a slash to the artifacts location but your template doesn't expect that slash. Open WebSite.json and find the properties for the MSDeploy extension. It has a property named **packageUri**. Remove the slash between the artifacts location and the package folder.

It should look like:

```
"packageUri": "[concat(parameters('_artifactsLocation'), parameters('ExampleAppPackageFolder'), '/',
parameters('ExampleAppPackageFileName'), parameters('_artifactsLocationSasToken'))]",
```

Notice in the preceding example there is no `'/',` between **parameters('_artifactsLocation')** and **parameters('ExampleAppPackageFolder')**.

Rebuild the project. Building the project makes sure the files you need to deploy are added to the staging folder.

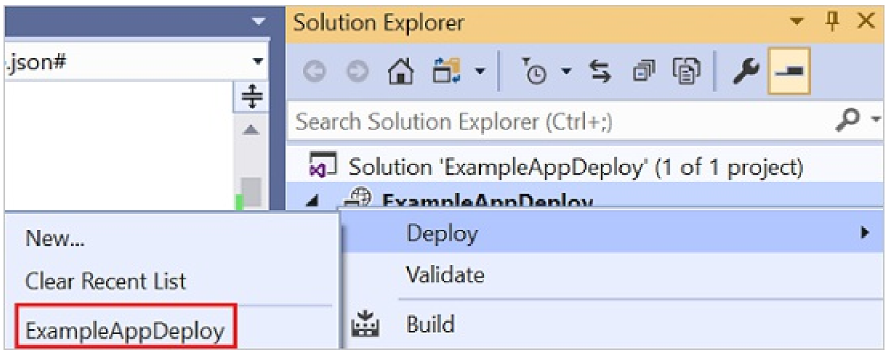
Now, open a PowerShell console and run:

```
.\Deploy-AzTemplate.ps1 -ArtifactStagingDirectory .\bin\Debug\staging\ExampleAppDeploy -Location centralus -
TemplateFile WebSite.json -TemplateParametersFile WebSite.parameters.json -UploadArtifacts -StorageAccountName
<storage-account-name>
```

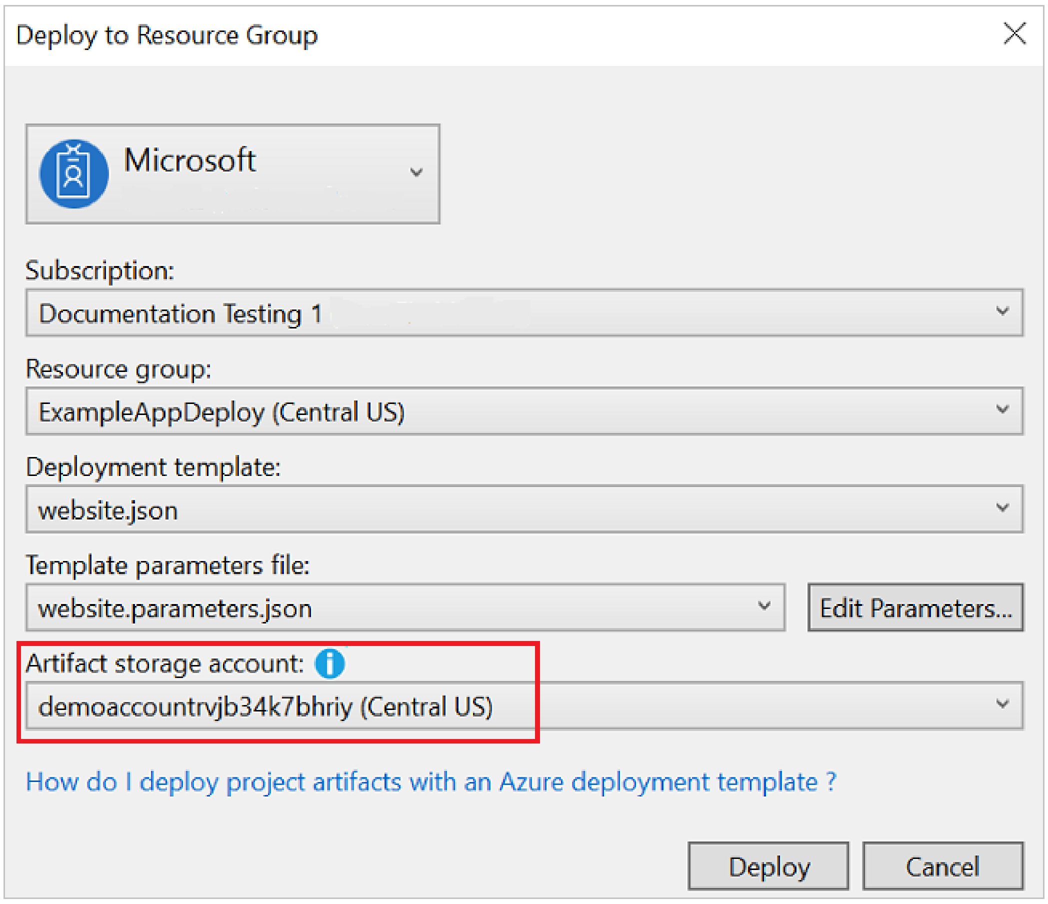
AzureRM module script

For the AzureRM module script, use Visual Studio:

1. To redeploy, choose **Deploy**, and the resource group you deployed earlier.



2. Select the storage account you deployed with this resource group for the **Artifact storage account**.


Deploy to Resource Group
Microsoft
Edit Parameters...
How do I deploy project artifacts with an Azure deployment template ?
Deploy
Cancel

Subscription: Documentation Testing 1

Resource group: ExampleAppDeploy (Central US)

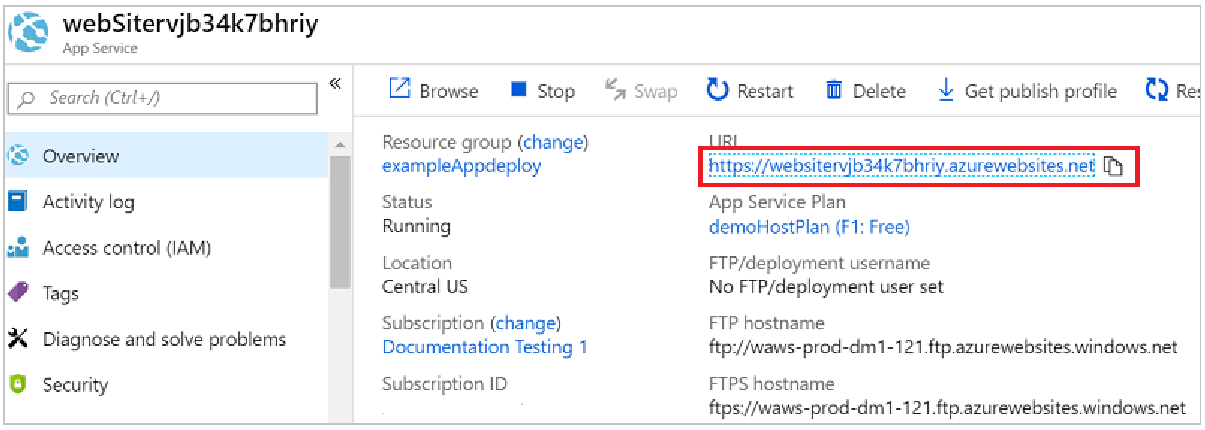
Deployment template: website.json

Template parameters file: website.parameters.json

Artifact storage account: demoaccountrvjb34k7bhriy (Central US)

View web app

1. After the deployment has finished, select your web app in the portal. Select the URL to browse to the site.



Resource group (change)
exampleAppdeploy

Status
Running

Location
Central US

Subscription (change)
Documentation Testing 1

Subscription ID

URL
https://websitervjb34k7bhriy.azurewebsites.net

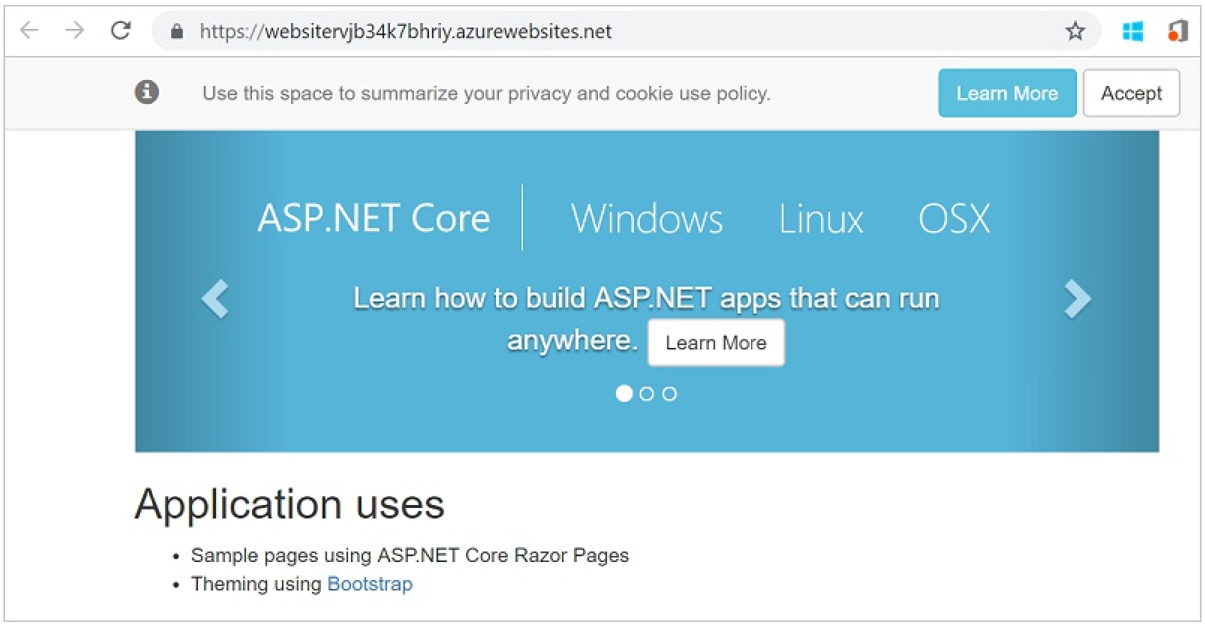
App Service Plan
demoHostPlan (F1: Free)

FTP/deployment username
No FTP/deployment user set

FTP hostname
ftp://waws-prod-dm1-121.ftp.azurewebsites.windows.net

FTPS hostname
ftps://waws-prod-dm1-121.ftp.azurewebsites.windows.net

2. Notice that you've successfully deployed the default ASP.NET app.


https://websitervjb34k7bhriy.azurewebsites.net
Use this space to summarize your privacy and cookie use policy.
Learn More
Accept
ASP.NET Core
Windows
Linux
OSX
Learn how to build ASP.NET apps that can run anywhere.
Learn More

Application uses

- Sample pages using ASP.NET Core Razor Pages
- Theming using Bootstrap

Add operations dashboard

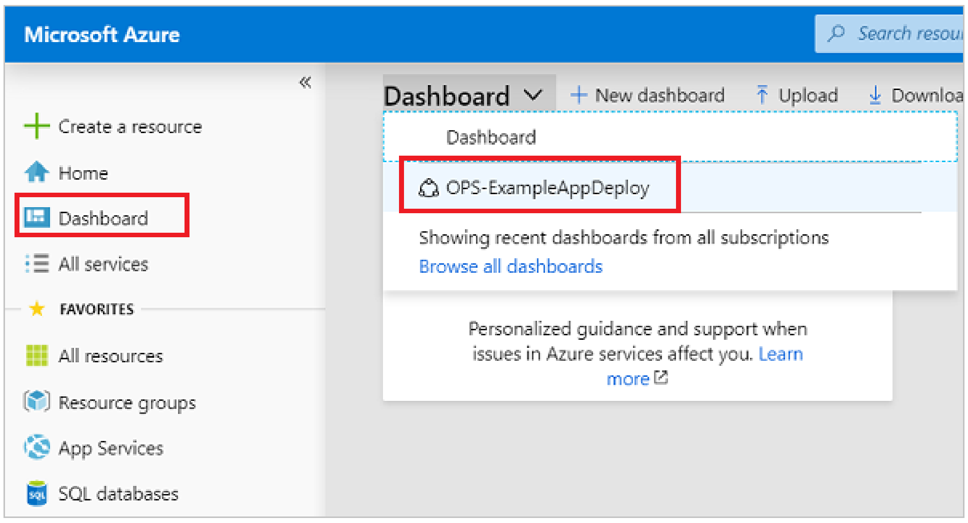
You aren't limited to only the resources that are available through the Visual Studio interface. You can customize your deployment by adding a custom resource to your template. To show adding a resource, you add an operational dashboard to manage the resource you deployed.

1. Open the WebSite.json file and add the following JSON after the storage account resource but before the closing `]` of the resources section.

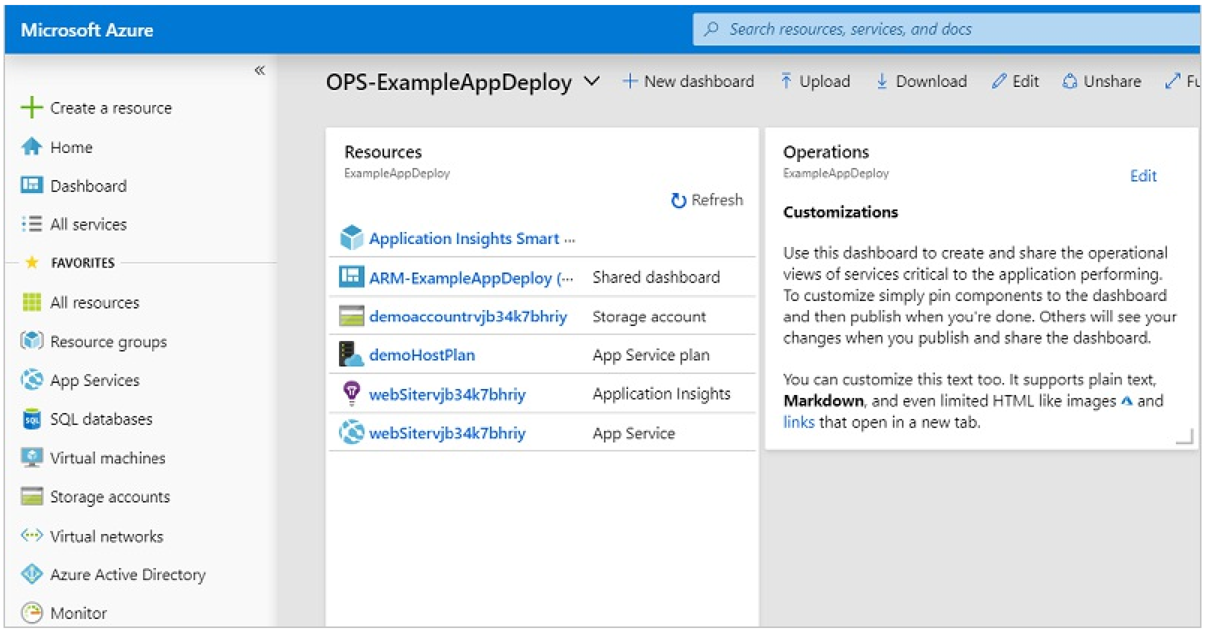
```
,{
  "properties": {
    "lenses": {
      "0": {
        "order": 0,
        "parts": {
          "0": {
            "position": {
              "x": 0,
              "y": 0,
              "colSpan": 4,
              "rowSpan": 6
            },
            "metadata": {
              "inputs": [
                {
                  "name": "resourceGroup",
                  "isOptional": true
                },
                {
                  "name": "id",
                  "value": "[resourceGroup().id]",
                  "isOptional": true
                }
              ],
              "type": "Extension/HubsExtension/PartType/ResourceGroupMapPinnedPart"
            }
          },
          "1": {
            "position": {
              "x": 4,
              "y": 0,
              "rowSpan": 3,
               colSpan : 4
             },
             "metadata": {
               "inputs": [],
               "type": "Extension[azure]/HubsExtension/PartType/MarkdownPart",
               "settings": {
                 "content": {
                   "settings": {
                     "content": "__Customizations__\n\nUse this dashboard to create and share the
operational views of services critical to the application performing. To customize simply pin components
to the dashboard and then publish when you're done. Others will see your changes when you publish and
share the dashboard.\n\nYou can customize this text too. It supports plain text, __Markdown__, and even
limited HTML like images <img width='10' src='https://portal.azure.com/favicon.ico'/> and <a
href='https://azure.microsoft.com' target='_blank'>links</a> that open in a new tab.\n",
                     "title": "Operations",
                     "subtitle": "[resourceGroup().name]"
                   }
                 }
               }
             }
           }
         }
       }
     },
     "metadata": {
       "model": {
         "timeRange": {
           "value": {
             "relative": {
               "duration": 24,
               "timeUnit": 1
             }
           },
           "type": "MsPortalFx.Composition.Configuration.ValueTypes.TimeRange"
         }
       }
     }
   },
   "apiVersion": "2015-08-01-preview",
   "name": "[concat('ARM-',resourceGroup().name)]",
   "type": "Microsoft.Portal/dashboards",
   "location": "[resourceGroup().location]",
   "tags": {
     "hidden-title": "[concat('OPS-',resourceGroup().name)]"
   }
 }
```

2. Redeploy your project.

3. After deployment has finished, view your dashboard in the portal. Select **Dashboard** and pick the one you deployed.



4. You see the customized dashboard.



Application Insights Smart ...	
ARM-ExampleAppDeploy (...	Shared dashboard
demoaccountrvjb34k7bhriy	Storage account
demoHostPlan	App Service plan
webSitervjb34k7bhriy	Application Insights
webSitervjb34k7bhriy	App Service

Use this dashboard to create and share the operational views of services critical to the application performing. To customize simply pin components to the dashboard and then publish when you're done. Others will see your changes when you publish and share the dashboard.

You can customize this text too. It supports plain text, **Markdown**, and even limited HTML like images and links that open in a new tab.

You can manage access to the dashboard by using RBAC groups. You can also customize the dashboard's appearance after it's deployed. However, if you redeploy the resource group, the dashboard is reset to its default state in your template. For more information about creating dashboards, see Programmatically create Azure Dashboards.

Clean up resources

When the Azure resources are no longer needed, clean up the resources you deployed by deleting the resource group.

1. From the Azure portal, select **Resource groups** from the left menu.
2. Select the resource group name.
3. Select **Delete resource group** from the top menu.

Next steps

In this quickstart, you learned how to create and deploy templates using Visual Studio. The next tutorial shows you how to find the information from template reference so you can create an encrypted Azure Storage account.

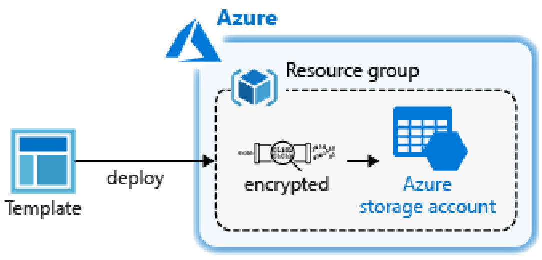
Create an encrypted storage account

Tutorial: Utilize the Azure Resource Manager template reference

3/12/2019 • 3 minutes to read • Edit Online

Learn how to find the template schema information, and use the information to create Azure Resource Manager templates.

In this tutorial, you use a base template from Azure Quickstart templates. Using template reference documentation, you customize the template to create an encrypted Storage account.



This tutorial covers the following tasks:

- Open a Quickstart template
- Understand the template
- Find the template reference
- Edit the template
- Deploy the template

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

To complete this article, you need:

- Visual Studio Code with Resource Manager Tools extension.

Open a Quickstart template

Azure QuickStart Templates is a repository for Resource Manager templates. Instead of creating a template from scratch, you can find a sample template and customize it. The template used in this quickstart is called Create a standard storage account. The template defines an Azure Storage account resource.

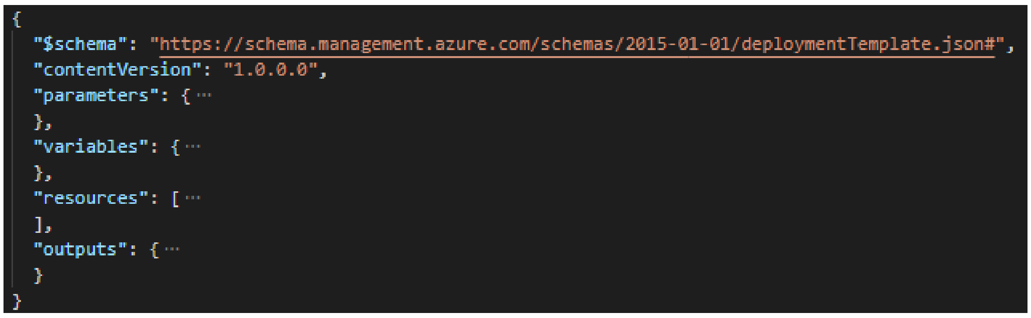
1. From Visual Studio Code, select **File**>**Open File**.
2. In **File name**, paste the following URL:

```
https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-storage-account-
create/azuredeploy.json
```

3. Select **Open** to open the file.
4. Select **File**>**Save As** to save the file as **azuredeploy.json** to your local computer.

Understand the schema

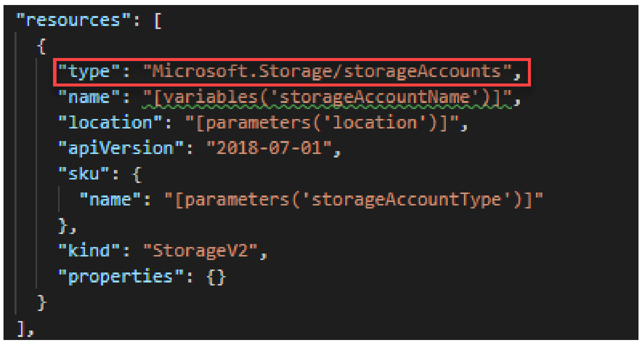
1. From VS Code, collapse the template to the root level. You have the simplest structure with the following elements:



```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {⋯
  },
  "variables": {⋯
  },
  "resources": [⋯
  ],
  "outputs": {⋯
  }
}
```

- **$schema**: specify the location of the JSON schema file that describes the version of the template language.
- **contentVersion**: specify any value for this element to document significant changes in your template.
- **parameters**: specify the values that are provided when deployment is executed to customize resource deployment.
- **variables**: specify the values that are used as JSON fragments in the template to simplify template language expressions.
- **resources**: specify the resource types that are deployed or updated in a resource group.
- **outputs**: specify the values that are returned after deployment.

2. Expand **resources**. There is a `Microsoft.Storage/storageAccounts` resource defined. The template creates a non-encrypted Storage account.



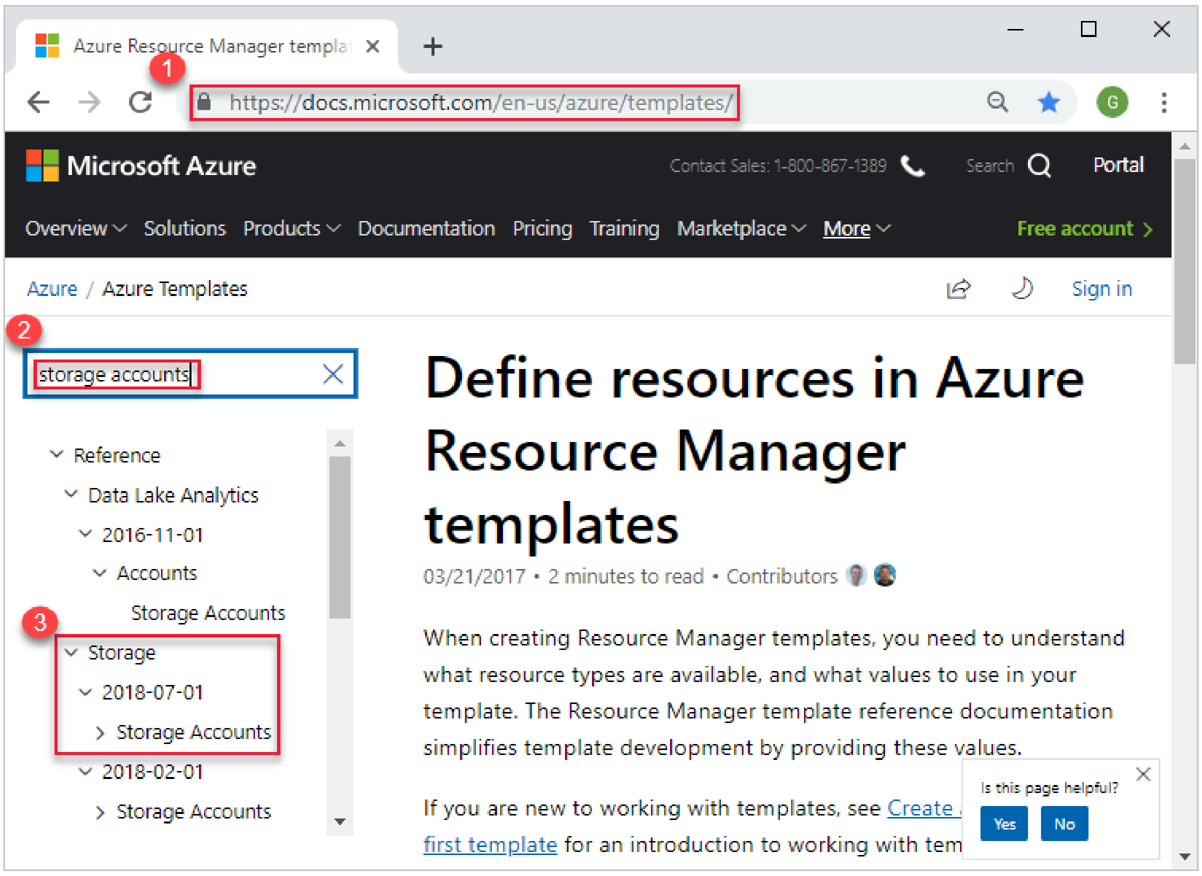
```
"resources": [
  {
    "type": "Microsoft.Storage/storageAccounts",
    "name": "[variables('storageAccountName')]",
    "location": "[parameters('location')]",
    "apiVersion": "2018-07-01",
    "sku": {
      "name": "[parameters('storageAccountType')]"
    },
    "kind": "StorageV2",
    "properties": {}
  }
],
```

Find the template reference

1. Browse to Azure Template reference.

2. In the **Filter by title** box, enter **storage accounts**.

3. Select **Reference/Template reference/Storage/<Version>/Storage Accounts** as shown in the following screenshot:


Azure Resource Manager templat
https://docs.microsoft.com/en-us/azure/templates/
Microsoft Azure
Contact Sales: 1-800-867-1389
Search
Portal
Overview Solutions Products Documentation Pricing Training Marketplace More
Free account
Azure / Azure Templates
Sign in
storage accounts

- Reference
  - Data Lake Analytics
    - 2016-11-01
      - Accounts
        - Storage Accounts
  - Storage
    - 2018-07-01
      - Storage Accounts
    - 2018-02-01
      - Storage Accounts

Define resources in Azure Resource Manager templates

03/21/2017 • 2 minutes to read • Contributors

When creating Resource Manager templates, you need to understand what resource types are available, and what values to use in your template. The Resource Manager template reference documentation simplifies template development by providing these values.

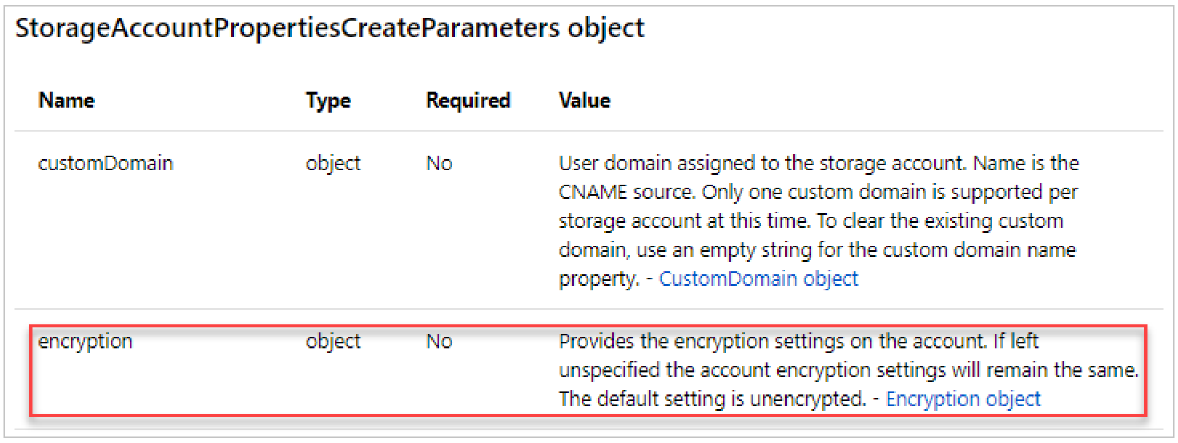
If you are new to working with templates, see Create first template for an introduction to working with tem

If you don't know which version to choose, use the latest version.

4. Find the encryption-related definition information.

```
"encryption": {
  "services": {
    "blob": {
      "enabled": boolean
    },
    "file": {
      "enabled": boolean
    }
  },
  "keySource": "string",
  "keyvaultproperties": {
    "keyname": "string",
    "keyversion": "string",
    "keyvaulturi": "string"
  }
},
```

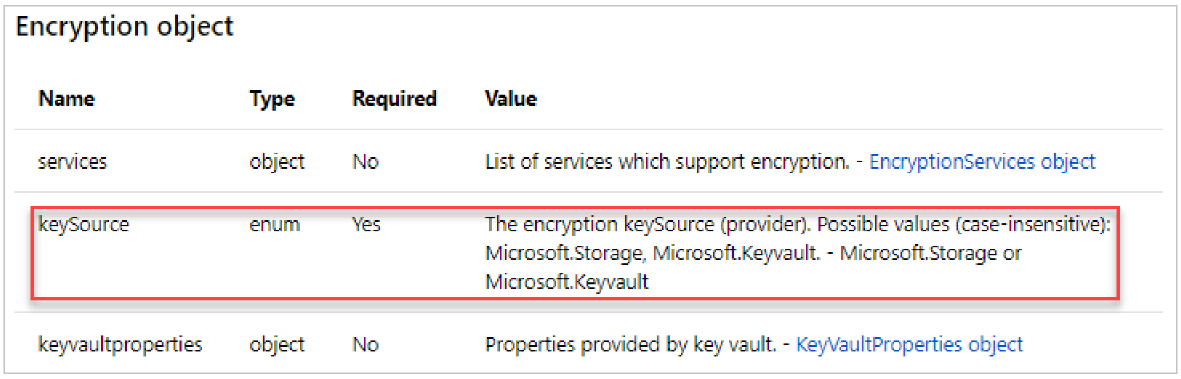
On the same web page, the following description confirms the `encryption` object is used to create an encrypted storage account.



StorageAccountPropertiesCreateParameters object

Name	Type	Required	Value
customDomain	object	No	User domain assigned to the storage account. Name is the CNAME source. Only one custom domain is supported per storage account at this time. To clear the existing custom domain, use an empty string for the custom domain name property. - CustomDomain object
encryption	object	No	Provides the encryption settings on the account. If left unspecified the account encryption settings will remain the same. The default setting is unencrypted. - Encryption object

And there are two ways for managing the encryption key. You can use Microsoft-managed encryption keys with Storage Service Encryption, or you can use your own encryption keys. To keep this tutorial simple, you use the `Microsoft.Storage` option, so you don't have to create an Azure Key Vault.



Encryption object

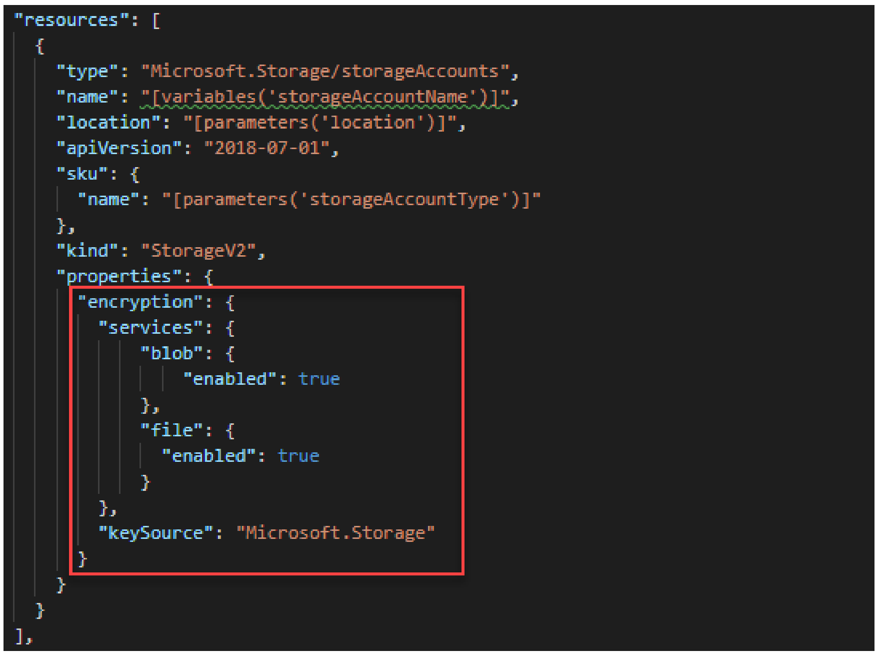
Name	Type	Required	Value
services	object	No	List of services which support encryption. - EncryptionServices object
keySource	enum	Yes	The encryption keySource (provider). Possible values (case-insensitive): Microsoft.Storage, Microsoft.Keyvault. - Microsoft.Storage or Microsoft.Keyvault
keyvaultproperties	object	No	Properties provided by key vault. - KeyVaultProperties object

Your encryption object shall look like:

```
"encryption": {
    "services": {
        "blob": {
            "enabled": true
        },
        "file": {
          "enabled": true
        }
    },
    "keySource": "Microsoft.Storage"
}
```

Edit the template

From Visual Studio Code, modify the template so that the resources element looks like:

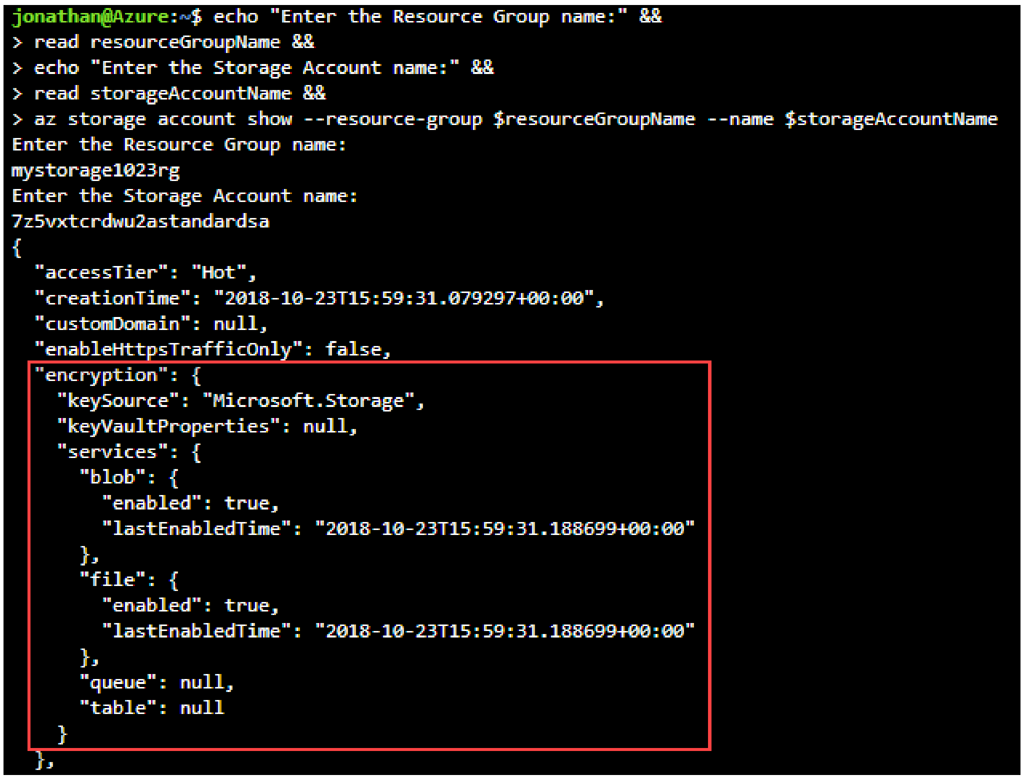


```
"resources": [
  {
    "type": "Microsoft.Storage/storageAccounts",
    "name": "[variables('storageAccountName')]",
    "location": "[parameters('location')]",
    "apiVersion": "2018-07-01",
    "sku": {
      "name": "[parameters('storageAccountType')]"
    },
    "kind": "StorageV2",
    "properties": {
      "encryption": {
        "services": {
          "blob": {
            "enabled": true
          },
          "file": {
          "enabled": true
          }
        },
        "keySource": "Microsoft.Storage"
      }
    }
  }
],
```

Deploy the template

Refer to the Deploy the template section in the Visual Studio Code quickstart for the deployment procedure.

The following screenshot shows the CLI command for listing the newly created storage account, which indicates encryption has been enabled for the blob storage.



```
jonathan@Azure:~$ echo "Enter the Resource Group name:" &&
> read resourceGroupName &&
> echo "Enter the Storage Account name:" &&
> read storageAccountName &&
> az storage account show --resource-group $resourceGroupName --name $storageAccountName
Enter the Resource Group name:
mystorage1023rg
Enter the Storage Account name:
7z5vxtcrdwu2astandardsa
{
  "accessTier": "Hot",
  "creationTime": "2018-10-23T15:59:31.079297+00:00",
  "customDomain": null,
  "enableHttpsTrafficOnly": false,
  "encryption": {
    "keySource": "Microsoft.Storage",
    "keyVaultProperties": null,
    "services": {
      "blob": {
        "enabled": true,
        "lastEnabledTime": "2018-10-23T15:59:31.188699+00:00"
      },
      "file": {
        "enabled": true,
        "lastEnabledTime": "2018-10-23T15:59:31.188699+00:00"
      },
      "queue": null,
      "table": null
    }
  },
```

Clean up resources

When the Azure resources are no longer needed, clean up the resources you deployed by deleting the resource group.

1. From the Azure portal, select **Resource group** from the left menu.
2. Enter the resource group name in the **Filter by name** field.
3. Select the resource group name. You shall see a total of six resources in the resource group.

4. Select **Delete resource group** from the top menu.

Next steps

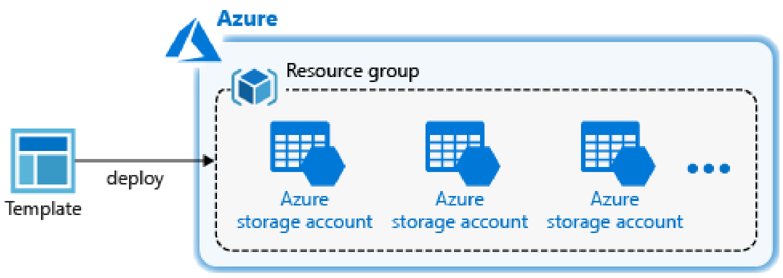
In this tutorial, you learned how to use template reference to customize an existing template. To learn how to create multiple storage account instances, see:

Create multiple instances

Tutorial: Create multiple resource instances with Resource Manager templates

4/7/2019 • 3 minutes to read • Edit Online

Learn how to iterate in your Azure Resource Manager template to create multiple instances of an Azure resource. In this tutorial, you modify a template to create three storage account instances.



This tutorial covers the following tasks:

- Open a QuickStart template
- Edit the template
- Deploy the template

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

To complete this article, you need:

- Visual Studio Code with Resource Manager Tools extension.

Open a Quickstart template

Azure QuickStart Templates is a repository for Resource Manager templates. Instead of creating a template from scratch, you can find a sample template and customize it. The template used in this quickstart is called Create a standard storage account. The template defines an Azure Storage account resource.

1. From Visual Studio Code, select **File**>**Open File**.
2. In **File name**, paste the following URL:

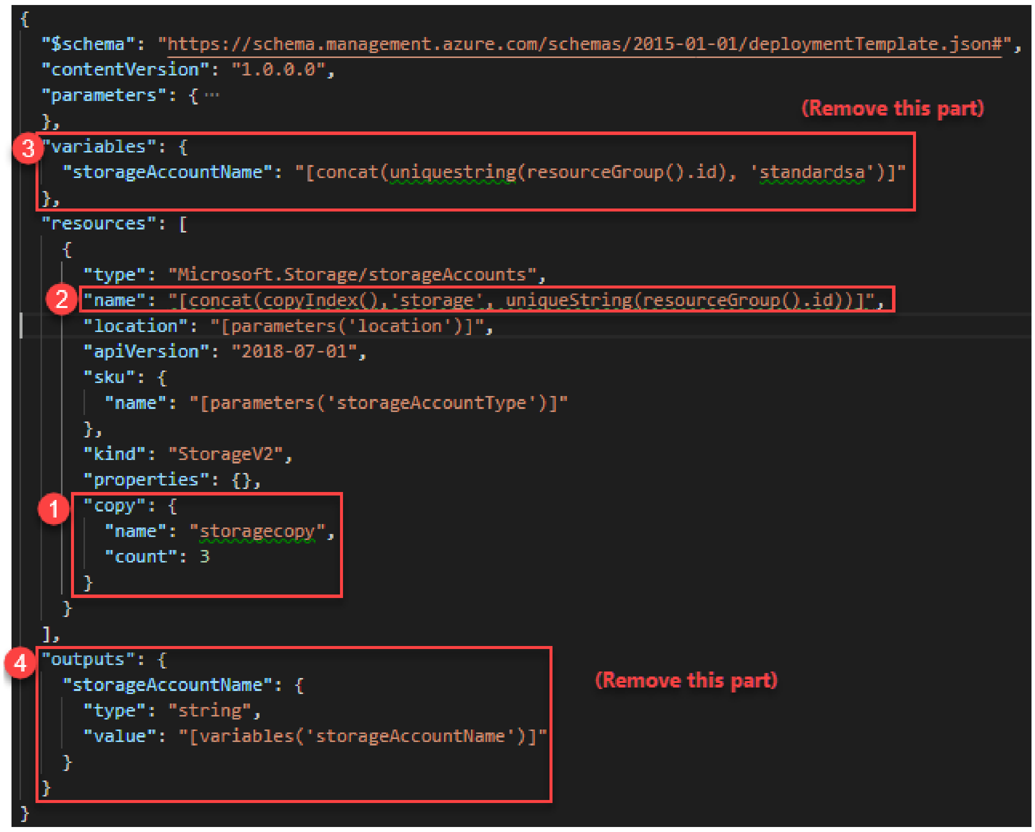
```
https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-storage-account-
create/azuredeploy.json
```

3. Select **Open** to open the file.
4. There is a 'Microsoft.Storage/storageAccounts' resource defined in the template. Compare the template to the template reference. It is helpful to get some basic understanding of the template before customizing it.
5. Select **File**>**Save As** to save the file as **azuredeploy.json** to your local computer.

Edit the template

The existing template creates one storage account. You customize the template to create three storage accounts.

From Visual Studio Code, make the following four changes:



```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {…
  },
  "variables": {
    "storageAccountName": "[concat(uniquestring(resourceGroup().id), 'standardsa')]"
  },
  "resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "name": "[concat(copyIndex(),'storage', uniqueString(resourceGroup().id))]",
      "location": "[parameters('location')]",
      "apiVersion": "2018-07-01",
      "sku": {
        "name": "[parameters('storageAccountType')]"
      },
      "kind": "StorageV2",
      "properties": {},
      "copy": {
        "name": "storagecopy",
        "count": 3
      }
    }
  ],
  "outputs": {
    "storageAccountName": {
      "type": "string",
      "value": "[variables('storageAccountName')]"
    }
  }
}
```

1. Add a `copy` element to the storage account resource definition. In the copy element, you specify the number of iterations and a variable for this loop. The count value must be a positive integer and can't exceed 800.
2. The `copyIndex()` function returns the current iteration in the loop. You use the index as the name prefix. `copyIndex()` is zero-based. To offset the index value, you can pass a value in the copyIndex() function. For example, *copyIndex(1)*.
3. Delete the **variables** element, because it is not used anymore.
4. Delete the **outputs** element. It is no longer needed.

The completed template looks like:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "storageAccountType": {
      "type": "string",
      "defaultValue": "Standard_LRS",
      "allowedValues": [
        "Standard_LRS",
        "Standard_GRS",
        "Standard_ZRS",
        "Premium_LRS"
      ],
      "metadata": {
        "description": "Storage Account type"
      }
    },
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]",
      "metadata": {
        "description": "Location for all resources."
      }
    }
  },
  "resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "name": "[concat(copyIndex(),'storage', uniqueString(resourceGroup().id))]",
      "apiVersion": "2018-02-01",
      "location": "[parameters('location')]",
      "sku": {
        "name": "[parameters('storageAccountType')]"
      },
      "kind": "Storage",
      "properties": {},
      "copy": {
        "name": "storagecopy",
        "count": 3
      }
    }
  ]
}
```

For more information about creating multiple instances, see Deploy multiple instances of a resource or property in Azure Resource Manager Templates

Deploy the template

Refer to the Deploy the template section in the Visual Studio Code quickstart for the deployment procedure.

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

To list all three storage accounts, omit the --name parameter:

- Azure CLI
- PowerShell

```
echo "Enter the Resource Group name:" &&
read resourceGroupName &&
az storage account list --resource-group $resourceGroupName
```

Compare the storage account names with the name definition in the template.

Clean up resources

When the Azure resources are no longer needed, clean up the resources you deployed by deleting the resource group.

1. From the Azure portal, select **Resource group** from the left menu.
2. Enter the resource group name in the **Filter by name** field.
3. Select the resource group name. You shall see a total of six resources in the resource group.
4. Select **Delete resource group** from the top menu.

Next steps

In this tutorial, you learned how to create multiple storage account instances. In the next tutorial, you develop a template with multiple resources and multiple resource types. Some of the resources have dependent resources.

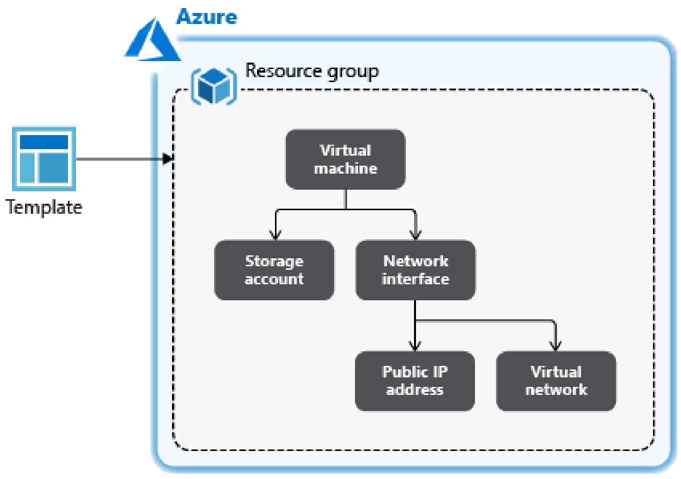
Create dependent resources

Tutorial: Create Azure Resource Manager templates with dependent resources

3/15/2019 • 5 minutes to read • Edit Online

Learn how to create an Azure Resource Manager template to deploy multiple resources and configure the deployment order. After you create the template, you deploy the template using the Cloud shell from the Azure portal.

In this tutorial, you create a storage account, a virtual machine, a virtual network, and some other dependent resources. Some of the resources cannot be deployed until another resource exists. For example, you can't create the virtual machine until its storage account and network interface exist. You define this relationship by making one resource as dependent on the other resources. Resource Manager evaluates the dependencies between resources, and deploys them in their dependent order. When resources aren't dependent on each other, Resource Manager deploys them in parallel. For more information, see Define the order for deploying resources in Azure Resource Manager Templates.



This tutorial covers the following tasks:

- Open a QuickStart template
- Explore the template
- Deploy the template

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

To complete this article, you need:

- Visual Studio Code with the Resource Manager Tools extension. See Install the extension .
- To increase security, use a generated password for the virtual machine administrator account. Here is a sample for generating a password:

  ```
  openssl rand -base64 32
  ```

  Azure Key Vault is designed to safeguard cryptographic keys and other secrets. For more information, see Tutorial: Integrate Azure Key Vault in Resource Manager Template deployment. We also recommend you to update your password every three months.

Open a Quickstart template

Azure QuickStart Templates is a repository for Resource Manager templates. Instead of creating a template from scratch, you can find a sample template and customize it. The template used in this tutorial is called Deploy a simple Windows VM.

1. From Visual Studio Code, select **File**>**Open File**.
2. In **File name**, paste the following URL:

```
https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-vm-simple-
windows/azuredeploy.json
```

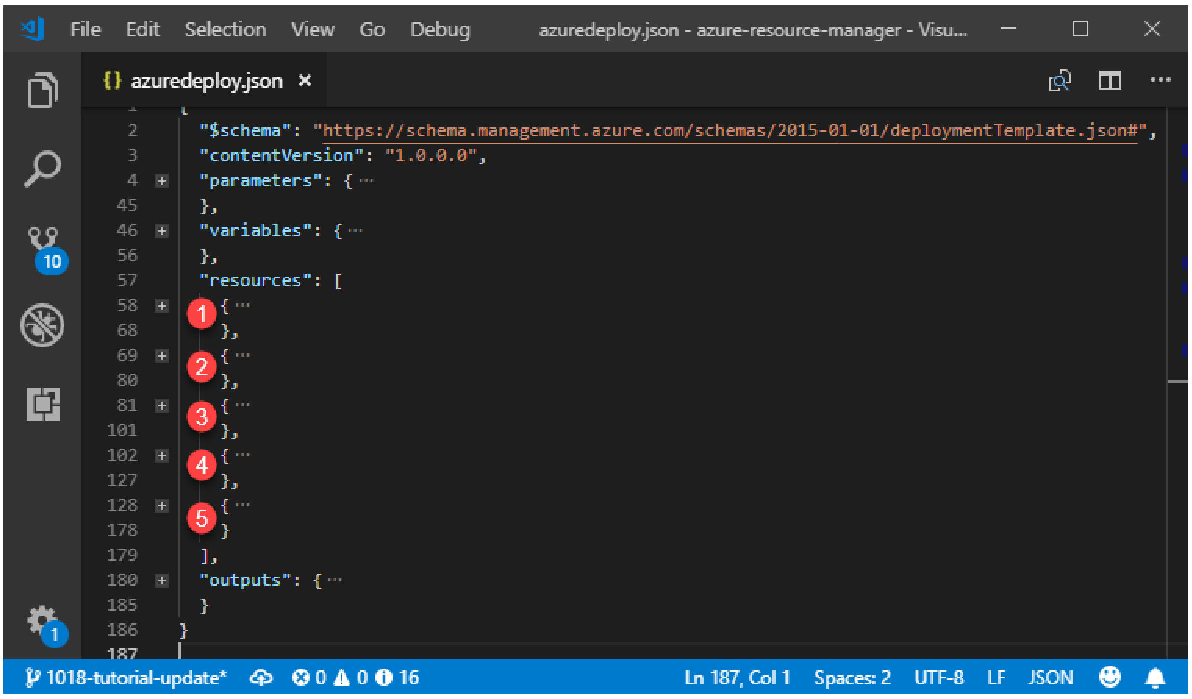
3. Select **Open** to open the file.
4. Select **File**>**Save As** to save a copy of the file to your local computer with the name **azuredeploy.json**.

Explore the template

When you explore the template in this section, try to answer these questions:

- How many Azure resources defined in this template?
- One of the resources is an Azure storage account. Does the definition look like the one used in the last tutorial?
- Can you find the template references for the resources defined in this template?
- Can you find the dependencies of the resources?

1. From Visual Studio Code, collapse the elements until you only see the first-level elements and the second-level elements inside **resources**:



```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {…
  },
  "variables": {…
  },
  "resources": [
    {…
    },
    {…
    },
    {…
    },
    {…
    },
    {…
    }
  ],
  "outputs": {…
  }
}
```

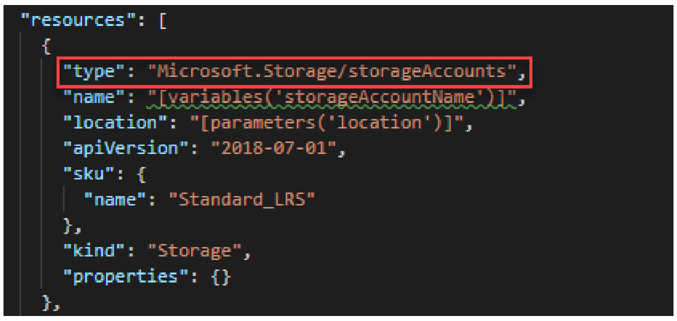
There are five resources defined by the template:

- `Microsoft.Storage/storageAccounts`. See the template reference.
- `Microsoft.Network/publicIPAddresses`. See the template reference.

- `Microsoft.Network/virtualNetworks`. See the template reference.
- `Microsoft.Network/networkInterfaces`. See the template reference.
- `Microsoft.Compute/virtualMachines`. See the template reference.

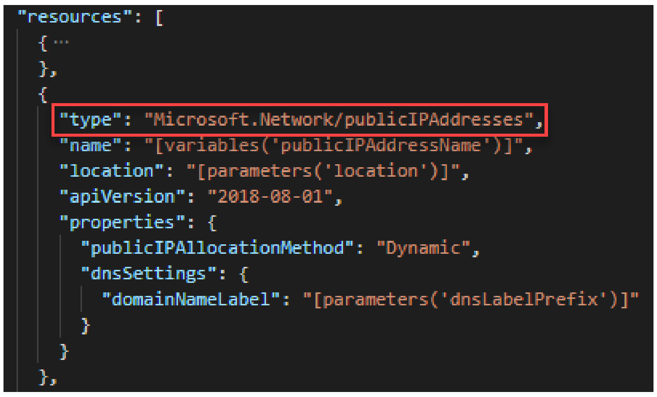
  It is helpful to get some basic understanding of the template before customizing it.

2. Expand the first resource. It is a storage account. Compare the resource definition to the template reference.



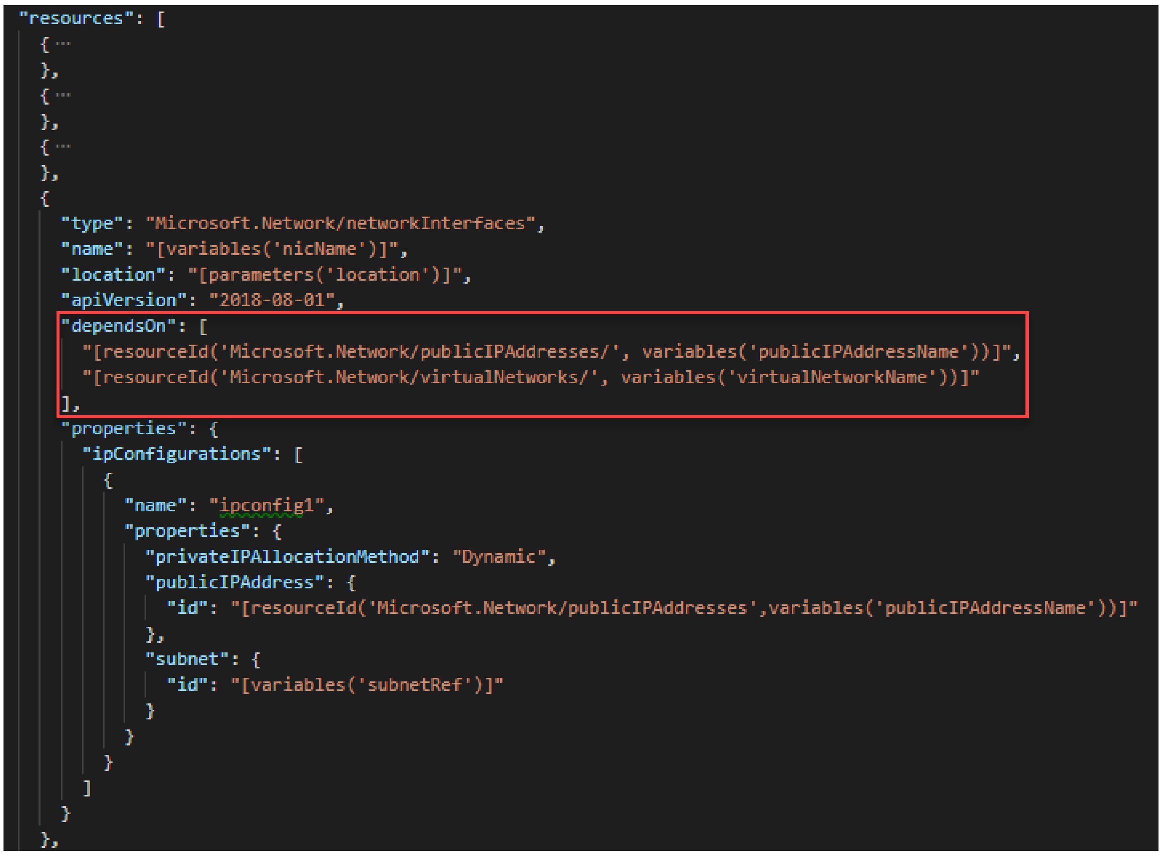
```
"resources": [
  {
    "type": "Microsoft.Storage/storageAccounts",
    "name": "[variables('storageAccountName')]",
    "location": "[parameters('location')]",
    "apiVersion": "2018-07-01",
    "sku": {
      "name": "Standard_LRS"
    },
    "kind": "Storage",
    "properties": {}
  },
```

3. Expand the second resource. The resource type is `Microsoft.Network/publicIPAddresses`. Compare the resource definition to the template reference.



```
"resources": [
  {…
  },
  {
    "type": "Microsoft.Network/publicIPAddresses",
    "name": "[variables('publicIPAddressName')]",
    "location": "[parameters('location')]",
    "apiVersion": "2018-08-01",
    "properties": {
      "publicIPAllocationMethod": "Dynamic",
      "dnsSettings": {
        "domainNameLabel": "[parameters('dnsLabelPrefix')]"
      }
    }
  },
```

4. Expand the fourth resource. The resource type is `Microsoft.Network/networkInterfaces`:



```
"resources": [
  {⋯
  },
  {⋯
  },
  {⋯
  },
  {
    "type": "Microsoft.Network/networkInterfaces",
    "name": "[variables('nicName')]",
    "location": "[parameters('location')]",
    "apiVersion": "2018-08-01",
    "dependsOn": [
      "[resourceId('Microsoft.Network/publicIPAddresses/', variables('publicIPAddressName'))]",
      "[resourceId('Microsoft.Network/virtualNetworks/', variables('virtualNetworkName'))]"
    ],
    "properties": {
      "ipConfigurations": [
        {
          "name": "ipconfig1",
          "properties": {
            "privateIPAllocationMethod": "Dynamic",
            "publicIPAddress": {
              "id": "[resourceId('Microsoft.Network/publicIPAddresses',variables('publicIPAddressName'))]"
            },
            "subnet": {
              "id": "[variables('subnetRef')]"
            }
          }
        }
      ]
    }
  },
```

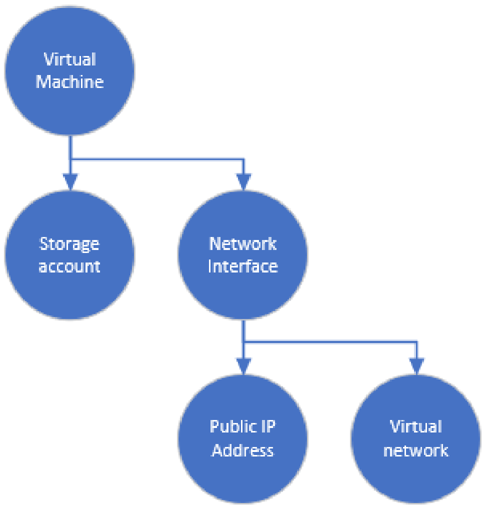
The dependsOn element enables you to define one resource as a dependent on one or more resources. The resource depends on two other resources:

- `Microsoft.Network/publicIPAddresses`
- `Microsoft.Network/virtualNetworks`

5. Expand the fifth resource. This resource is a virtual machine. It depends on two other resources:

- `Microsoft.Storage/storageAccounts`
- `Microsoft.Network/networkInterfaces`

The following diagram illustrates the resources and the dependency information for this template:



By specifying the dependencies, Resource Manager efficiently deploys the solution. It deploys the storage account, public IP address, and virtual network in parallel because they have no dependencies. After the public IP address and virtual network are deployed, the network interface is created. When all other resources are deployed, Resource Manager deploys the virtual machine.

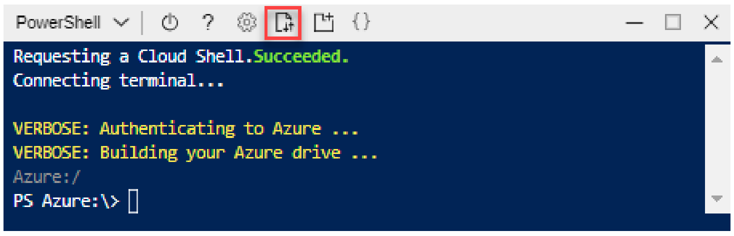
Deploy the template

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

There are many methods for deploying templates. In this tutorial, you use Cloud Shell from the Azure portal.

1. Sign in to the Cloud Shell.
2. Select **PowerShell** from the upper left corner of the Cloud shell, and then select **Confirm**. You use PowerShell in this tutorial.
3. Select **Upload file** from the Cloud shell:



```
Requesting a Cloud Shell.Succeeded.
Connecting terminal...

VERBOSE: Authenticating to Azure ...
VERBOSE: Building your Azure drive ...
Azure:/
PS Azure:\>
```

4. Select the template you saved earlier in the tutorial. The default name is **azuredeploy.json**. If you have a file with the same file name, the old file is overwritten without any notification.

   You can optionally use the **ls $HOME** command and the **cat $HOME/azuredeploy.json** command to verify the files areis uploaded successfully.

5. From the Cloud shell, run the following PowerShell commands. To increase security, use a generated password for the virtual machine administrator account. See Prerequisites.

```
$resourceGroupName = Read-Host -Prompt "Enter the Resource Group name"
$location = Read-Host -Prompt "Enter the location (i.e. centralus)"
$adminUsername = Read-Host -Prompt "Enter the virtual machine admin username"
$adminPassword = Read-Host -Prompt "Enter the admin password" -AsSecureString
$dnsLabelPrefix = Read-Host -Prompt "Enter the DNS label prefix"

New-AzResourceGroup -Name $resourceGroupName -Location "$location"
New-AzResourceGroupDeployment `
    -ResourceGroupName $resourceGroupName `
    -adminUsername $adminUsername `
    -adminPassword $adminPassword `
    -dnsLabelPrefix $dnsLabelPrefix `
    -TemplateFile "$HOME/azuredeploy.json"
```

6. Run the following PowerShell command to list the newly created virtual machine:

```
$resourceGroupName = Read-Host -Prompt "Enter the Resource Group name"
Get-AzVM -Name SimpleWinVM -ResourceGroupName $resourceGroupName
```

   The virtual machine name is hard-coded as **SimpleWinVM** inside the template.

7. RDP to the virtual machine to verify the virtual machine has been created successfully.

Clean up resources

When the Azure resources are no longer needed, clean up the resources you deployed by deleting the resource group.

1. From the Azure portal, select **Resource group** from the left menu.
2. Enter the resource group name in the **Filter by name** field.
3. Select the resource group name. You shall see a total of six resources in the resource group.
4. Select **Delete resource group** from the top menu.

Next steps

In this tutorial, you developed and deployed a template to create a virtual machine, a virtual network, and the dependent resources. To learn how to deploy Azure resources based on conditions, see:

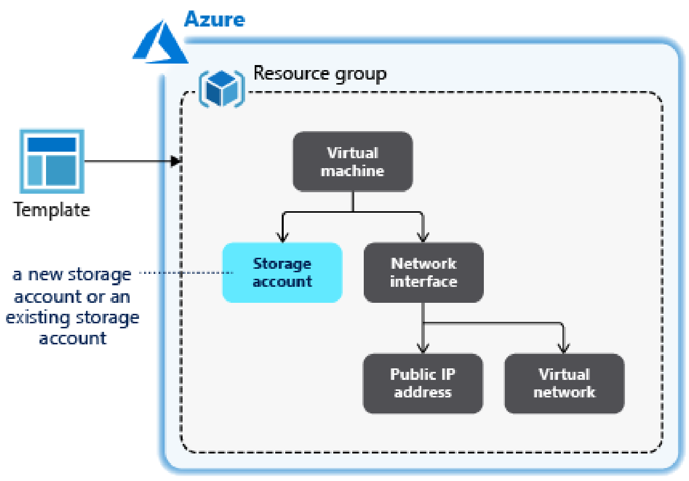
Use conditions

Tutorial: Use condition in Azure Resource Manager templates

5/29/2019 • 4 minutes to read • Edit Online

Learn how to deploy Azure resources based on conditions.

In the Set resource deployment order tutorial, you create a virtual machine, a virtual network, and some other dependent resources including a storage account. Instead of creating a new storage account every time, you let people choose between creating a new storage account and using an existing storage account. To accomplish this goal, you define an additional parameter. If the value of the parameter is "new", a new storage account is created. Otherwise, an existing storage account with the name provided is used.



This tutorial covers the following tasks:

- Open a QuickStart template
- Modify the template
- Deploy the template
- Clean up resources

This tutorial only covers a basic scenario of using conditions. For more information, see:

- Template file structure: Condition.
- Conditionally deploy a resource in an Azure Resource Manager template.
- Template function: If.
- Comparison functions for Azure Resource Manager templates

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

To complete this article, you need:

- Visual Studio Code with Resource Manager Tools extension.
- To increase security, use a generated password for the virtual machine administrator account. Here is a sample for generating a password:

```
openssl rand -base64 32
```

Azure Key Vault is designed to safeguard cryptographic keys and other secrets. For more information, see Tutorial: Integrate Azure Key Vault in Resource Manager Template deployment. We also recommend you to update your password every three months.

Open a Quickstart template

Azure QuickStart Templates is a repository for Resource Manager templates. Instead of creating a template from scratch, you can find a sample template and customize it. The template used in this tutorial is called Deploy a simple Windows VM.

1. From Visual Studio Code, select **File**>**Open File**.
2. In **File name**, paste the following URL:

   ```
   https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-vm-simple-
   windows/azuredeploy.json
   ```

3. Select **Open** to open the file.
4. There are five resources defined by the template:

   - `Microsoft.Storage/storageAccounts`. See the template reference.
   - `Microsoft.Network/publicIPAddresses`. See the template reference.
   - `Microsoft.Network/virtualNetworks`. See the template reference.
   - `Microsoft.Network/networkInterfaces`. See the template reference.
   - `Microsoft.Compute/virtualMachines`. See the template reference.

   It is helpful to get some basic understanding of the template before customizing it.

5. Select **File**>**Save As** to save a copy of the file to your local computer with the name **azuredeploy.json**.

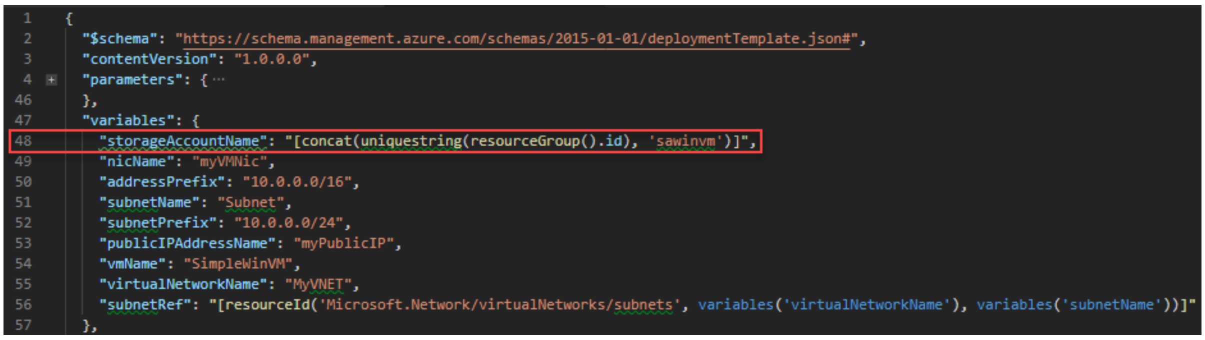
Modify the template

Make two changes to the existing template:

- Add a storage account name parameter. Users can specify either a new storage account name or an existing storage account name.
- Add a new parameter called **newOrExisting**. The deployment uses this parameter to determine where to create a new storage account or use an existing storage account.

Here is the procedure to make the changes:

1. Open **azuredeploy.json** in Visual Studio Code.
2. Replace the three **variables('storageAccountName')** with **parameters('storageAccountName')** in the whole template.
3. Remove the following variable definition:

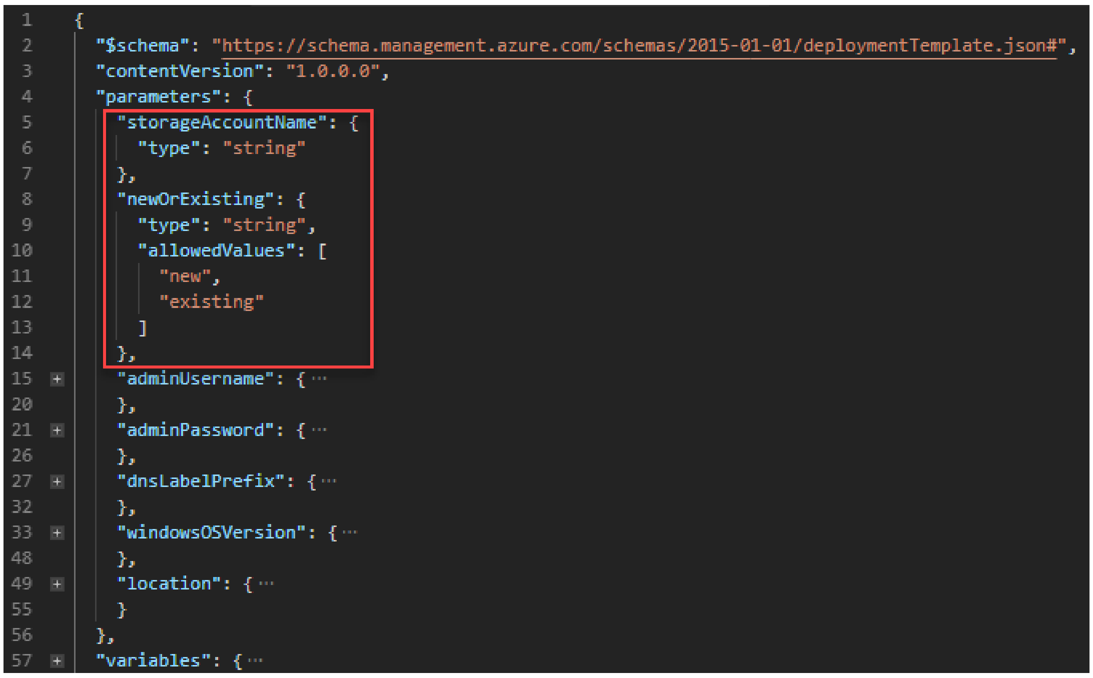


```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {…
  },
  "variables": {
    "storageAccountName": "[concat(uniquestring(resourceGroup().id), 'sawinvm')]",
    "nicName": "myVMNic",
    "addressPrefix": "10.0.0.0/16",
    "subnetName": "Subnet",
    "subnetPrefix": "10.0.0.0/24",
    "publicIPAddressName": "myPublicIP",
    "vmName": "SimpleWinVM",
    "virtualNetworkName": "MyVNET",
    "subnetRef": "[resourceId('Microsoft.Network/virtualNetworks/subnets', variables('virtualNetworkName'), variables('subnetName'))]"
  },
```

4. Add the following two parameters to the template:

```
"storageAccountName": {
  "type": "string"
},
"newOrExisting": {
  "type": "string",
  "allowedValues": [
    "new",
    "existing"
  ]
},
```

The updated parameters definition looks like:



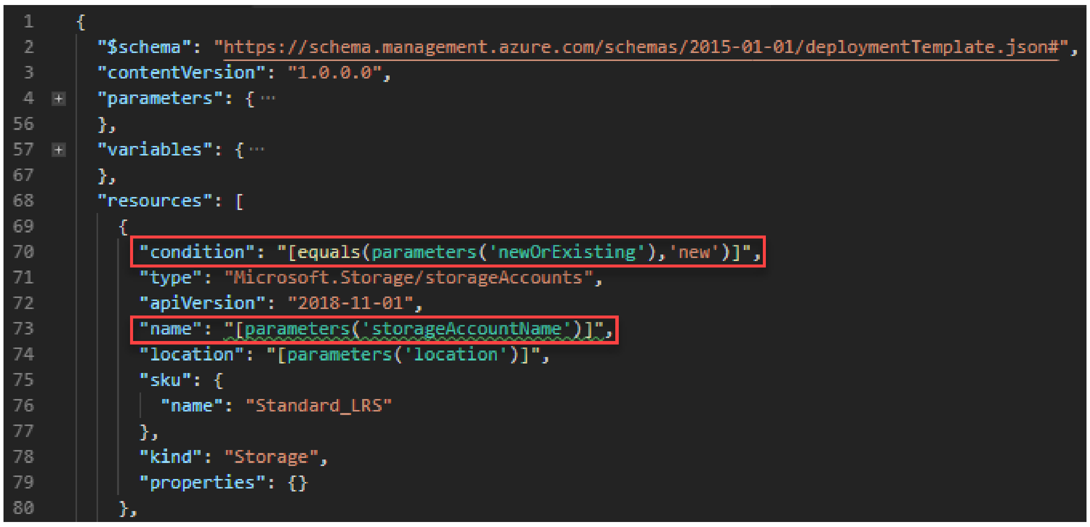
```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "storageAccountName": {
      "type": "string"
    },
    "newOrExisting": {
      "type": "string",
      "allowedValues": [
        "new",
        "existing"
      ]
    },
    "adminUsername": {…
    },
    "adminPassword": {…
    },
    "dnsLabelPrefix": {…
    },
    "windowsOSVersion": {…
    },
    "location": {…
    }
  },
  "variables": {…
```

5. Add the following line to the beginning of the storage account definition.

```
"condition": "[equals(parameters('newOrExisting'),'new')]",
```

The condition checks the value of a parameter called **newOrExisting**. If the parameter value is **new**, the deployment creates the storage account.

The updated storage account definition looks like:



```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {…
  },
  "variables": {…
  },
  "resources": [
    {
      "condition": "[equals(parameters('newOrExisting'),'new')]",
      "type": "Microsoft.Storage/storageAccounts",
      "apiVersion": "2018-11-01",
      "name": "[parameters('storageAccountName')]",
      "location": "[parameters('location')]",
      "sku": {
        "name": "Standard_LRS"
      },
      "kind": "Storage",
      "properties": {}
    },
```

6. Update the **storageUri** property of the virtual machine resource definition with the following value:

```
"storageUri": "[concat('https://', parameters('storageAccountName'), '.blob.core.windows.net')]"
```

   This change is necessary when you use an existing storage account under a different resource group.

7. Save the changes.

Deploy the template

Follow the instructions in Deploy the template to open the Cloud shell and upload the revised template, and then run the follow PowerShell script to deploy the template.

```
$resourceGroupName = Read-Host -Prompt "Enter the resource group name"
$storageAccountName = Read-Host -Prompt "Enter the storage account name"
$newOrExisting = Read-Host -Prompt "Create new or use existing (Enter new or existing)"
$location = Read-Host -Prompt "Enter the Azure location (i.e. centralus)"
$vmAdmin = Read-Host -Prompt "Enter the admin username"
$vmPassword = Read-Host -Prompt "Enter the admin password" -AsSecureString
$dnsLabelPrefix = Read-Host -Prompt "Enter the DNS Label prefix"

New-AzResourceGroup -Name $resourceGroupName -Location $location
New-AzResourceGroupDeployment `
    -ResourceGroupName $resourceGroupName `
    -adminUsername $vmAdmin `
    -adminPassword $vmPassword `
    -dnsLabelPrefix $dnsLabelPrefix `
    -storageAccountName $storageAccountName `
    -newOrExisting $newOrExisting `
    -TemplateFile "$HOME/azuredeploy.json"
```

NOTE

The deployment fails if **newOrExisting** is **new**, but the storage account with the storage account name specified already exists.

Try making another deployment with **newOrExisting** set to "existing" and specify an exiting storage account. To create a storage account beforehand, see Create a storage account.

Clean up resources

When the Azure resources are no longer needed, clean up the resources you deployed by deleting the resource group. To delete the resource group, select **Try it** to open the Cloud shell. To paste the PowerShell script, right-click the shell pane, and then select **Paste**.

```
$resourceGroupName = Read-Host -Prompt "Enter the same resource group name you used in the last procedure"
Remove-AzResourceGroup -Name $resourceGroupName
```

Next steps

In this tutorial, you developed a template that allows users to choose between creating a new storage account and using an existing storage account. To learn how to retrieve secrets from Azure Key Vault, and use the secrets as passwords in the template deployment, see:

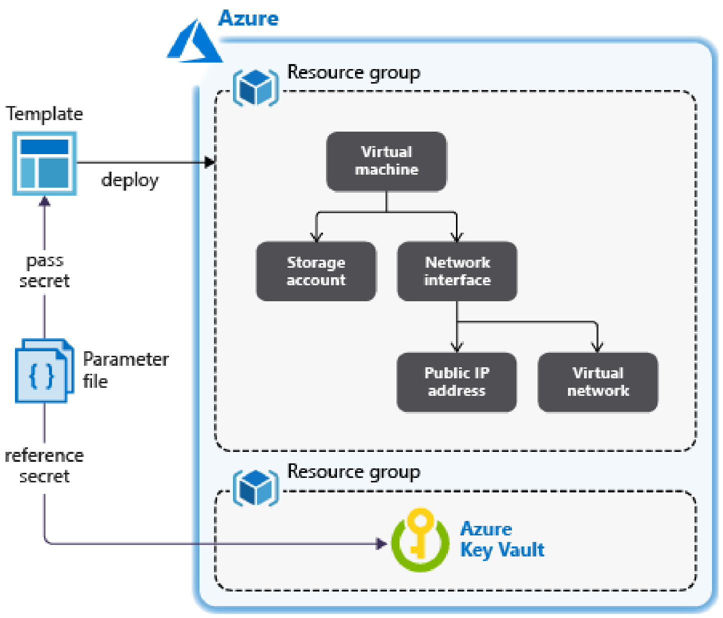
Integrate Key Vault in template deployment

Tutorial: Integrate Azure Key Vault in your Resource Manager template deployment

6/26/2019 • 6 minutes to read • Edit Online

Learn how to retrieve secrets from an Azure key vault and pass the secrets as parameters when you deploy Azure Resource Manager. The parameter value is never exposed, because you reference only its key vault ID. For more information, see Use Azure Key Vault to pass secure parameter value during deployment.

In the Set resource deployment order tutorial, you create a virtual machine (VM). You need to provide the VM administrator username and password. Instead of providing the password, you can pre-store the password in an Azure key vault and then customize the template to retrieve the password from the key vault during the deployment.



This tutorial covers the following tasks:

- Prepare a key vault
- Open a quickstart template
- Edit the parameters file
- Deploy the template
- Validate the deployment
- Clean up resources

If you don't have an Azure subscription, create a free account before you begin.

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

Prerequisites

To complete this article, you need:

- Visual Studio Code with the Resource Manager Tools extension.
- To increase security, use a generated password for the VM administrator account. Here's a sample for generating a password:

  ```
  openssl rand -base64 32
  ```

  Verify that the generated password meets the VM password requirements. Each Azure service has specific password requirements. For the VM password requirements, see What are the password requirements when you create a VM?.

Prepare a key vault

In this section, you create a key vault and add a secret to it, so that you can retrieve the secret when you deploy your template. There are many ways to create a key vault. In this tutorial, you use Azure PowerShell to deploy a Resource Manager template. This template does the following:

- Creates a key vault with the `enabledForTemplateDeployment` property enabled. This property must be *true* before the template deployment process can access the secrets that are defined in the key vault.
- Adds a secret to the key vault. The secret stores the VM administrator password.

NOTE

As the user who's deploying the virtual machine template, if you're not the Owner of or a Contributor to the key vault, the Owner or a Contributor must grant you access to the *Microsoft.KeyVault/vaults/deploy/action* permission for the key vault. For more information, see Use Azure Key Vault to pass a secure parameter value during deployment.

To run the following Azure PowerShell script, select **Try it** to open Azure Cloud Shell. To paste the script, right-click the shell pane, and then select **Paste**.

```
$projectName = Read-Host -Prompt "Enter a project name that is used for generating resource names"
$location = Read-Host -Prompt "Enter the location (i.e. centralus)"
$upn = Read-Host -Prompt "Enter your user principal name (email address) used to sign in to Azure"
$secretValue = Read-Host -Prompt "Enter the virtual machine administrator password" -AsSecureString

$resourceGroupName = "${projectName}rg"
$keyVaultName = $projectName
$adUserId = (Get-AzADUser -UserPrincipalName $upn).Id
$templateUri = "https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/tutorials-use-key-vault/CreateKeyVault.json"

New-AzResourceGroup -Name $resourceGroupName -Location $location
New-AzResourceGroupDeployment -ResourceGroupName $resourceGroupName -TemplateUri $templateUri -keyVaultName $keyVaultName -adUserId $adUserId -secretValue $secretValue
```

IMPORTANT

- The resource group name is the project name, but with **rg** appended to it. To make it easier to clean up the resources that you created in this tutorial, use the same project name and resource group name when you deploy the next template.
- The default name for the secret is **vmAdminPassword**. It's hardcoded in the template.
- To enable the template to retrieve the secret, you must enable an access policy called "Enable access to Azure Resource Manager for template deployment" for the key vault. This policy is enabled in the template. For more information about the access policy, see Deploy key vaults and secrets.

The template has one output value, called *keyVaultId*. Write down the ID value for later use, when you deploy the virtual machine. The resource ID format is:

```
/subscriptions/<SubscriptionID>/resourceGroups/mykeyvaultdeploymentrg/providers/Microsoft.KeyVault/vaults/<Ke
yVaultName>
```

When you copy and paste the ID, it might be broken into multiple lines. Merge the lines and trim the extra spaces.

To validate the deployment, run the following PowerShell command in the same shell pane to retrieve the secret in clear text. The command works only in the same shell session, because it uses the variable *$keyVaultName*, which is defined in the preceding PowerShell script.

```
(Get-AzKeyVaultSecret -vaultName $keyVaultName  -name "vmAdminPassword").SecretValueText
```

Now you've prepared a key vault and a secret. The following sections show you how to customize an existing template to retrieve the secret during the deployment.

Open a quickstart template

Azure Quickstart Templates is a repository for Resource Manager templates. Instead of creating a template from scratch, you can find a sample template and customize it. The template that's used in this tutorial is called Deploy a simple Windows VM.

1. In Visual Studio Code, select **File** > **Open File**.
2. In the **File name** box, paste the following URL:

   ```
   https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-vm-simple-
   windows/azuredeploy.json
   ```

3. Select **Open** to open the file. The scenario is the same as the one that's used in Tutorial: Create Azure Resource Manager templates with dependent resources. The template defines five resources:

   - `Microsoft.Storage/storageAccounts`. See the template reference.
   - `Microsoft.Network/publicIPAddresses`. See the template reference.
   - `Microsoft.Network/virtualNetworks`. See the template reference.
   - `Microsoft.Network/networkInterfaces`. See the template reference.
   - `Microsoft.Compute/virtualMachines`. See the template reference.

   It's helpful to have some basic understanding of the template before you customize it.

4. Select **File** > **Save As**, and then save a copy of the file to your local computer with the name *azuredeploy.json*.

5. Repeat steps 1-3 to open the following URL, and then save the file as *azuredeploy.parameters.json*.

```
https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-vm-simple-
windows/azuredeploy.parameters.json
```

Edit the parameters file

You don't need to make any changes to the template file.

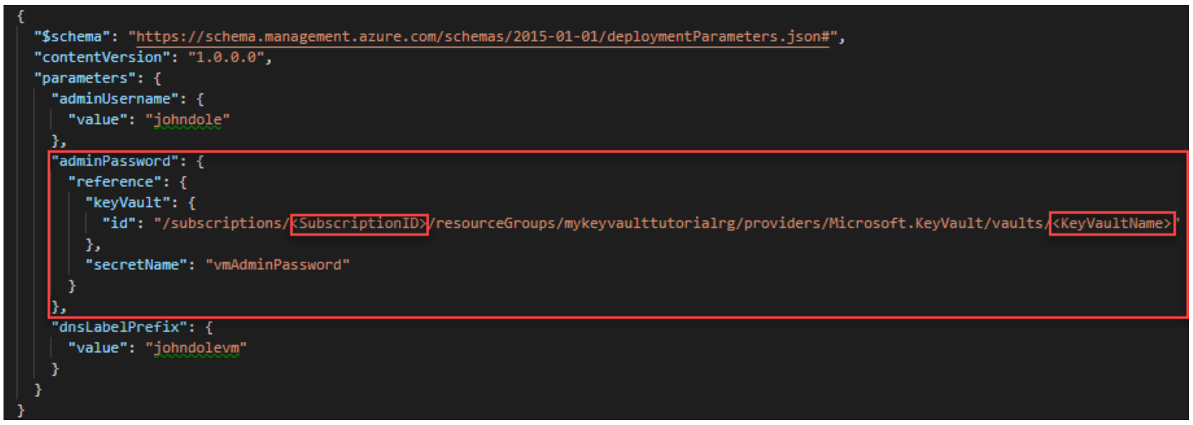
1. In Visual Studio Code, open *azuredeploy.parameters.json* if it's not already open.

2. Update the `adminPassword` parameter to:

```
"adminPassword": {
    "reference": {
        "keyVault": {
        "id":
"/subscriptions/<SubscriptionID>/resourceGroups/mykeyvaultdeploymentrg/providers/Microsoft.KeyVault/vau
lts/<KeyVaultName>"
        },
        "secretName": "vmAdminPassword"
    }
},
```

IMPORTANT

Replace the value for **id** with the resource ID of the key vault that you created in the previous procedure.



```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "adminUsername": {
      "value": "johndole"
    },
    "adminPassword": {
      "reference": {
        "keyVault": {
          "id": "/subscriptions/<SubscriptionID>/resourceGroups/mykeyvaulttutorialrg/providers/Microsoft.KeyVault/vaults/<KeyVaultName>"
        },
        "secretName": "vmAdminPassword"
      }
    },
    "dnsLabelPrefix": {
      "value": "johndolevm"
    }
  }
}
```

3. Update the following values:

   - **adminUsername**: The name of the virtual machine administrator account.
   - **dnsLabelPrefix**: Name the dnsLabelPrefix value.

   For examples of names, see the preceding image.

4. Save the changes.

Deploy the template

Follow the instructions in Deploy the template. Upload both *azuredeploy.json* and *azuredeploy.parameters.json* to Cloud Shell, and then use the following PowerShell script to deploy the template:

```
$projectName = Read-Host -Prompt "Enter the same project name that is used for creating the key vault"
$location = Read-Host -Prompt "Enter the same location that is used for creating the key vault (i.e.
centralus)"
$resourceGroupName = "${projectName}rg"

New-AzResourceGroupDeployment `
    -ResourceGroupName $resourceGroupName `
    -TemplateFile "$HOME/azuredeploy.json" `
    -TemplateParameterFile "$HOME/azuredeploy.parameters.json"
```

When you deploy the template, use the same resource group that you used in the key vault. This approach makes it easier for you to clean up the resources, because you need to delete only one resource group instead of two.

Validate the deployment

After you've successfully deployed the virtual machine, test the sign-in credentials by using the password that's stored in the key vault.

1. Open the Azure portal.
2. Select **Resource groups** > **<*YourResourceGroupName*>** > **simpleWinVM**.
3. Select **connect** at the top.
4. Select **Download RDP File**, and then follow the instructions to sign in to the virtual machine by using the password that's stored in the key vault.

Clean up resources

When you no longer need your Azure resources, clean up the resources that you deployed by deleting the resource group.

```
$projectName = Read-Host -Prompt "Enter the same project name that is used for creating the key vault"
$resourceGroupName = "${projectName}rg"

Remove-AzResourceGroup -Name $resourceGroupName
```

Next steps

In this tutorial, you retrieved a secret from your Azure key vault. You then used the secret in your template deployment. To learn how to create linked templates, see:

Create linked templates

Tutorial: Create linked Azure Resource Manager templates

5/23/2019 • 8 minutes to read • Edit Online

Learn how to create linked Azure Resource Manager templates. Using linked templates, you can have one template call another template. It is great for modularizing templates. In this tutorial, you use the same template used in Tutorial: Create Azure Resource Manager templates with dependent resources, which creates a virtual machine, a virtual network, and other dependent resource including a storage account. You separate the storage account resource creation to a linked template.

Calling a linked template is like making a function call. You also learn how to pass parameter values to the linked template, and how to get "return values" from the linked template.

This tutorial covers the following tasks:

- Open a QuickStart template
- Create the linked template
- Upload the linked template
- Link to the linked template
- Configure dependency
- Deploy the template
- Additional practices

For more information, see Use linked and nested templates when deploying Azure resources.

If you don't have an Azure subscription, create a free account before you begin.

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

Prerequisites

To complete this article, you need:

- Visual Studio Code with Resource Manager Tools extension.
- To increase security, use a generated password for the virtual machine administrator account. Here is a sample for generating a password:

  ```
  openssl rand -base64 32
  ```

  Azure Key Vault is designed to safeguard cryptographic keys and other secrets. For more information, see Tutorial: Integrate Azure Key Vault in Resource Manager Template deployment. We also recommend you to update your password every three months.

Open a Quickstart template

Azure QuickStart Templates is a repository for Resource Manager templates. Instead of creating a template from scratch, you can find a sample template and customize it. The template used in this tutorial is called Deploy a simple Windows VM. This is the same template used in Tutorial: Create Azure Resource Manager templates with dependent resources. You save two copies of the same template to be used as:

- **The main template**: create all the resources except the storage account.
- **The linked template**: create the storage account.

1. From Visual Studio Code, select **File**>**Open File**.
2. In **File name**, paste the following URL:

   ```
   https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-vm-simple-
   windows/azuredeploy.json
   ```

3. Select **Open** to open the file.
4. There are five resources defined by the template:

   - `Microsoft.Storage/storageAccounts`
   - `Microsoft.Network/publicIPAddresses`
   - `Microsoft.Network/virtualNetworks`
   - `Microsoft.Network/networkInterfaces`
   - `Microsoft.Compute/virtualMachines`

   It is helpful to get some basic understanding of the template schema before customizing the template.

5. Select **File**>**Save As** to save a copy of the file to your local computer with the name **azuredeploy.json**.
6. Select **File**>**Save As** to create another copy of the file with the name **linkedTemplate.json**.

Create the linked template

The linked template creates a storage account. The linked template can be used as a standalone template to create a storage account. In this tutorial, the linked template takes two parameters, and passes a value back to the main template. This "return" value is defined in the `outputs` element.

1. Open **linkedTemplate.json** in Visual Studio Code if the file is not opened.
2. Make the following changes:

   - Remove all the parameters other than **location**.
   - Add a parameter called **storageAccountName**.

     ```
     "storageAccountName":{
       "type": "string",
       "metadata": {
           "description": "Azure Storage account name."
       }
     },
     ```

     The storage account name and location are passed from the main template to the linked template as parameters.

- Remove the **variables** element, and all the variable definitions.
- Remove all the resources other than the storage account. You remove a total of four resources.
- Update the value of the **name** element of the storage account resource to:

```
  "name": "[parameters('storageAccountName')]",
```

- Update the **outputs** element, so it looks like:

```
"outputs": {
  "storageUri": {
      "type": "string",
      "value": "[reference(parameters('storageAccountName')).primaryEndpoints.blob]"
    }
}
```

storageUri is required by the virtual machine resource definition in the main template. You pass the value back to the main template as an output value.

When you are done, the template shall look like:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "storageAccountName": {
      "type": "string",
      "metadata": {
        "description": "Azure Storage account name."
      }
    },
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]",
      "metadata": {
        "description": "Location for all resources."
      }
    }
  },
  "resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "name": "[parameters('storageAccountName')]",
      "location": "[parameters('location')]",
      "apiVersion": "2018-07-01",
      "sku": {
        "name": "Standard_LRS"
      },
      "kind": "Storage",
      "properties": {}
    }
  ],
  "outputs": {
    "storageUri": {
      "type": "string",
      "value": "[reference(parameters('storageAccountName')).primaryEndpoints.blob]"
    }
  }
}
```

3. Save the changes.

Upload the linked template

The main template and the linked template need to be accessible from where you run the deployment. In this tutorial, you use the Cloud shell deployment method as you used in Tutorial: Create Azure Resource Manager templates with dependent resources. The main template (azuredeploy.json) is uploaded to the shell. The linked template (linkedTemplate.json) must be shared somewhere securely. The following PowerShell script creates an Azure Storage account, uploads the template to the Storage account, and then generates a SAS token to grant limited access to the template file. To simplify the tutorial, the script downloads a completed linked template from a shared location. If you want to use the linked template you created, you can use the Cloud shell to upload your linked template, and then modify the script to use your own linked template.

NOTE

The script limits the SAS token to be used within eight hours. If you need more time to complete this tutorial, increase the expiry time.

```
$projectNamePrefix = Read-Host -Prompt "Enter a project name:"   # This name is used to generate names for
Azure resources, such as storage account name.
$location = Read-Host -Prompt "Enter a location (i.e. centralus)"

$resourceGroupName = $projectNamePrefix + "rg"
$storageAccountName = $projectNamePrefix + "store"
$containerName = "linkedtemplates" # The name of the Blob container to be created.

$linkedTemplateURL = "https://armtutorials.blob.core.windows.net/linkedtemplates/linkedStorageAccount.json" #
A completed linked template used in this tutorial.
$fileName = "linkedStorageAccount.json" # A file name used for downloading and uploading the linked template.

# Download the tutorial linked template
Invoke-WebRequest -Uri $linkedTemplateURL -OutFile "$home/$fileName"

# Create a resource group
New-AzResourceGroup -Name $resourceGroupName -Location $location

# Create a storage account
$storageAccount = New-AzStorageAccount `
    -ResourceGroupName $resourceGroupName `
    -Name $storageAccountName `
    -Location $location `
    -SkuName "Standard_LRS"

$context = $storageAccount.Context

# Create a container
New-AzStorageContainer -Name $containerName -Context $context

# Upload the linked template
Set-AzStorageBlobContent `
    -Container $containerName `
    -File "$home/$fileName" `
    -Blob $fileName `
    -Context $context

# Generate a SAS token
$templateURI = New-AzStorageBlobSASToken `
    -Context $context `
    -Container $containerName `
    -Blob $fileName `
    -Permission r `
    -ExpiryTime (Get-Date).AddHours(8.0) `
    -FullUri

echo "You need the following values later in the tutorial:"
echo "Resource Group Name: $resourceGroupName"
echo "Linked template URI with SAS token: $templateURI"
```

1. Select the **Try It** green button to open the Azure cloud shell pane.
2. Select **Copy** to copy the PowerShell script.
3. Right-click anywhere inside the shell pane (the navy blue part), and then select **Paste**.
4. Make a note of the two values (Resource Group Name and Linked template URI) at the end of the shell pane. You need the values later in the tutorial.
5. Select **Exit focus mode** to close the shell pane.

In practice, you generate a SAS token when you deploy the main template, and give the SAS token expiry a smaller window to make it more secure. For more information, see Provide SAS token during deployment.

Call the linked template

The main template is called azuredeploy.json.

1. Open **azuredeploy.json** in Visual Studio Code if it is not opened.

2. Delete the storage account resource definition from the template:

   ```
   {
     "type": "Microsoft.Storage/storageAccounts",
     "name": "[variables('storageAccountName')]",
     "location": "[parameters('location')]",
     "apiVersion": "2018-07-01",
     "sku": {
       "name": "Standard_LRS"
     },
     "kind": "Storage",
     "properties": {}
   },
   ```

3. Add the following json snippet to the place where you had the storage account definition:

   ```
   {
     "name": "linkedTemplate",
     "type": "Microsoft.Resources/deployments",
     "apiVersion": "2018-05-01",
     "properties": {
         "mode": "Incremental",
         "templateLink": {
             "uri":"https://armtutorials.blob.core.windows.net/linkedtemplates/linkedStorageAccount.json"
         },
         "parameters": {
             "storageAccountName":{"value": "[variables('storageAccountName')]"},
             "location":{"value": "[parameters('location')]"}
         }
     }
   },
   ```

   Pay attention to these details:

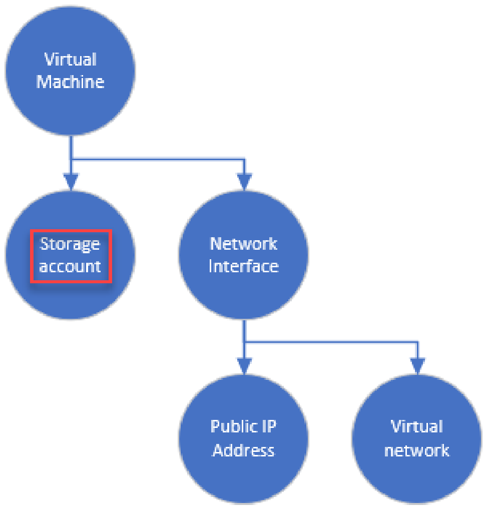
   - A `Microsoft.Resources/deployments` resource in the main template is used to link to another template.
   - The `deployments` resource has a name called `linkedTemplate`. This name is used for configuring dependency.
   - You can only use Incremental deployment mode when calling linked templates.
   - `templateLink/uri` contains the linked template URI. Update the value to the URI you get when you upload the linked template (the one with a SAS token).
   - Use `parameters` to pass values from the main template to the linked template.

4. Make sure you have updated the value of the `uri` element to the value you got when you upload the linked template (the one with a SAS token). In practice, you want to supply the URI with a parameter.

5. Save the revised template

Configure dependency

Recall from Tutorial: Create Azure Resource Manager templates with dependent resources, the virtual machine resource depends on the storage account:



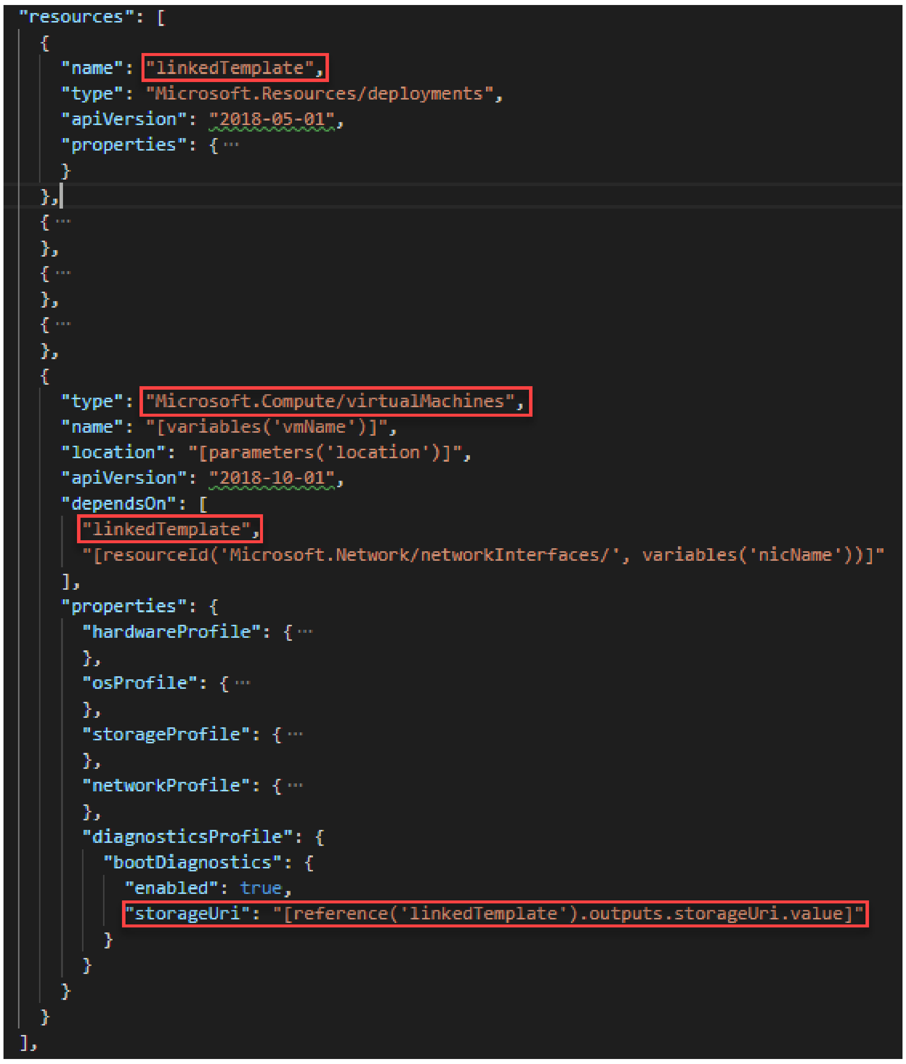
Because the storage account is defined in the linked template now, you must update the following two elements of the `Microsoft.Compute/virtualMachines` resource.

- Reconfigure the `dependOn` element. The storage account definition is moved to the linked template.
- Reconfigure the `properties/diagnosticsProfile/bootDiagnostics/storageUri` element. In Create the linked template, you added an output value:

  ```
  "outputs": {
      "storageUri": {
          "type": "string",
          "value": "[reference(parameters('storageAccountName')).primaryEndpoints.blob]"
          }
  }
  ```

  This value is required by the main template.

1. Open azuredeploy.json in Visual Studio Code if it is not opened.
2. Expand the virtual machine resource definition, update **dependsOn** as shown in the following screenshot:


```
"resources": [
  {
    "name": "linkedTemplate",
    "type": "Microsoft.Resources/deployments",
    "apiVersion": "2018-05-01",
    "properties": {⋯
    }
  },
  {⋯
  },
  {⋯
  },
  {⋯
  },
  {
    "type": "Microsoft.Compute/virtualMachines",
    "name": "[variables('vmName')]",
    "location": "[parameters('location')]",
    "apiVersion": "2018-10-01",
    "dependsOn": [
      "linkedTemplate",
      "[resourceId('Microsoft.Network/networkInterfaces/', variables('nicName'))]"
    ],
    "properties": {
      "hardwareProfile": {⋯
      },
      "osProfile": {⋯
      },
      "storageProfile": {⋯
      },
      "networkProfile": {⋯
      },
      "diagnosticsProfile": {
        "bootDiagnostics": {
          "enabled": true,
          "storageUri": "[reference('linkedTemplate').outputs.storageUri.value]"
        }
      }
    }
  }
],
```

linkedTemplate is the name of the deployments resource.

3. Update **properties/diagnosticsProfile/bootDiagnostics/storageUri** as shown in the previous screenshot.

4. Save the revised template.

Deploy the template

Refer to the Deploy the template section for the deployment procedure. Use the same resource group name as the storage account for storing the linked template. It makes it easier to clean up resources in the next section. To increase security, use a generated password for the virtual machine administrator account. See Prerequisites.

Clean up resources

When the Azure resources are no longer needed, clean up the resources you deployed by deleting the resource group.

1. From the Azure portal, select **Resource group** from the left menu.
2. Enter the resource group name in the **Filter by name** field.
3. Select the resource group name. You shall see a total of six resources in the resource group.
4. Select **Delete resource group** from the top menu.

Additional practice

To improve the project, make the following additional changes to the completed project:

1. Modify the main template (azuredeploy.json) so that it takes the linked template URI value via a parameter.
2. Instead of generating a SAS token when you upload the linked template, generate the token when you deploy the main template. For more information, see Provide SAS token during deployment.

Next steps

In this tutorial, you modularized a template into a main template and a linked template. To learn how to use virtual machine extensions to perform post deployment tasks, see:

Deploy virtual machine extensions

Tutorial: Deploy virtual machine extensions with Azure Resource Manager templates

3/15/2019 • 3 minutes to read • Edit Online

Learn how to use Azure virtual machine extensions to perform post-deployment configuration and automation tasks on Azure VMs. Many different VM extensions are available for use with Azure VMs. In this tutorial, you deploy a Custom Script extension from an Azure Resource Manager template to run a PowerShell script on a Windows VM. The script installs Web Server on the VM.

This tutorial covers the following tasks:

- Prepare a PowerShell script
- Open a quickstart template
- Edit the template
- Deploy the template
- Verify the deployment

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

To complete this article, you need:

- Visual Studio Code with the Resource Manager Tools extension. See Install the extension.
- To increase security, use a generated password for the virtual machine administrator account. Here is a sample for generating a password:

  ```
  openssl rand -base64 32
  ```

  Azure Key Vault is designed to safeguard cryptographic keys and other secrets. For more information, see Tutorial: Integrate Azure Key Vault in Resource Manager Template deployment. We also recommend that you update your password every three months.

Prepare a PowerShell script

A PowerShell script with the following content is shared from an Azure storage account with public access:

```
Install-WindowsFeature -name Web-Server -IncludeManagementTools
```

If you choose to publish the file to your own location, you must update the `fileUri` element in the template later in the tutorial.

Open a quickstart template

Azure Quickstart Templates is a repository for Resource Manager templates. Instead of creating a template from scratch, you can find a sample template and customize it. The template used in this tutorial is called Deploy a simple Windows VM.

1. In Visual Studio Code, select **File** > **Open File**.
2. In the **File name** box, paste the following URL: https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-vm-simple-windows/azuredeploy.json
3. To open the file, select **Open**.
   The template defines five resources:
   - **Microsoft.Storage/storageAccounts**. See the template reference.
   - **Microsoft.Network/publicIPAddresses**. See the template reference.
   - **Microsoft.Network/virtualNetworks**. See the template reference.
   - **Microsoft.Network/networkInterfaces**. See the template reference.
   - **Microsoft.Compute/virtualMachines**. See the template reference.

     It's helpful to get some basic understanding of the template before you customize it.
4. Save a copy of the file to your local computer with the name *azuredeploy.json* by selecting **File** > **Save As**.

Edit the template

Add a virtual machine extension resource to the existing template with the following content:

```
{
    "apiVersion": "2018-06-01",
    "type": "Microsoft.Compute/virtualMachines/extensions",
    "name": "[concat(variables('vmName'),'/', 'InstallWebServer')]",
    "location": "[parameters('location')]",
    "dependsOn": [
        "[concat('Microsoft.Compute/virtualMachines/',variables('vmName'))]"
    ],
    "properties": {
        "publisher": "Microsoft.Compute",
        "type": "CustomScriptExtension",
        "typeHandlerVersion": "1.7",
        "autoUpgradeMinorVersion":true,
        "settings": {
            "fileUris": [
                "https://armtutorials.blob.core.windows.net/usescriptextensions/installWebServer.ps1"
            ],
            "commandToExecute": "powershell.exe -ExecutionPolicy Unrestricted -File installWebServer.ps1"
        }
    }
}
```

For more information about this resource definition, see the extension reference. The following are some important elements:

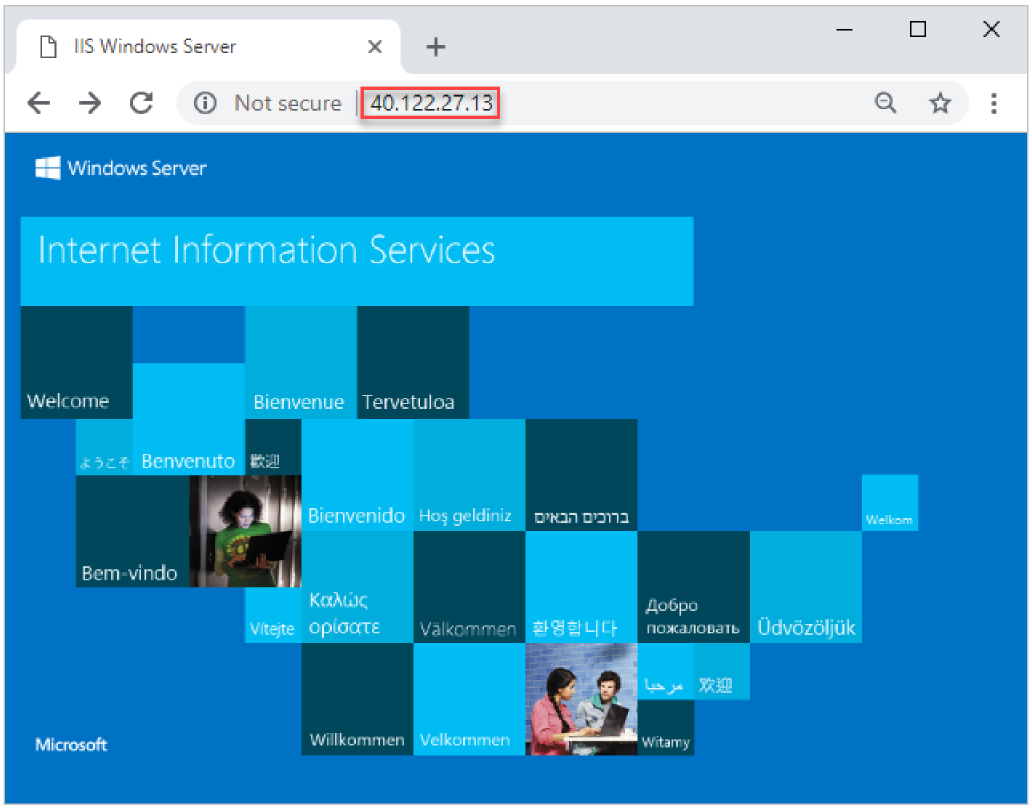
- **name**: Because the extension resource is a child resource of the virtual machine object, the name must have the virtual machine name prefix. See Child resources.
- **dependsOn**: Create the extension resource after you've created the virtual machine.
- **fileUris**: The locations where the script files are stored. If you choose not to use the provided location, you need to update the values.
- **commandToExecute**: This command invokes the script.

Deploy the template

For the deployment procedure, see the "Deploy the template" section of Tutorial: Create Azure Resource Manager templates with dependent resources. We recommended that you use a generated password for the virtual machine administrator account. See this article's Prerequisites section.

Verify the deployment

1. In the Azure portal, select the VM.
2. In the VM overview, copy the IP address by selecting **Click to copy**, and then paste it in a browser tab. The default Internet Information Services (IIS) welcome page opens:



Clean up resources

When you no longer need the Azure resources you deployed, clean them up by deleting the resource group.

1. In the Azure portal, in the left pane, select **Resource group**.
2. In the **Filter by name** box, enter the resource group name.
3. Select the resource group name.
   Six resources are displayed in the resource group.
4. In the top menu, select **Delete resource group**.

Next steps

In this tutorial, you deployed a virtual machine and a virtual machine extension. The extension installed the IIS web server on the virtual machine. To learn how to use the Azure SQL Database extension to import a BACPAC file, see:

Deploy SQL extensions

Tutorial: Import SQL BACPAC files with Azure Resource Manager templates

4/8/2019 • 4 minutes to read • Edit Online

Learn how to use Azure SQL Database extensions to import a BACPAC file with Azure Resource Manager templates. Deployment artifacts are any files, in addition to the main template file that are needed to complete a deployment. The BACPAC file is an artifact. In this tutorial, you create a template to deploy an Azure SQL Server, a SQL Database, and import a BACPAC file. For information about deploying Azure virtual machine extensions using Azure Resource Manager templates, see # Tutorial: Deploy virtual machine extensions with Azure Resource Manager templates.

This tutorial covers the following tasks:

- Prepare a BACPAC file
- Open a Quickstart template
- Edit the template
- Deploy the template
- Verify the deployment

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

To complete this article, you need:

- Visual Studio Code with the Resource Manager Tools extension. See Install the extension .
- To increase security, use a generated password for the SQL Server administrator account. Here is a sample for generating a password:

  ```
  openssl rand -base64 32
  ```

  Azure Key Vault is designed to safeguard cryptographic keys and other secrets. For more information, see Tutorial: Integrate Azure Key Vault in Resource Manager Template deployment. We also recommend you to update your password every three months.

Prepare a BACPAC file

A BACPAC file is shared on an Azure Storage account with the public access. To create your own, see Export an Azure SQL database to a BACPAC file. If you choose to publish the file to your own location, you must update the template later in the tutorial.

Open a Quickstart template

The template used in this tutorial is stored in an Azure Storage account.

1. From Visual Studio Code, select **File**>**Open File**.
2. In **File name**, paste the following URL:

```
https://armtutorials.blob.core.windows.net/createsql/azuredeploy.json
```

3. Select **Open** to open the file.

   There are three resources defined in the template:

   - `Microsoft.Sql/servers`. See the template reference.
   - `Microsoft.SQL/servers/securityAlertPolicies`. See the template reference.
   - `Microsoft.SQL.servers/databases`. See the template reference.

     It is helpful to get some basic understanding of the template before customizing it.

4. Select **File**>**Save As** to save a copy of the file to your local computer with the name **azuredeploy.json**.

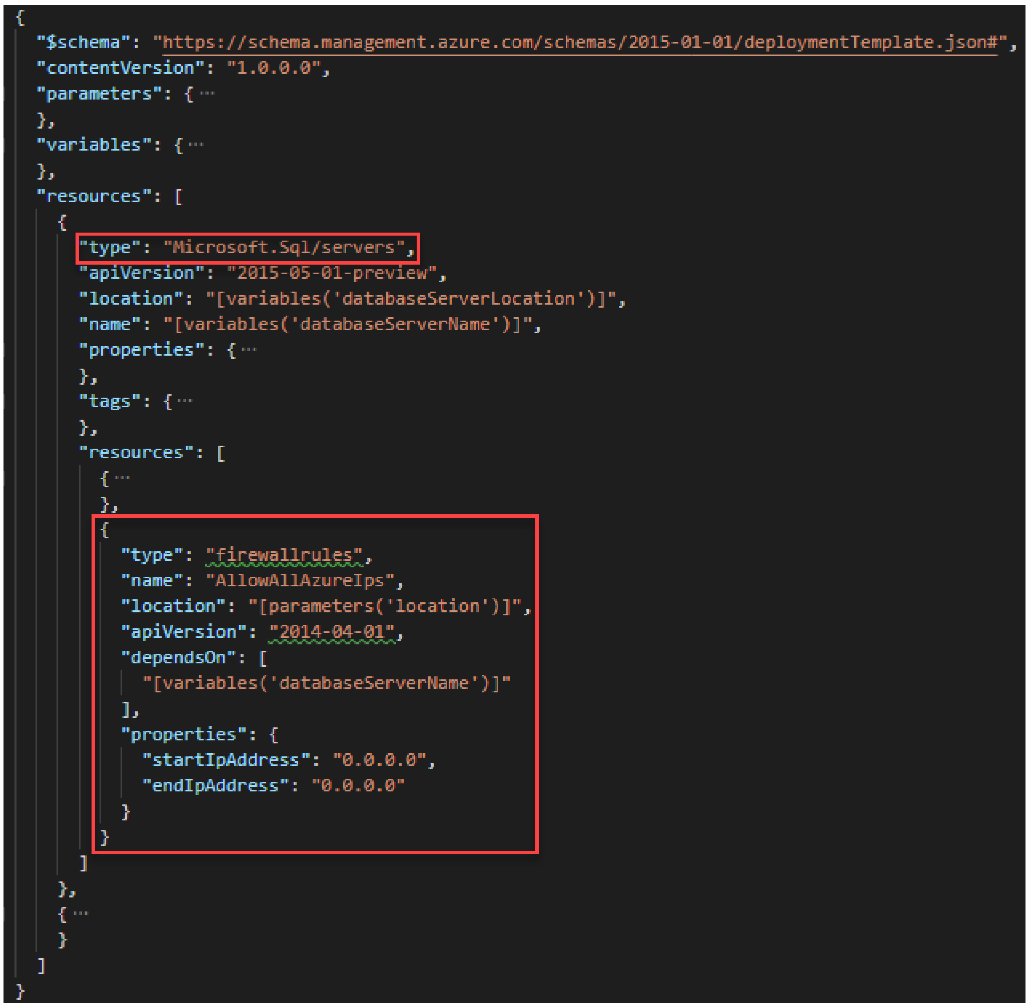
Edit the template

Add two additional resources to the template.

- To allow the SQL database extension to import BACPAC files, you need to allow access to Azure services. Add the following JSON to the SQL server definition:

  ```
  {
      "type": "firewallrules",
      "name": "AllowAllAzureIps",
      "location": "[parameters('location')]",
      "apiVersion": "2015-05-01-preview",
      "dependsOn": [
          "[variables('databaseServerName')]"
      ],
      "properties": {
          "startIpAddress": "0.0.0.0",
          "endIpAddress": "0.0.0.0"
      }
  }
  ```

  The template shall look like:

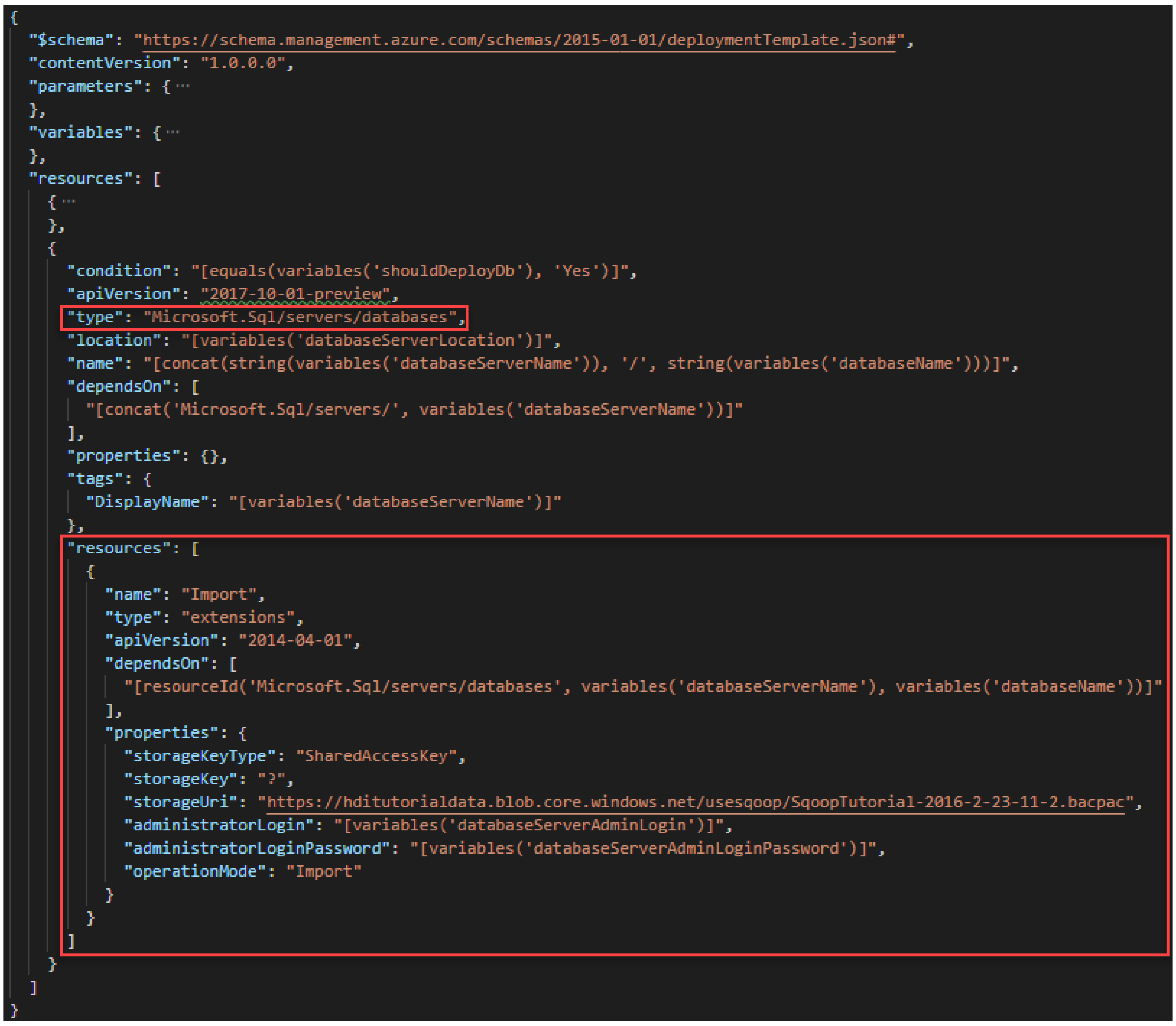


```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {···
  },
  "variables": {···
  },
  "resources": [
    {
      "type": "Microsoft.Sql/servers",
      "apiVersion": "2015-05-01-preview",
      "location": "[variables('databaseServerLocation')]",
      "name": "[variables('databaseServerName')]",
      "properties": {···
      },
      "tags": {···
      },
      "resources": [
        {···
        },
        {
          "type": "firewallrules",
          "name": "AllowAllAzureIps",
          "location": "[parameters('location')]",
          "apiVersion": "2014-04-01",
          "dependsOn": [
            "[variables('databaseServerName')]"
          ],
          "properties": {
            "startIpAddress": "0.0.0.0",
            "endIpAddress": "0.0.0.0"
          }
        }
      ]
    },
    {···
    }
  ]
}
```

- Add a SQL Database extension resource to the database definition with the following JSON:

```
"resources": [
    {
        "name": "Import",
        "type": "extensions",
        "apiVersion": "2014-04-01",
        "dependsOn": [
            "[resourceId('Microsoft.Sql/servers/databases', variables('databaseServerName'),
variables('databaseName'))]"
        ],
        "properties": {
            "storageKeyType": "SharedAccessKey",
            "storageKey": "?",
            "storageUri":
"https://armtutorials.blob.core.windows.net/sqlextensionbacpac/SQLDatabaseExtension.bacpac",
            "administratorLogin": "[variables('databaseServerAdminLogin')]",
            "administratorLoginPassword": "[variables('databaseServerAdminLoginPassword')]",
            "operationMode": "Import",
        }
    }
]
```

The template shall look like:



```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {⋯
  },
  "variables": {⋯
  },
  "resources": [
    {⋯
    },
    {
      "condition": "[equals(variables('shouldDeployDb'), 'Yes')]",
      "apiVersion": "2017-10-01-preview",
      "type": "Microsoft.Sql/servers/databases",
      "location": "[variables('databaseServerLocation')]",
      "name": "[concat(string(variables('databaseServerName')), '/', string(variables('databaseName')))]",
      "dependsOn": [
        "[concat('Microsoft.Sql/servers/', variables('databaseServerName'))]"
      ],
      "properties": {},
      "tags": {
        "DisplayName": "[variables('databaseServerName')]"
      },
      "resources": [
        {
          "name": "Import",
          "type": "extensions",
          "apiVersion": "2014-04-01",
          "dependsOn": [
            "[resourceId('Microsoft.Sql/servers/databases', variables('databaseServerName'), variables('databaseName'))]"
          ],
          "properties": {
            "storageKeyType": "SharedAccessKey",
            "storageKey": "?",
            "storageUri": "https://hditutorialdata.blob.core.windows.net/usesqoop/SqoopTutorial-2016-2-23-11-2.bacpac",
            "administratorLogin": "[variables('databaseServerAdminLogin')]",
            "administratorLoginPassword": "[variables('databaseServerAdminLoginPassword')]",
            "operationMode": "Import"
          }
        }
      ]
    }
  ]
}
```

To understand the resource definition, see the SQL Database extension reference. The following are some important elements:

- **dependsOn**: The extension resource must be created after the SQL database has been created.
- **storageKeyType**: The type of the storage key to use. The value can be either `StorageAccessKey` or `SharedAccessKey`. Because the provided BACPAC file is shared on an Azure Storage account with public access, `SharedAccessKey' is used here.
- **storageKey**: The storage key to use. If storage key type is SharedAccessKey, it must be preceded with a "?."
- **storageUri**: The storage uri to use. If you choose not to use the BACPAC file provided, you need to update the values.
- **administratorLoginPassword**: The password of the SQL administrator. Use a generated password. See Prerequisites.

Deploy the template

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

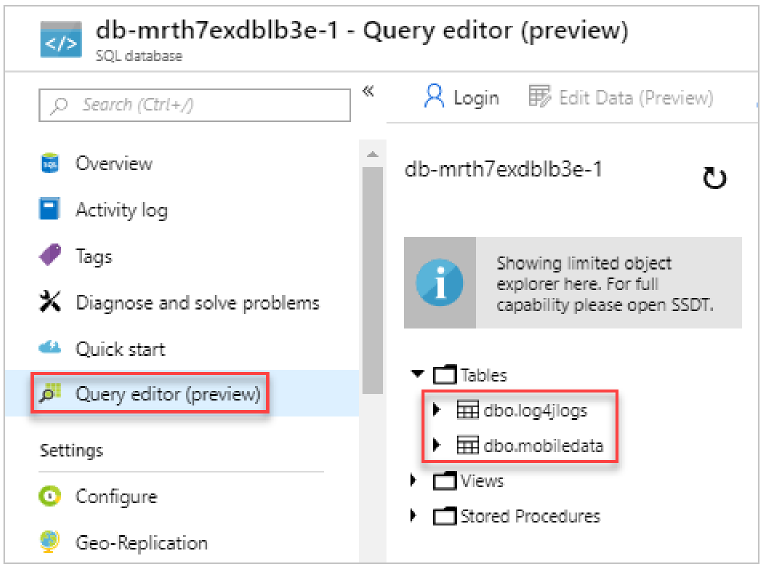
Refer to the Deploy the template section for the deployment procedure. Use the following PowerShell deployment script instead:

```
$resourceGroupName = Read-Host -Prompt "Enter the Resource Group name"
$location = Read-Host -Prompt "Enter the location (i.e. centralus)"
$adminUsername = Read-Host -Prompt "Enter the SQL admin username"
$adminPassword = Read-Host -Prompt "Enter the admin password" -AsSecureString

New-AzResourceGroup -Name $resourceGroupName -Location $location
New-AzResourceGroupDeployment `
    -ResourceGroupName $resourceGroupName `
    -adminUser $adminUsername `
    -adminPassword $adminPassword `
    -TemplateFile "$HOME/azuredeploy.json"
```

Use a generated password. See Prerequisites.

Verify the deployment

In the portal, select the SQL database from the newly deployed resource group. Select **Query editor (preview)**, and then enter the administrator credentials. You shall see two tables imported into the database:



Clean up resources

When the Azure resources are no longer needed, clean up the resources you deployed by deleting the resource group.

1. From the Azure portal, select **Resource group** from the left menu.
2. Enter the resource group name in the **Filter by name** field.
3. Select the resource group name. You shall see a total of six resources in the resource group.
4. Select **Delete resource group** from the top menu.

Next steps

In this tutorial, you deployed a SQL Server, a SQL Database, and imported a BACPAC file. The BACPAC file is stored in an Azure storage account. Anybody with the URL can access the file. To learn how to secure the BACPAC file (artifact), see

Secure the artifacts

Tutorial: Secure artifacts in Azure Resource Manager template deployments

6/10/2019 • 7 minutes to read • Edit Online

Learn how to secure the artifacts used in your Azure Resource Manager templates using Azure Storage account with shared access signatures (SAS). Deployment artifacts are any files, in addition to the main template file, that are needed to complete a deployment. For example, in Tutorial: Import SQL BACPAC files with Azure Resource Manager templates, the main template creates an Azure SQL Database; it also calls a BACPAC file to create tables and insert data. The BACPAC file is an artifact. The artifact is stored in an Azure storage account with public access. In this tutorial, you use SAS to grant limited access to the BACPAC file in your own Azure Storage account. For more information about SAS, see Using shared access signatures (SAS).

To learn how to secure linked template, see Tutorial: Create linked Azure Resource Manager templates.

This tutorial covers the following tasks:

- Prepare a BACPAC file
- Open an existing template
- Edit the template
- Deploy the template
- Verify the deployment

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

To complete this article, you need:

- Visual Studio Code with the Resource Manager Tools extension. See Install the extension .
- Review Tutorial: Import SQL BACPAC files with Azure Resource Manager templates. The template used in this tutorial is the one developed in that tutorial. A download link of the completed template is provided in this article.
- To increase security, use a generated password for the SQL Server administrator account. Here is a sample for generating a password:

  ```
  openssl rand -base64 32
  ```

  Azure Key Vault is designed to safeguard cryptographic keys and other secrets. For more information, see Tutorial: Integrate Azure Key Vault in Resource Manager Template deployment. We also recommend you to update your password every three months.

Prepare a BACPAC file

In this section, you prepare the BACPAC file so the file is accessible securely when you deploy the Resource Manager template. There are five procedures in this section:

- Download the BACPAC file.
- Create an Azure Storage account.

- Create a Storage account Blob container.
- Upload the BACPAC file to the container.
- Retrieve the SAS token of the BACPAC file.

To automate these steps using a PowerShell script, see the script from Upload the linked template.

Download the BACPAC file

Download the BACPAC file, and save the file to your local computer with the same name, **SQLDatabaseExtension.bacpac**.

Create a storage account

1. Select the following image to open a Resource Manager template in the Azure portal.

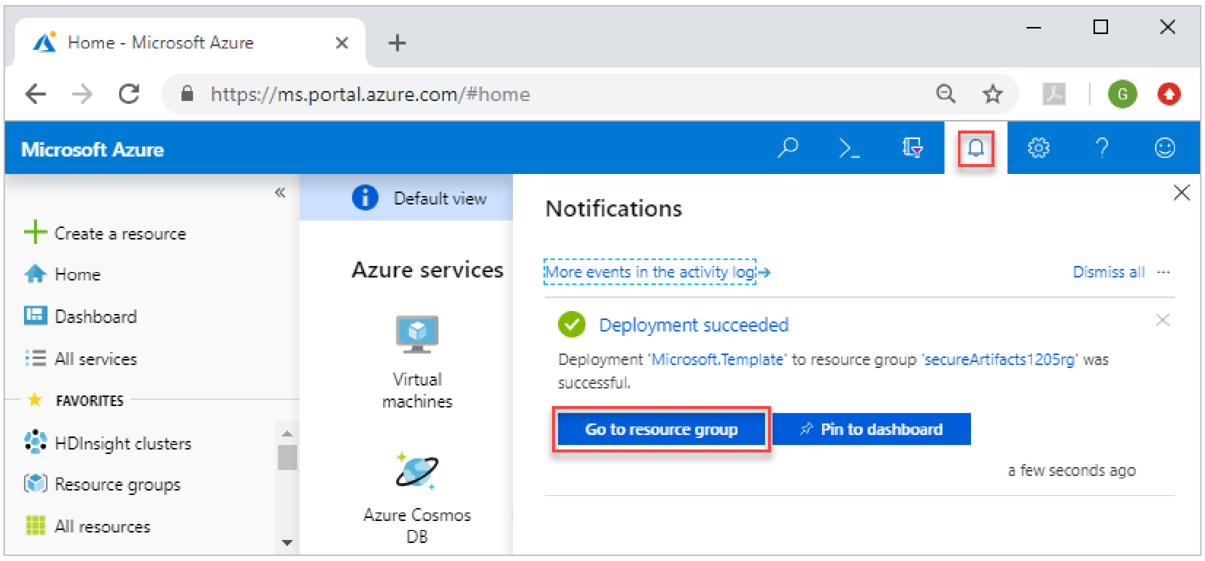
   Deploy to Azure

2. Enter the following properties:

   - **Subscription**: Select your Azure subscription.
   - **Resource Group**: Select **Create new** and give it a name. A resource group is a container for Azure resources for the management purpose. In this tutorial, you can use the same resource group for the storage account and the Azure SQL Database. Make a note of this resource group name, you need it when you create the Azure SQL Database later in the tutorials.
   - **Location**: Select a region. For example, **Central US**.
   - **Storage Account Type**: use the default value, which is **Standard_LRS**.
   - **Location**: Use the default value, which is **[resourceGroup().location]**. That means you use the resource group location for the storage account.
   - **I agree to the terms and conditions started above**: (selected)

3. Select **Purchase**.

4. Select the notification icon (the bell icon) on the upper right corner of the portal to see the deployment status.



Deployment succeeded

Deployment 'Microsoft.Template' to resource group 'secureArtifacts1205rg' was successful.

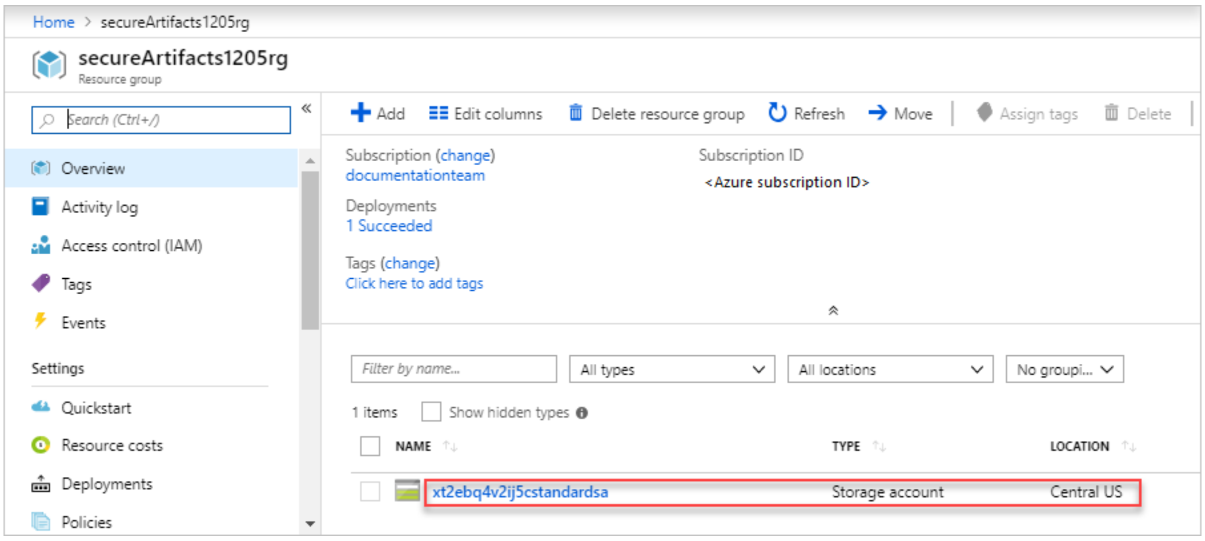
5. After the storage account is deployed successfully, select **Go to resource group** from the notification pane to open the resource group.

Create a Blob container

A Blob container is needed before you can upload any files.

1. Select the storage account to open it. You shall see only one storage account listed in the resource group.

Your storage account name is different from the one shown in the following screenshot.


Home > secureArtifacts1205rg

secureArtifacts1205rg
Resource group

Search (Ctrl+/)

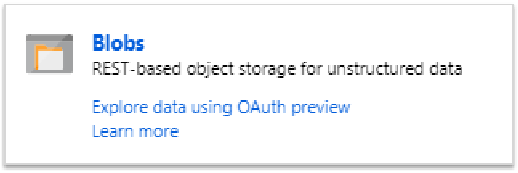
Subscription (change): documentationteam
Subscription ID: <Azure subscription ID>
Deployments: 1 Succeeded
Tags (change): Click here to add tags

Filter by name... | All types | All locations | No groupi...

1 items Show hidden types

NAME	TYPE	LOCATION
xt2ebq4v2ij5cstandardsa	Storage account	Central US

2. Select the **Blobs** tile.


Blobs
REST-based object storage for unstructured data
Explore data using OAuth preview
Learn more

3. Select **+ Container** from the top to create a new container.

4. Enter the following values:

   - **Name**: enter **sqlbacpac**.
   - **Public access level**: use the default value, **Private (no anonymous access)**.

5. Select **OK**.

6. Select **sqlbacpac** to open the newly created container.

Upload the BACPAC file to the container

1. Select **Upload**.

2. Enter the following values:

   - **Files**: Following the instructions to select the BACPAC file you downloaded earlier. The default name is **SQLDatabaseExtension.bacpac**.
   - **Authentication type**: Select **SAS**. *SAS* is the default value.

3. Select **Upload**. Once the file is uploaded successfully, the file name shall be listed in the container.

Generate a SAS token

1. Right-click **SQLDatabaseExtension.bacpac** from the container, and then select **Generate SAS**.

2. Enter the following values:

   - **Permission**: Use the default, **Read**.
   - **Start and expiry date/time**: The default value gives you eight hours to use the SAS token. If you need more time to complete this tutorial, update **Expiry**.
   - **Allowed IP addresses**: Leave this field blank.
   - **Allowed protocols**: use the default value: **HTTPS**.
   - **Signing key**: use the default value: **Key 1**.

3. Select **Generate blob SAS token and URL**.

4. Make a copy of **Blob SAS URL**. In the middle of the URL is the file name **SQLDatabaseExtension.bacpac**. The file name divides the URL into three parts:

   - **Artifact location**: https://xxxxxxxxxxxxxx.blob.core.windows.net/sqlbacpac/. Make sure the location ends with a "/".
   - **BACPAC file name**: SQLDatabaseExtension.bacpac.
   - **Artifact location SAS token**: Make sure the token precedes with a "?."

     You need these three values in Deploy the template.

Open an existing template

In this session, you modify the template you created in Tutorial: Import SQL BACPAC files with Azure Resource Manager templates to call the BACPAC file with a SAS token. The template developed in the SQL extension tutorial is shared at https://armtutorials.blob.core.windows.net/sqlextensionbacpac/azuredeploy.json.

1. From Visual Studio Code, select **File**>**Open File**.
2. In **File name**, paste the following URL:

   ```
   https://armtutorials.blob.core.windows.net/sqlextensionbacpac/azuredeploy.json
   ```

3. Select **Open** to open the file.

   There are five resources defined in the template:

   - `Microsoft.Sql/servers`. See the template reference.
   - `Microsoft.SQL/servers/securityAlertPolicies`. See the template reference.
   - `Microsoft.SQL/servers/filewallRules`. See the template reference.
   - `Microsoft.SQL/servers/databases`. See the template reference.
   - `Microsoft.SQL/server/databases/extensions`. See the template reference.

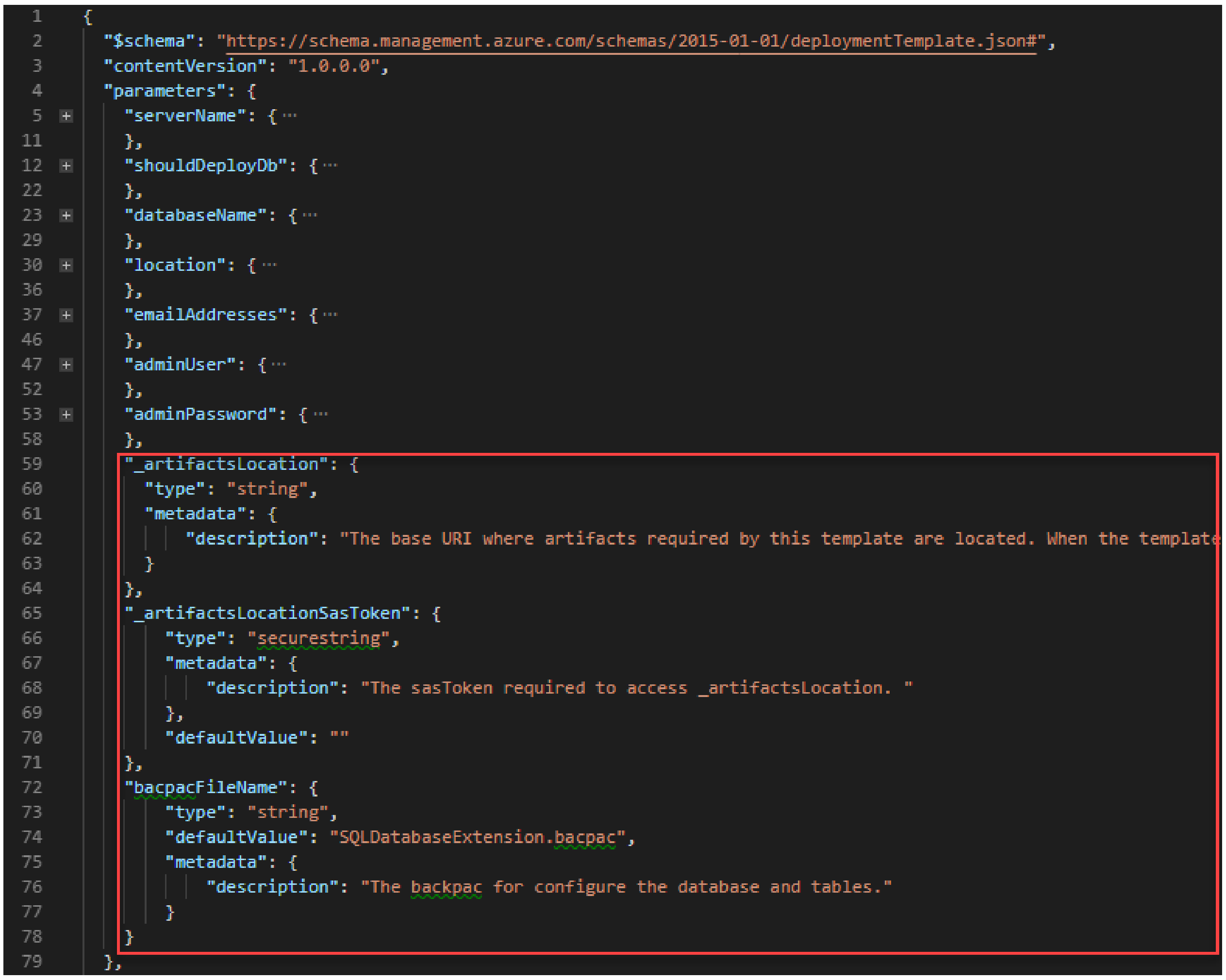
     It is helpful to get some basic understanding of the template before customizing it.

4. Select **File**>**Save As** to save a copy of the file to your local computer with the name **azuredeploy.json**.

Edit the template

Add the following additional parameters:

```
"_artifactsLocation": {
    "type": "string",
    "metadata": {
        "description": "The base URI where artifacts required by this template are located."
    }
},
"_artifactsLocationSasToken": {
    "type": "securestring",
    "metadata": {
        "description": "The sasToken required to access _artifactsLocation."
    },
    "defaultValue": ""
},
"bacpacFileName": {
    "type": "string",
    "defaultValue": "SQLDatabaseExtension.bacpac",
    "metadata": {
        "description": "The bacpac for configure the database and tables."
    }
}
```



```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "serverName": { ⋯
    },
    "shouldDeployDb": { ⋯
    },
    "databaseName": { ⋯
    },
    "location": { ⋯
    },
    "emailAddresses": { ⋯
    },
    "adminUser": { ⋯
    },
    "adminPassword": { ⋯
    },
    "_artifactsLocation": {
      "type": "string",
      "metadata": {
          "description": "The base URI where artifacts required by this template are located. When the template
      }
    },
    "_artifactsLocationSasToken": {
        "type": "securestring",
        "metadata": {
            "description": "The sasToken required to access _artifactsLocation. "
        },
        "defaultValue": ""
    },
    "bacpacFileName": {
        "type": "string",
        "defaultValue": "SQLDatabaseExtension.bacpac",
        "metadata": {
            "description": "The backpac for configure the database and tables."
        }
    }
  },
```

Update the value of the following two elements:

```
"storageKey": "[parameters('_artifactsLocationSasToken')]",
"storageUri": "[uri(parameters('_artifactsLocation'), parameters('bacpacFileName'))]",
```

Deploy the template

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

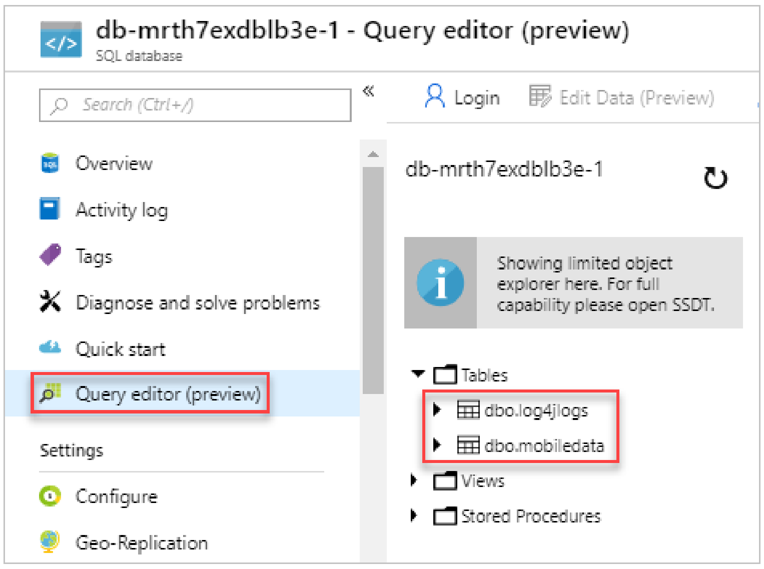
Refer to the Deploy the template section for the deployment procedure. Use the following PowerShell deployment script instead:

```
$resourceGroupName = Read-Host -Prompt "Enter the Resource Group name"
$location = Read-Host -Prompt "Enter the location (i.e. centralus)"
$adminUsername = Read-Host -Prompt "Enter the virtual machine admin username"
$adminPassword = Read-Host -Prompt "Enter the admin password" -AsSecureString
$artifactsLocation = Read-Host -Prompt "Enter the artifacts location"
$artifactsLocationSasToken = Read-Host -Prompt "Enter the artifacts location SAS token" -AsSecureString
$bacpacFileName = Read-Host -Prompt "Enter the BACPAC file name"

New-AzResourceGroup -Name $resourceGroupName -Location $location
New-AzResourceGroupDeployment `
    -ResourceGroupName $resourceGroupName `
    -adminUser $adminUsername `
    -adminPassword $adminPassword `
    -_artifactsLocation $artifactsLocation `
    -_artifactsLocationSasToken $artifactsLocationSasToken `
    -bacpacFileName $bacpacFileName `
    -TemplateFile "$HOME/azuredeploy.json"
```

Use a generated password. See Prerequisites. For the values of _artifactsLocation, _artifactsLocationSasToken and bacpacFileName, see Generate a SAS token.

Verify the deployment

In the portal, select the SQL database from the newly deployed resource group. Select **Query editor (preview)**, and then enter the administrator credentials. You shall see two tables imported into the database:



Clean up resources

When the Azure resources are no longer needed, clean up the resources you deployed by deleting the resource group.

1. From the Azure portal, select **Resource group** from the left menu.
2. Enter the resource group name in the **Filter by name** field.
3. Select the resource group name. You shall see a total of six resources in the resource group.
4. Select **Delete resource group** from the top menu.

Next steps

In this tutorial, you deployed a SQL Server, a SQL Database, and imported a BACPAC file using SAS token. To learn how to create an Azure Pipeline to continuously develop and deploy Resource Manager templates, see

Continuous integration with Azure Pipeline

Tutorial: Continuous integration of Azure Resource Manager templates with Azure Pipelines

6/12/2019 • 8 minutes to read • Edit Online

Learn how to use Azure Pipelines to continuously build and deploy Azure Resource Manager template projects.

Azure DevOps provides developer services to support teams to plan work, collaborate on code development, and build and deploy applications. Developers can work in the cloud using Azure DevOps Services. Azure DevOps provides an integrated set of features that you can access through your web browser or IDE client. Azure Pipeline is one of these features. Azure Pipelines is a fully featured continuous integration (CI) and continuous delivery (CD) service. It works with your preferred Git provider and can deploy to most major cloud services. Then you can automate the build, testing, and deployment of your code to Microsoft Azure, Google Cloud Platform, or Amazon Web Services.

This tutorial is designed for Azure Resource Manager template developers who are new Azure DevOps Services and Azure Pipelines. If you are already familiar with GitHub and DevOps, you can skip to Create a pipeline.

NOTE

Pick a project name. When you go through the tutorial, replace any of the **AzureRmPipeline** with your project name.

This tutorial covers the following tasks:

- Prepare a GitHub repository
- Create an Azure DevOps project
- Create an Azure pipeline
- Verify the pipeline deployment
- Update the template and redeploy
- Clean up resources

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

To complete this article, you need:

- **A GitHub account**, where you use it to create a repository for your templates. If you don’t have one, you can create one for free. For more information about using GitHub repositories, see Build GitHub repositories.
- **Install Git**. This tutorial instruction uses *Git Bash* or *Git Shell*. For instructions, see Install Git.
- **An Azure DevOps organization**. If you don't have one, you can create one for free. See Create an organization or project collection.
- **Visual Studio Code with the Resource Manager Tools extension**. See Install the extension .

Prepare a GitHub repository

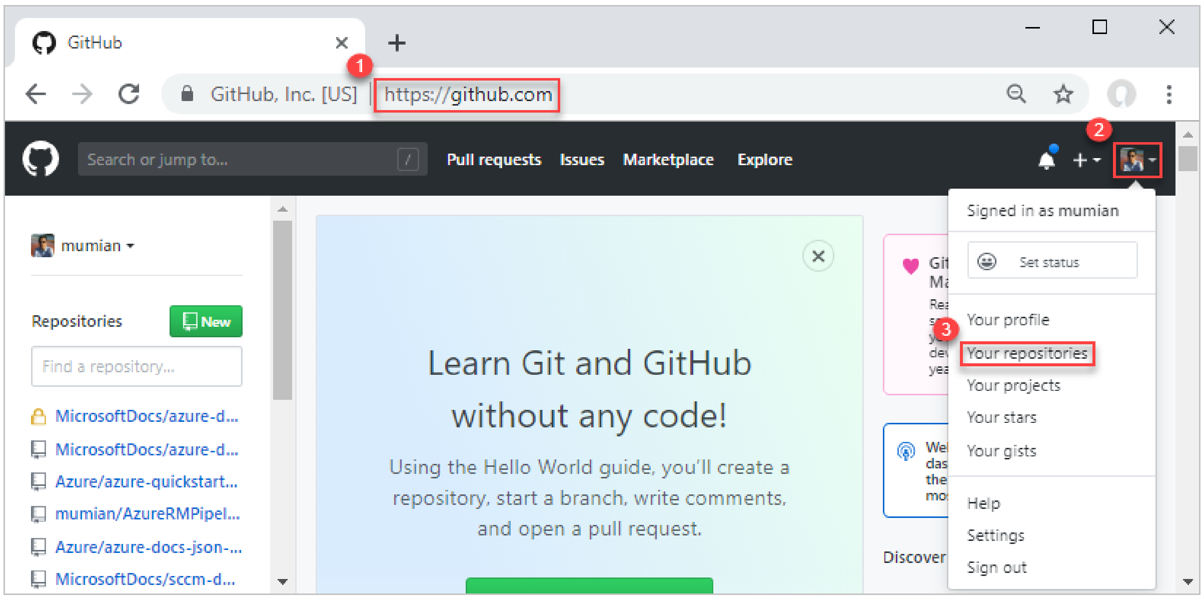
GitHub is used to store your project source code including Resource Manager templates. For other supported repositories, see repositories supported by Azure DevOps.

Create a GitHub repository

If you don't have a GitHub account, see Prerequisites.

1. Sign in to GitHub.

2. Select your account image on the upper right corner, and then select **Your repositories**.



3. Select **New**, a green button.

4. In **Repository name**, enter a repository name. For example, **AzureRmPipeline-repo**. Remember to replace any of **AzureRmPipeline** with your project name. You can select either **Public** or **private** for going through this tutorial. And then select **Create repository**.

5. Write down the URL. The repository URL is the following format:

```
https://github.com/[YourAccountName]/[YourRepositoryName]
```

This repository is referred to as a *remote repository*. Each of the developers of the same project can clone his/her own *local repository*, and merge the changes to the remote repository.

Clone the remote repository

1. Open Git Shell or Git Bash. See Prerequisites.

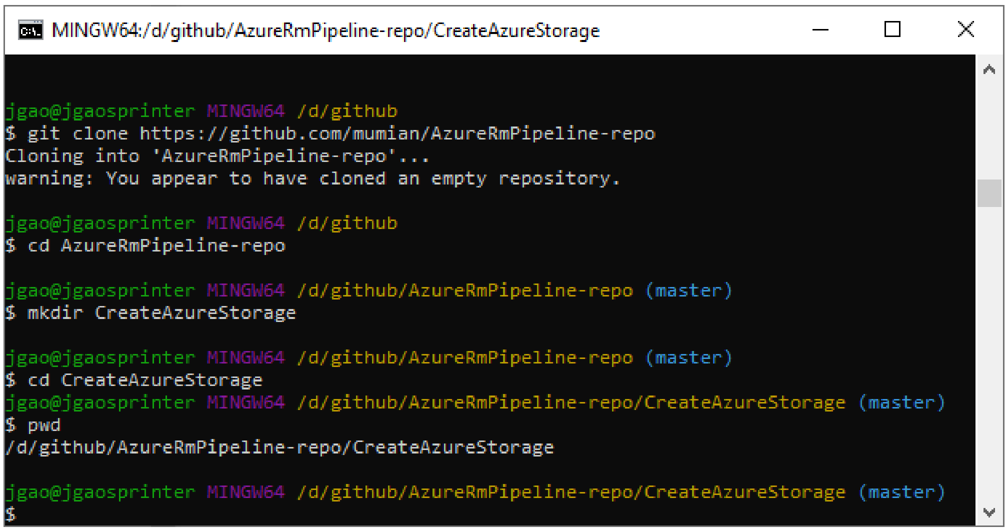
2. Verify your current folder is **github**.

3. Run the following command:

```
git clone https://github.com/[YourAccountName]/[YourGitHubRepositoryName]
cd [YourGitHubRepositoryName]
mkdir CreateAzureStorage
cd CreateAzureStorage
pwd
```

Replace **[YourAccountName]** with your GitHub account name, and replace **[YourGitHubRepositoryName]** with your repository name you created in the previous procedure.

The following screenshots shows an example.


MINGW64:/d/github/AzureRmPipeline-repo/CreateAzureStorage

```
jgao@jgaosprinter MINGW64 /d/github
$ git clone https://github.com/mumian/AzureRmPipeline-repo
Cloning into 'AzureRmPipeline-repo'...
warning: You appear to have cloned an empty repository.

jgao@jgaosprinter MINGW64 /d/github
$ cd AzureRmPipeline-repo

jgao@jgaosprinter MINGW64 /d/github/AzureRmPipeline-repo (master)
$ mkdir CreateAzureStorage

jgao@jgaosprinter MINGW64 /d/github/AzureRmPipeline-repo (master)
$ cd CreateAzureStorage
jgao@jgaosprinter MINGW64 /d/github/AzureRmPipeline-repo/CreateAzureStorage (master)
$ pwd
/d/github/AzureRmPipeline-repo/CreateAzureStorage

jgao@jgaosprinter MINGW64 /d/github/AzureRmPipeline-repo/CreateAzureStorage (master)
$
```

The **CreateAzureStorage** folder is the folder where the template is stored. The **pwd** command shows the folder path. The path is where you save the template to in the following procedure.

Download a Quickstart template

Instead of creating a template, you can download a Quickstart template. This template creates an Azure Storage account.

1. Open Visual Studio code. See Prerequisites.

2. Open the template with the following URL:

```
https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-storage-account-create/azuredeploy.json
```

3. Save the file as **azuredeploy.json** to the **CreateAzureStorage** folder. Both the folder name and the file name are used as they are in the pipeline. If you change these names, you must update the names used in the pipeline.

Push the template to the remote repository

The azuredeploy.json has been added to the local repository. Next, you upload the template to the remote repository.

1. Open *Git Shell* or *Git Bash*, if it is not opened.

2. Change directory to the CreateAzureStorage folder in your local repository.

3. Verify the **azuredeploy.json** file is in the folder.

4. Run the following command:

```
git add .
git commit -m “Add a new create storage account template.”
git push origin master
```

   You might get a warning about LF. You can ignore the warning. **master** is the master branch. You typically create a branch for each update. To simplify the tutorial, you use the master branch directly.

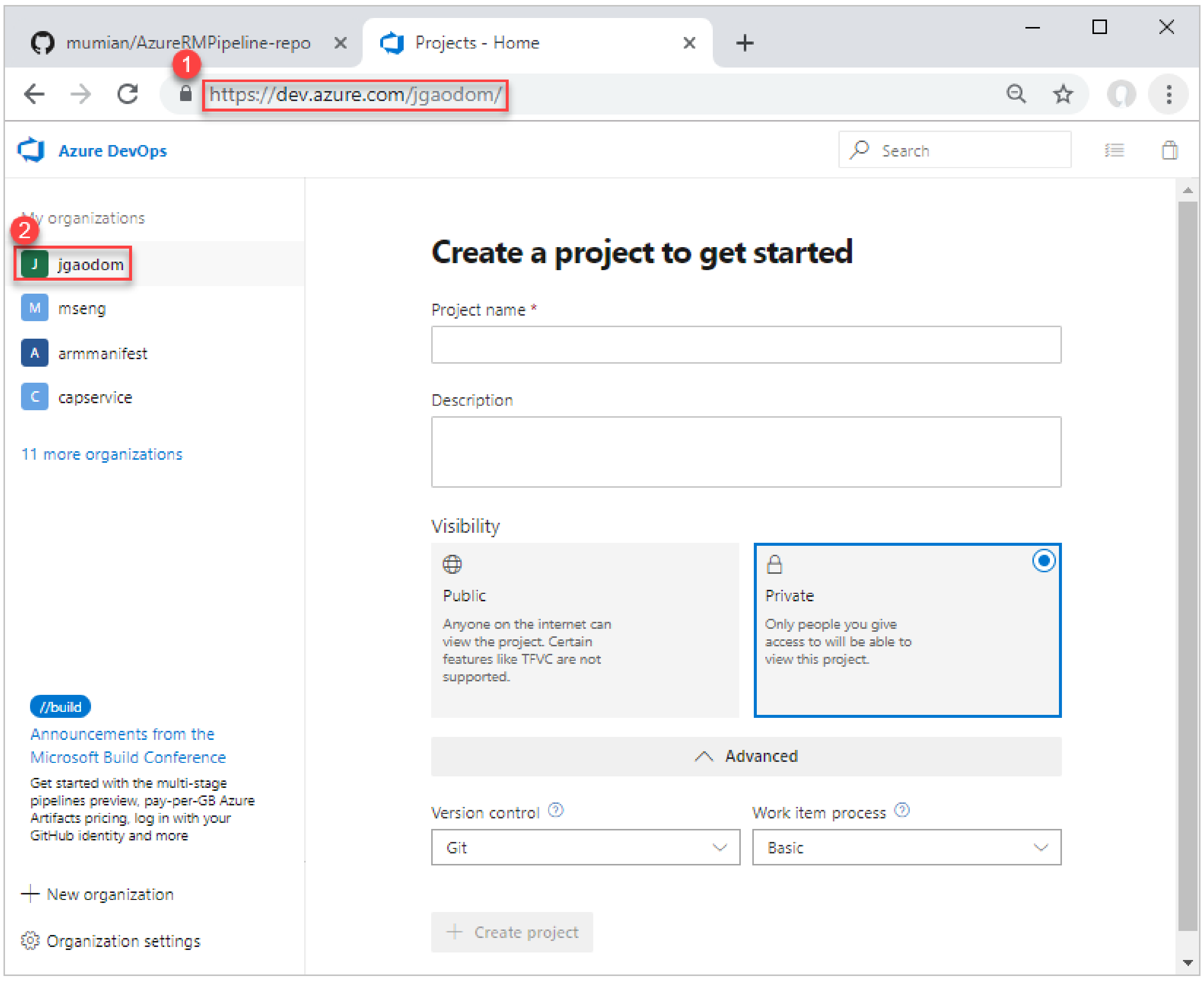
5. Browse to your GitHub repository from a browser. The URL is **https://github.com/[YourAccountName]/[YourGitHubRepository]**. You shall see the **CreateAzureStorage** folder and **Azuredeploy.json** inside the folder.

So far, you have created a GitHub repository, and uploaded a template to the repository.

Create a DevOps project

A DevOps organization is needed before you can proceed to the next procedure. If you don't have one, see Prerequisites.

1. Sign in to Azure DevOps.
2. Select a DevOps organization from the left.



Get started with the multi-stage pipelines preview, pay-per-GB Azure Artifacts pricing, log in with your GitHub identity and more

Create a project to get started

Project name *

Description

Visibility

Public
Anyone on the internet can view the project. Certain features like TFVC are not supported.

Private
Only people you give access to will be able to view this project.

Version control: Git

Work item process: Basic

3. Select **Create project**. If you don't have any projects, the create project page is opened automatically.
4. Enter the following values:
   - **Project name**: enter a project name. You can use the project name you picked at the very beginning of the tutorial.
   - **Version control**: Select **Git**. You might need to expand **Advanced** to see **Version control**.

   Use the default value for the other properties.
5. Select **Create project**.

Create a service connection that is used to deploy projects to Azure.

1. Select **Project settings** from the bottom of the left menu.
2. Select **Service connections** under **Pipelines**.
3. Select **New Service connection**, and then select **AzureResourceManager**.
4. Enter the following values:
   - **Connection name**: enter a connection name. For example, **AzureRmPipeline-conn**. Write down this name, you need the name when you create your pipeline.

- **Scope level**: select **Subscription**.
- **Subscription**: select your subscription.
- **Resource Group**: Leave it blank.
- **Allow all pipelines to use this connection**. (selected)

5. Select **OK**.

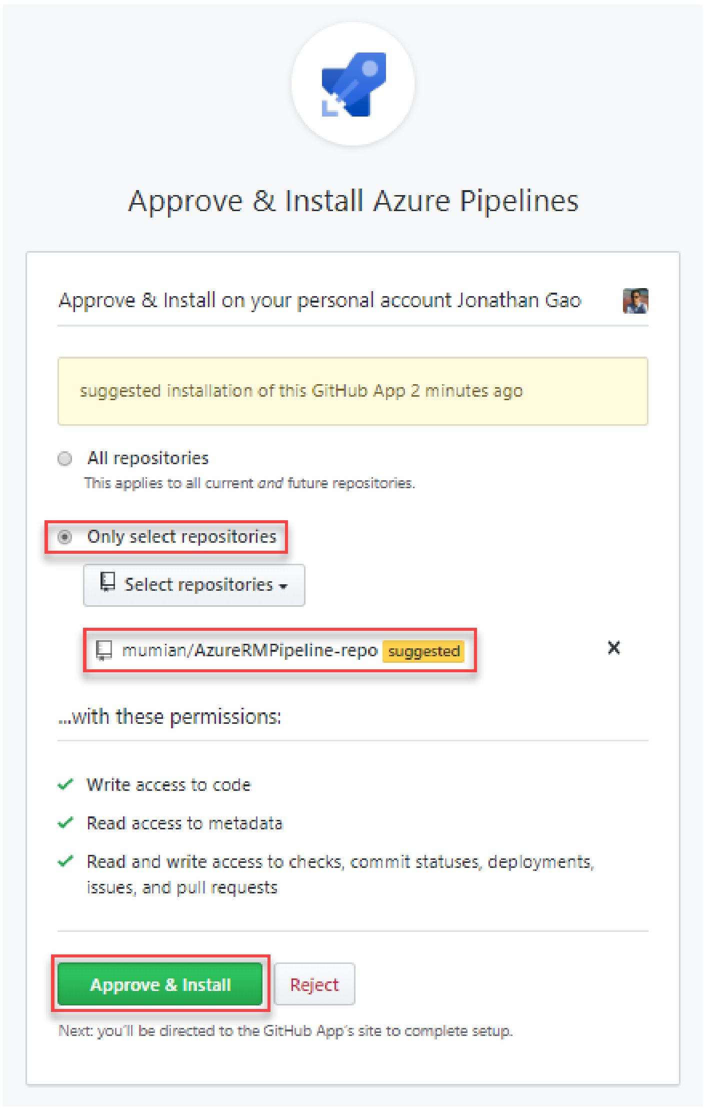
Create a pipeline

Until now, you have completed the following tasks. If you skip the previous sections because you are familiar with GitHub and DevOps, you must complete the tasks before you continue.

- Create a GitHub repository, and save this template to the **CreateAzureStorage** folder in the repository.
- Create a DevOps project, and create an Azure Resource Manager service connection.

To create a pipeline with a step to deploy a template:

1. Select **Pipelines** from the left menu.
2. Select **New pipeline**.
3. From the **Connect** tab, select **GitHub**. If asked, enter your GitHub credentials, and then follow the instructions. If you see the following screen, select **Only select repositories**, and verify your repository is in the list before you select **Approve & Install**.



Approve & Install Azure Pipelines

Approve & Install on your personal account Jonathan Gao

suggested installation of this GitHub App 2 minutes ago

All repositories
This applies to all current *and* future repositories.

Only select repositories

Select repositories ▾

mumian/AzureRMPipeline-repo suggested ×

...with these permissions:

✓ Write access to code
✓ Read access to metadata
✓ Read and write access to checks, commit statuses, deployments, issues, and pull requests

Approve & Install Reject

Next: you'll be directed to the GitHub App's site to complete setup.

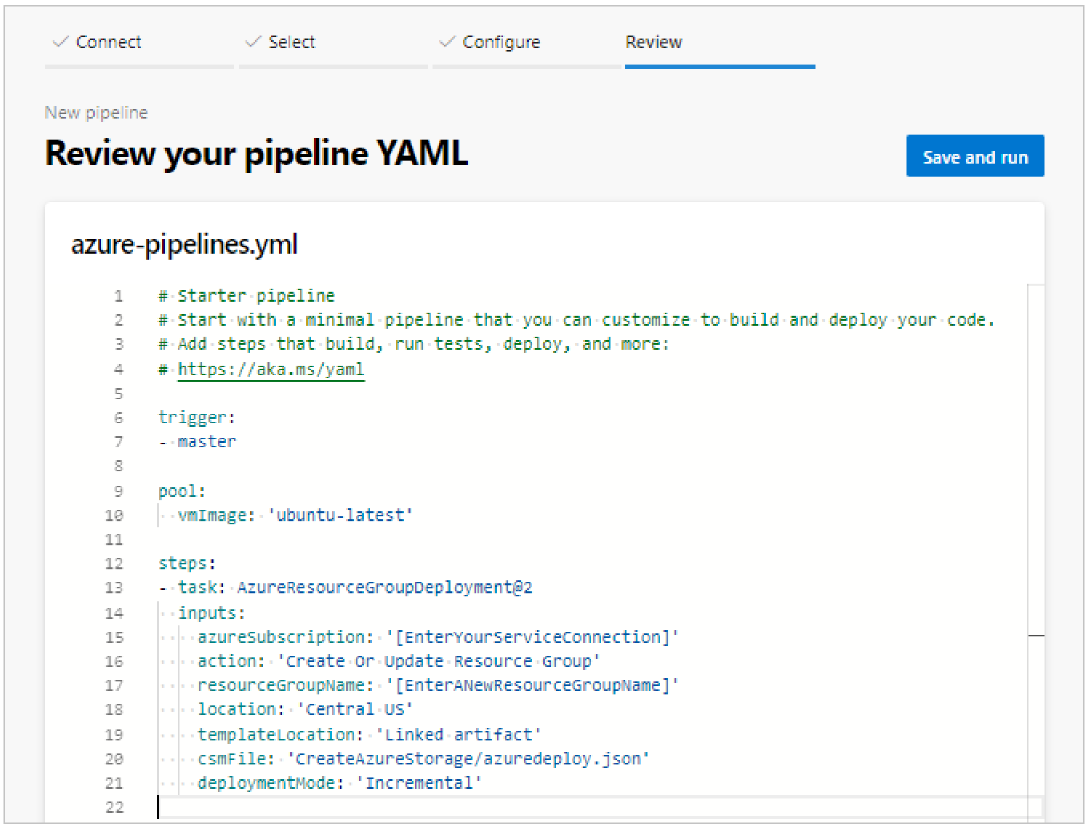
4. From the **Select** tab, select your repository. The default name is **[YourAccountName]/[YourGitHubRepositoryName]**.

5. From the **Configure** tab, select **Starter pipeline**. It shows the **azure-pipelines.yml** pipeline file with two script steps.

6. Replace the **steps** section with the following YAML:

```
steps:
- task: AzureResourceGroupDeployment@2
  inputs:
    azureSubscription: '[YourServiceConnectionName]'
    action: 'Create Or Update Resource Group'
    resourceGroupName: '[EnterANewResourceGroupName]'
    location: 'Central US'
    templateLocation: 'Linked artifact'
    csmFile: 'CreateAzureStorage/azuredeploy.json'
    deploymentMode: 'Incremental'
```

It shall look like:



Review your pipeline YAML

azure-pipelines.yml

```
# Starter pipeline
# Start with a minimal pipeline that you can customize to build and deploy your code.
# Add steps that build, run tests, deploy, and more:
# https://aka.ms/yaml

trigger:
- master

pool:
  vmImage: 'ubuntu-latest'

steps:
- task: AzureResourceGroupDeployment@2
  inputs:
    azureSubscription: '[EnterYourServiceConnection]'
    action: 'Create Or Update Resource Group'
    resourceGroupName: '[EnterANewResourceGroupName]'
    location: 'Central US'
    templateLocation: 'Linked artifact'
    csmFile: 'CreateAzureStorage/azuredeploy.json'
    deploymentMode: 'Incremental'

```

Make the following changes:

- **azureSubscription**: update the value with the service connection created in the previous procedure.
- **action**: the **Create Or Update Resource Group** action does 2 actions - 1. create a resource group if a new resource group name is provided; 2. deploy the template specified.
- **resourceGroupName**: specify a new resource group name. For example, **AzureRmPipeline-rg**.
- **location**: specify the location for the resource group.
- **templateLocation**: when **Linked artifact** is specified, the task looks for the template file directly from the connected repository.
- **csmFile** is the path to the template file. You don't need to specify a template parameters file because all of the parameters defined in the template have default values.

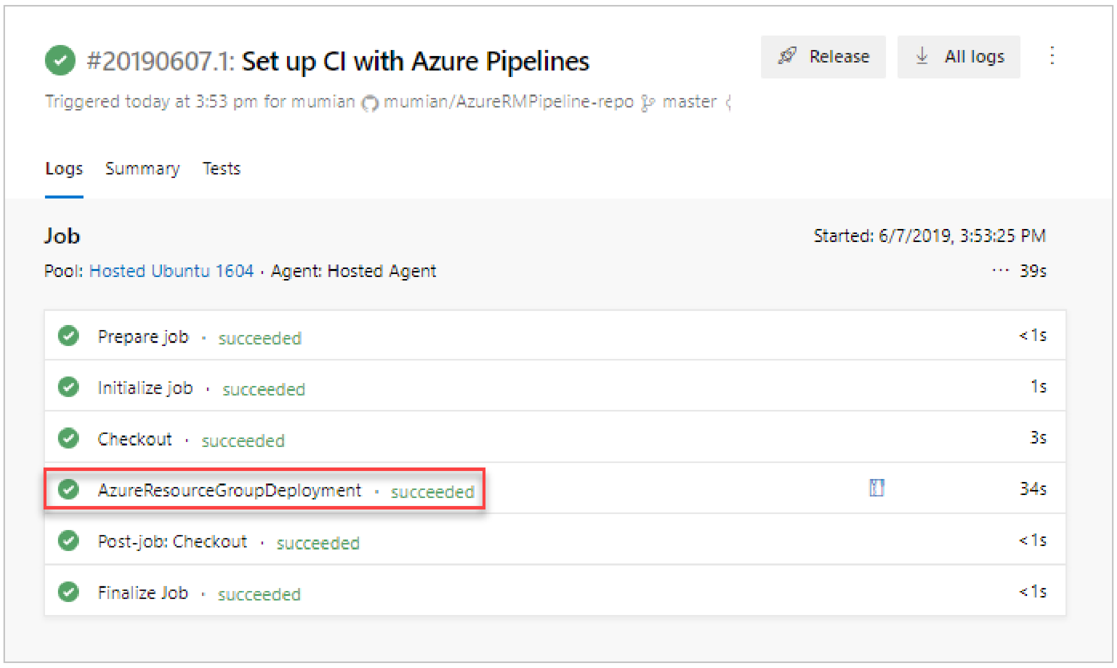
For more information about the task, see Azure Resource Group Deployment task

7. Select **Save and run**.

8. Select **Save and run** again. A copy of the YAML file is saved into the connected repository. You can see the

YAML file by browse to your repository.

9. Verify that the pipeline is executed successfully.



#20190607.1: Set up CI with Azure Pipelines

Triggered today at 3:53 pm for mumian mumian/AzureRMPipeline-repo master

Job

Pool: Hosted Ubuntu 1604 · Agent: Hosted Agent

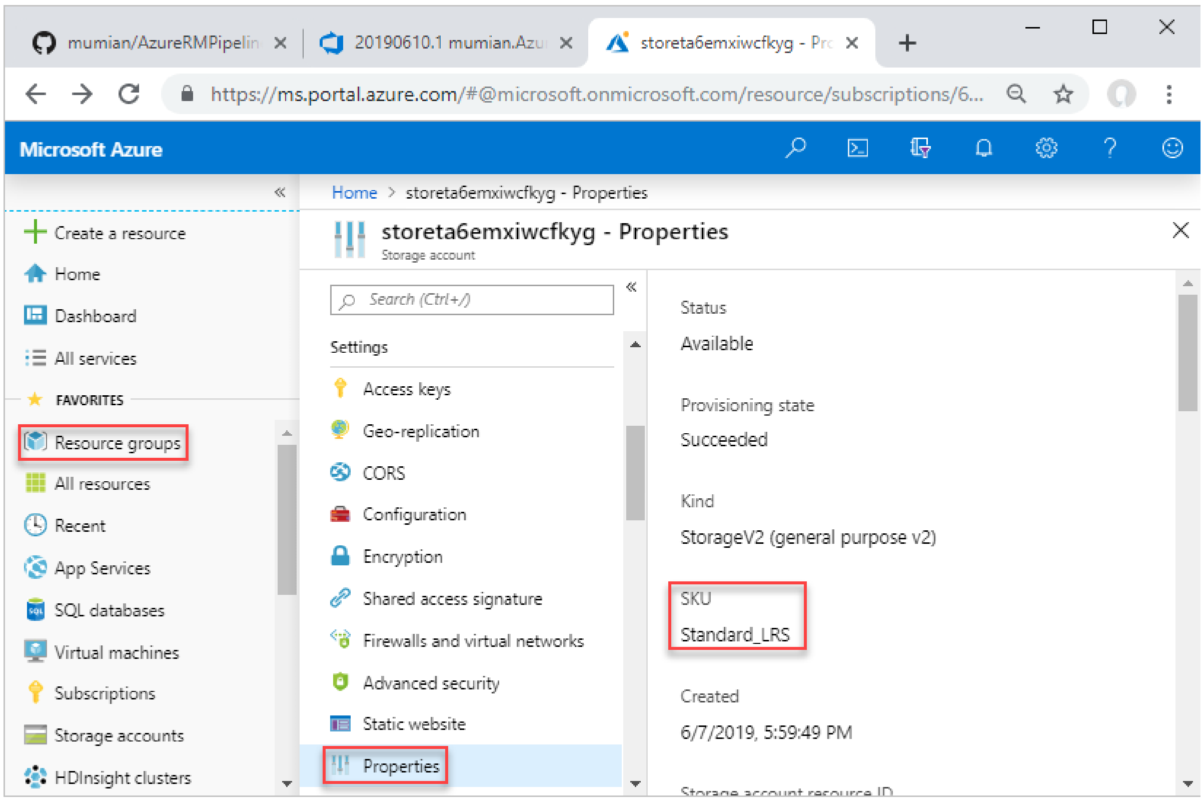
Started: 6/7/2019, 3:53:25 PM

··· 39s

```
Prepare job · succeeded                          <1s
Initialize job · succeeded                        1s
Checkout · succeeded                              3s
AzureResourceGroupDeployment · succeeded         34s
Post-job: Checkout · succeeded                   <1s
Finalize Job · succeeded                         <1s
```

Verify the deployment

1. Sign in to the Azure portal.
2. Open the resource group. The name is what you specified in the pipeline YAML file. You shall see one storage account created. The storage account name starts with **store**.
3. Select the storage account name to open it.
4. Select **Properties**. Notice the **SKU** is **Standard_LRS**.



https://ms.portal.azure.com/#@microsoft.onmicrosoft.com/resource/subscriptions/6...

Home > storeta6emxiwcfkyg - Properties

storeta6emxiwcfkyg - Properties

Storage account

Status

Available

Provisioning state

Succeeded

Kind

StorageV2 (general purpose v2)

SKU

Standard_LRS

Created

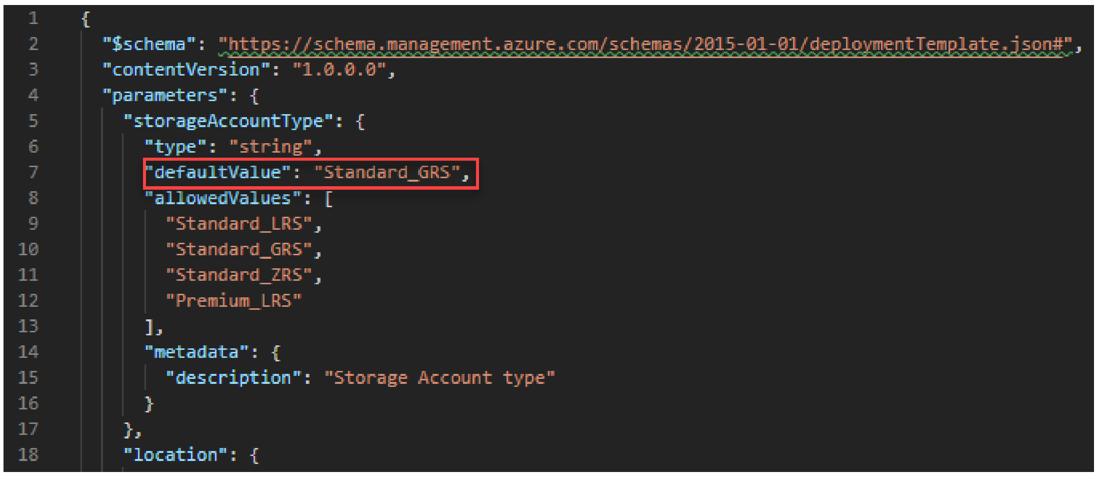
6/7/2019, 5:59:49 PM

Storage account resource ID

Update and redeploy

When you update the template and push the changes to the remote repository, the pipeline automatically updates the resources, the storage account in this case.

1. Open **azuredeploy.json** from your local repository in Visual Studio Code.
2. Update the **defaultValue** of **storageAccountType** to **Standard_GRS**. See the following screenshot:



```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "storageAccountType": {
      "type": "string",
      "defaultValue": "Standard_GRS",
      "allowedValues": [
        "Standard_LRS",
        "Standard_GRS",
        "Standard_ZRS",
        "Premium_LRS"
      ],
      "metadata": {
        "description": "Storage Account type"
      }
    },
    "location": {
```

3. Save the changes.
4. Push the changes to the remote repository by running the following commands from Git Bash/Shell.

```
git pull origin master
git add .
git commit -m “Add a new create storage account template.”
git push origin master
```

   The first command syncs the local repository with the remote repository. Remember the pipeline YAML file was added to the remote repository.

   With the master branch of the remote repository updated, the pipeline is fired again.

To verify the changes, you can check the SKU of the storage account. See Verify the deployment.

Clean up resources

When the Azure resources are no longer needed, clean up the resources you deployed by deleting the resource group.

1. From the Azure portal, select **Resource group** from the left menu.
2. Enter the resource group name in the **Filter by name** field.
3. Select the resource group name.
4. Select **Delete resource group** from the top menu.

You might also want to delete the GitHub repository and the Azure DevOps project.

Next steps

In this tutorial, you create an Azure DevOps pipeline to deploy an Azure Resource Manager template. To learn how to deploy Azure resources across multiple regions, and how to use safe deployment practices, see

Use Azure Deployment Manager

Tutorial: Use Azure Deployment Manager with Resource Manager templates (Public preview)

6/18/2019 • 14 minutes to read • Edit Online

Learn how to use Azure Deployment Manager to deploy your applications across multiple regions. If you prefer a faster approach, Azure Deployment Manager quickstart creates the required configurations in your subscription and customizes the artifacts to deploy an application across multiple regions. The quickstart performs the same tasks as it does in this tutorial.

To use Deployment Manager, you need to create two templates:

- **A topology template**: describes the Azure resources the make up your applications and where to deploy them.
- **A rollout template**: describes the steps to take when deploying your applications.

IMPORTANT

If your subscription is marked for Canary to test out new Azure features, you can only use Azure Deployment Manager to deploy to the Canary regions.

This tutorial covers the following tasks:

- Understand the scenario
- Download the tutorial files
- Prepare the artifacts
- Create the user-defined managed identity
- Create the service topology template
- Create the rollout template
- Deploy the templates
- Verify the deployment
- Deploy the newer version
- Clean up resources

Additional resources:

- The Azure Deployment Manager REST API reference.
- Tutorial: Use health check in Azure Deployment Manager.

If you don't have an Azure subscription, create a free account before you begin.

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

Prerequisites

To complete this article, you need:

- Some experience with developing Azure Resource Manager templates.
- Azure PowerShell. For more information, see Get started with Azure PowerShell.
- Deployment Manager cmdlets. To install these prerelease cmdlets, you need the latest version of PowerShellGet. To get the latest version, see Installing PowerShellGet. After installing PowerShellGet, close your PowerShell window. Open a new elevated PowerShell window, and use the following command:

  ```
  Install-Module -Name Az.DeploymentManager
  ```

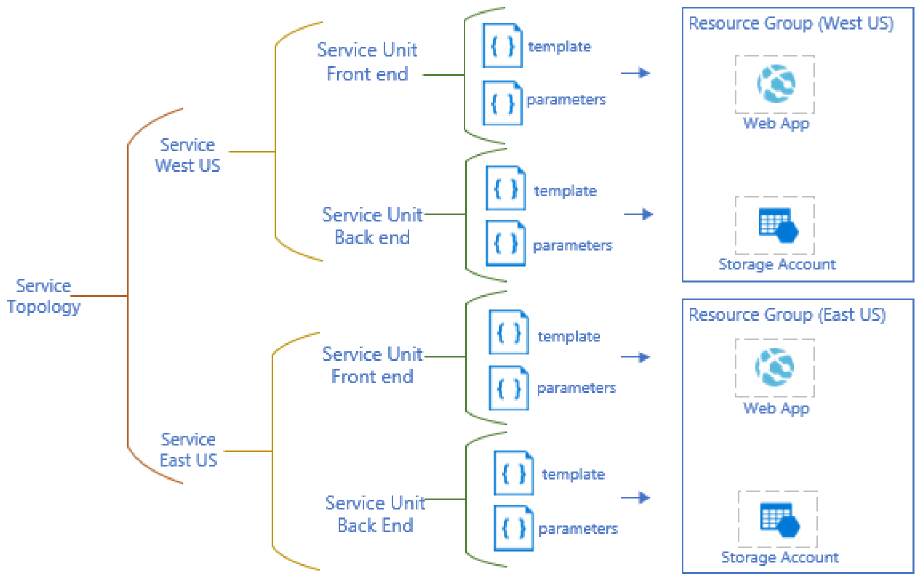
- Microsoft Azure Storage Explorer. Azure Storage Explorer is not required, but it makes things easier.

Understand the scenario

The service topology template describes the Azure resources the make up your service and where to deploy them. The service topology definition has the following hierarchy:

- Service topology
  - Services
    - Service units

The following diagram illustrates the service topology used in this tutorial:



There are two services allocated in the west U.S. and the east U.S. locations. Each service has two service units - a web application frontend and a storage account for the backend. The service unit definitions contain links to the template and parameter files for creating the web applications and the storage accounts.

Download the tutorial files

1. Download the templates and the artifacts used by this tutorial.
2. Unzip the files to your location computer.

Under the root folder, there are two folders:

- **ADMTemplates**: contains the Deployment Manager templates, that include:
  - CreateADMServiceTopology.json
  - CreateADMServiceTopology.Parameters.json

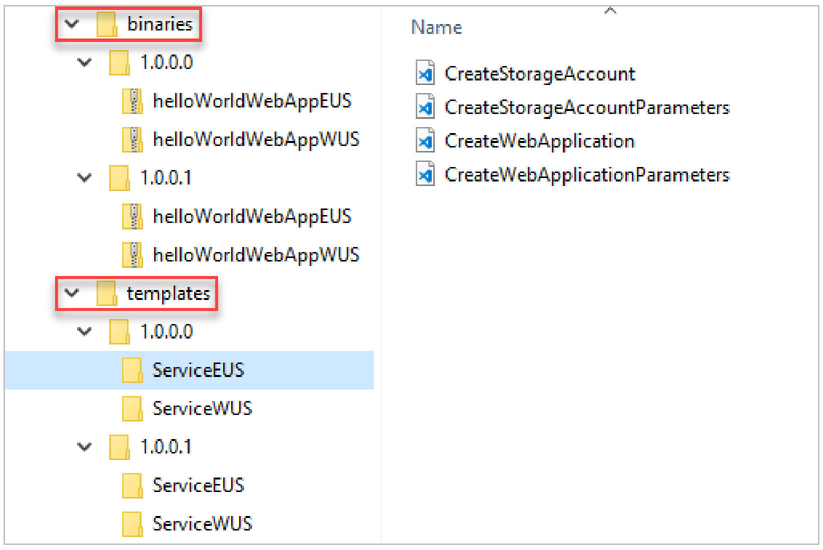
- CreateADMRollout.json
- CreateADMRollout.Parameters.json

- **ArtifactStore**: contains both the template artifacts and the binary artifacts. See Prepare the artifacts.

Note there are two sets of templates. One set is the Deployment Manager templates that are used to deploy the service topology and the rollout; the other set is called from the service units to create web services and storage accounts.

Prepare the artifacts

The ArtifactStore folder from the download contains two folders:

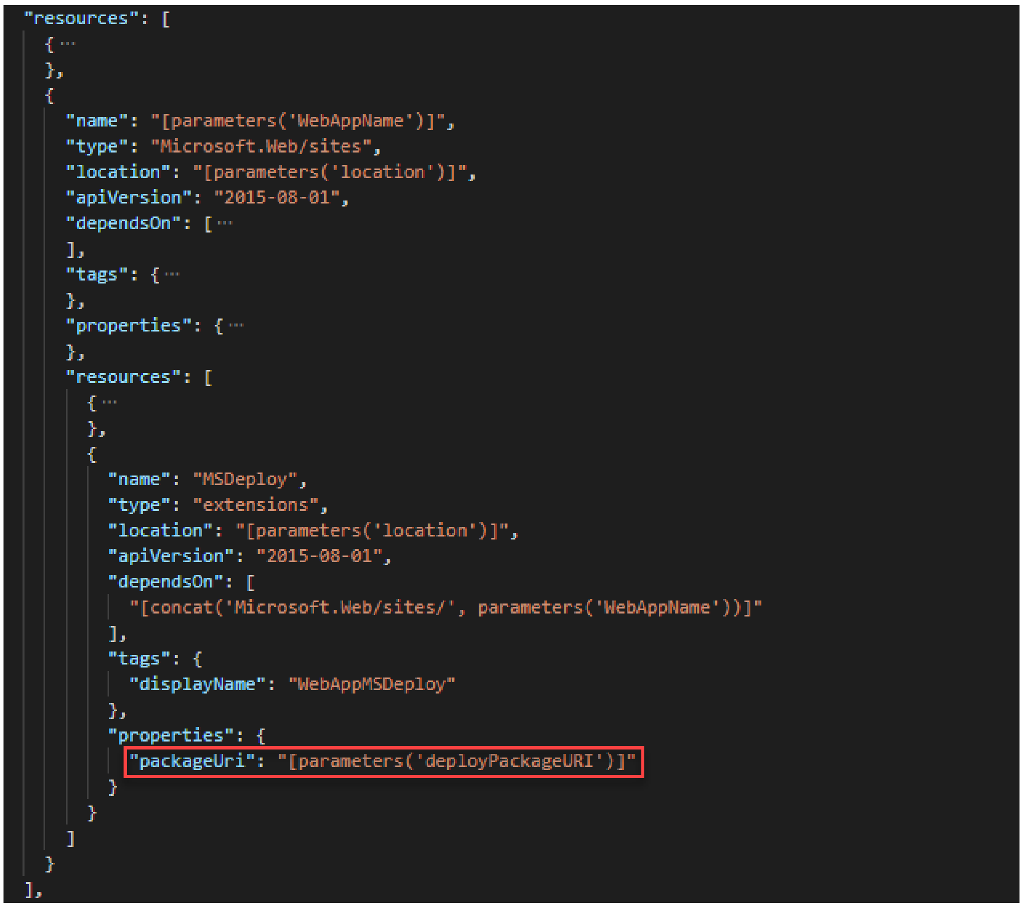


- The **templates** folder: contains the template artifacts. **1.0.0.0** and **1.0.0.1** represent the two versions of the binary artifacts. Within each version, there is a folder for each service (Service East U.S. and Service West U.S.). Each service has a pair of template and parameter files for creating a storage account, and another pair for creating a web application. The web application template calls a compressed package, which contains the web application files. The compressed file is a binary artifact stored in the binaries folder.
- The **binaries** folder: contains the binary artifacts. **1.0.0.0** and **1.0.0.1** represent the two versions of the binary artifacts. Within each version, there is one zip file for creating the web application in the west U.S. location, and the other zip file to create the web application in the east U.S. location.

The two versions (1.0.0.0 and 1.0.0.1) are for the revision deployment. Even though both the template artifacts and the binary artifacts have two versions, only the binary artifacts are different between the two versions. In practice, binary artifacts are updated more frequently comparing to template artifacts.

1. Open **\ArtifactStore\templates\1.0.0.0\ServiceWUS\CreateStorageAccount.json** in a text editor. It is a basic template for creating a storage account.

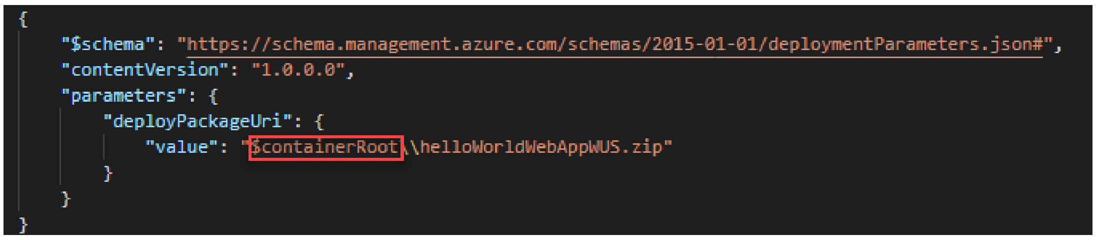
2. Open **\ArtifactStore\templates\1.0.0.0\ServiceWUS\CreateWebApplication.json**.



```
"resources": [
  {…
  },
  {
    "name": "[parameters('WebAppName')]",
    "type": "Microsoft.Web/sites",
    "location": "[parameters('location')]",
    "apiVersion": "2015-08-01",
    "dependsOn": […
    ],
    "tags": {…
    },
    "properties": {…
    },
    "resources": [
      {…
      },
      {
        "name": "MSDeploy",
        "type": "extensions",
        "location": "[parameters('location')]",
        "apiVersion": "2015-08-01",
        "dependsOn": [
          "[concat('Microsoft.Web/sites/', parameters('WebAppName'))]"
        ],
        "tags": {
          "displayName": "WebAppMSDeploy"
        },
        "properties": {
          "packageUri": "[parameters('deployPackageURI')]"
        }
      }
    ]
  }
],
```

The template calls a deploy package, which contains the files of the web application. In this tutorial, the compressed package only contains an index.html file.

3. Open **\ArtifactStore\templates\1.0.0.0\ServiceWUS\CreateWebApplicationParameters.json**.



```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "deployPackageUri": {
            "value": "$containerRoot\helloWorldWebAppWUS.zip"
        }
    }
}
```

The value of deployPackageUri is the path to the deployment package. The parameter contains a **$containerRoot** variable. The value of $containerRoot is provided in the rollout template by concatenating the artifact source SAS location, artifact root, and deployPackageUri.

4. Open **\ArtifactStore\binaries\1.0.0.0\helloWorldWebAppWUS.zip\index.html**.

```
<html>
  <head>
    <title>Azure Deployment Manager tutorial</title>
  </head>
  <body>
    <p>Hello world from west U.S.!</p>
    <p>Version 1.0.0.0</p>
  </body>
</html>
```

The html shows the location and the version information. The binary file in the 1.0.0.1 folder shows "Version 1.0.0.1". After you deploy the service, you can browse to these pages.

5. Check out other artifact files. It helps you to understand the scenario better.

Template artifacts are used by the service topology template, and binary artifacts are used by the rollout template. Both the topology template and the rollout template define an artifact source Azure resource, which is a resource used to point Resource Manager to the template and binary artifacts that are used in the deployment. To simplify the tutorial, one storage account is used to store both the template artifacts and the binary artifacts. Both artifact sources point to the same storage account.

1. Create an Azure storage account. For the instructions, see Quickstart: Upload, download, and list blobs using the Azure portal.

2. Create a blob container in the storage account.

3. Copy the two folders (binaries and templates) and the content of the two folders to the blob container. Microsoft Azure Storage Explorer supports the drag and drop feature.

4. Get the SAS location of the container using the following instructions:

   a. From Azure Storage Explorer, navigate to the blob container.
   b. Right-click the blob container from the left pane, and then select **Get Shared Access Signature**.
   c. Configure **Start time** and **Expiry time**.
   d. Select **Create**.
   e. Make a copy of the URL. This URL is needed to populate a field in the two parameter files, topology parameters file and rollout parameters file.

Create the user-assigned managed identity

Later in the tutorial, you deploy a rollout. A user-assigned managed identity is needed to perform the deployment actions (for example, deploy the web applications and the storage account). This identity must be granted access to the Azure subscription you're deploying the service to, and have sufficient permission to complete the artifact deployment.

You need to create a user-assigned managed identity and configure the access control for your subscription.

IMPORTANT

The user-assigned managed identity must be in the same location as the rollout. Currently, the Deployment Manager resources, including rollout, can only be created in either Central US or East US 2. However, this is only true for the Deployment Manager resources (such as the service topology, services, service units, rollout, and steps). Your target resources can be deployed to any supported Azure region. In this tutorial, for example, the Deployment Manager resources are deployed to Central US, but the services are deployed to East US and West US. This restriction will be lifted in the future.

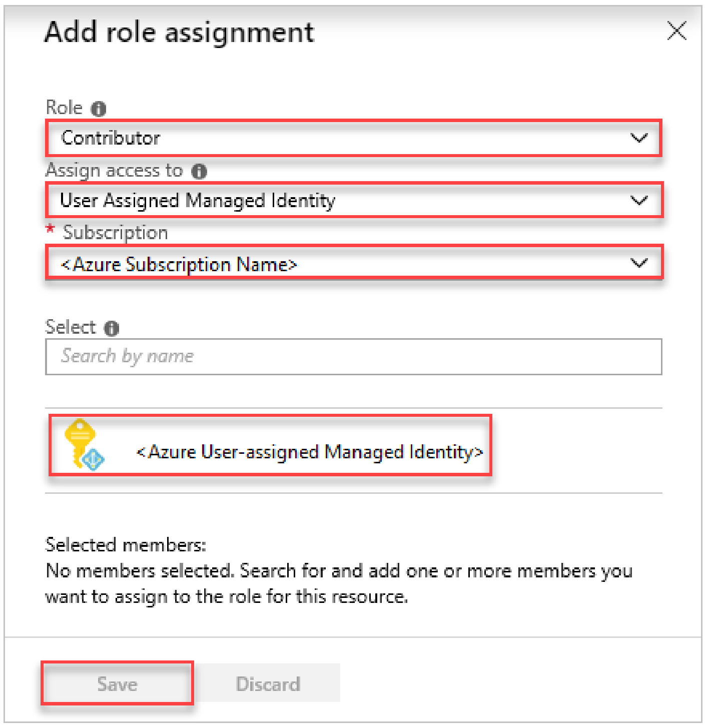
1. Sign in to the Azure portal.

2. Create a user-assigned managed identity.

3. From the portal, select **Subscriptions** from the left menu, and then select your subscription.

4. Select **Access control (IAM)**, and then select **Add role assignment**.

5. Enter or select the following values:


Add role assignment

Role
Contributor
Assign access to
User Assigned Managed Identity
* Subscription
<Azure Subscription Name>

Select
Search by name

<Azure User-assigned Managed Identity>

Selected members:
No members selected. Search for and add one or more members you want to assign to the role for this resource.

- **Role**: give sufficient permission to complete the artifact deployment (the web applications and the storage accounts). Select **Contributor** in this tutorial. In practice, you want to restrict the permissions to the minimum.
- **Assigned access to**: select **User Assigned Managed Identity**.
- Select the user-assigned managed identity you created earlier in the tutorial.

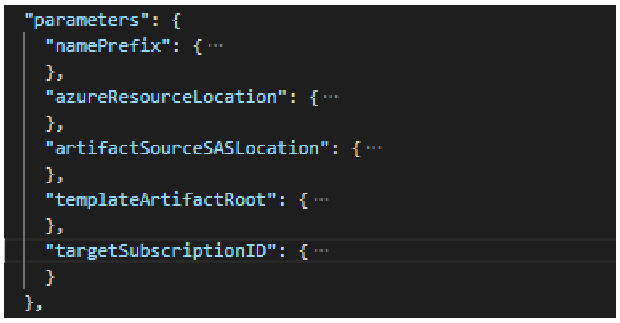
6. Select **Save**.

Create the service topology template

Open **\ADMTemplates\CreateADMServiceTopology.json**.

The parameters

The template contains the following parameters:


```
"parameters": {
  "namePrefix": {⋯
  },
  "azureResourceLocation": {⋯
  },
  "artifactSourceSASLocation": {⋯
  },
  "templateArtifactRoot": {⋯
  },
  "targetSubscriptionID": {⋯
  }
},
```

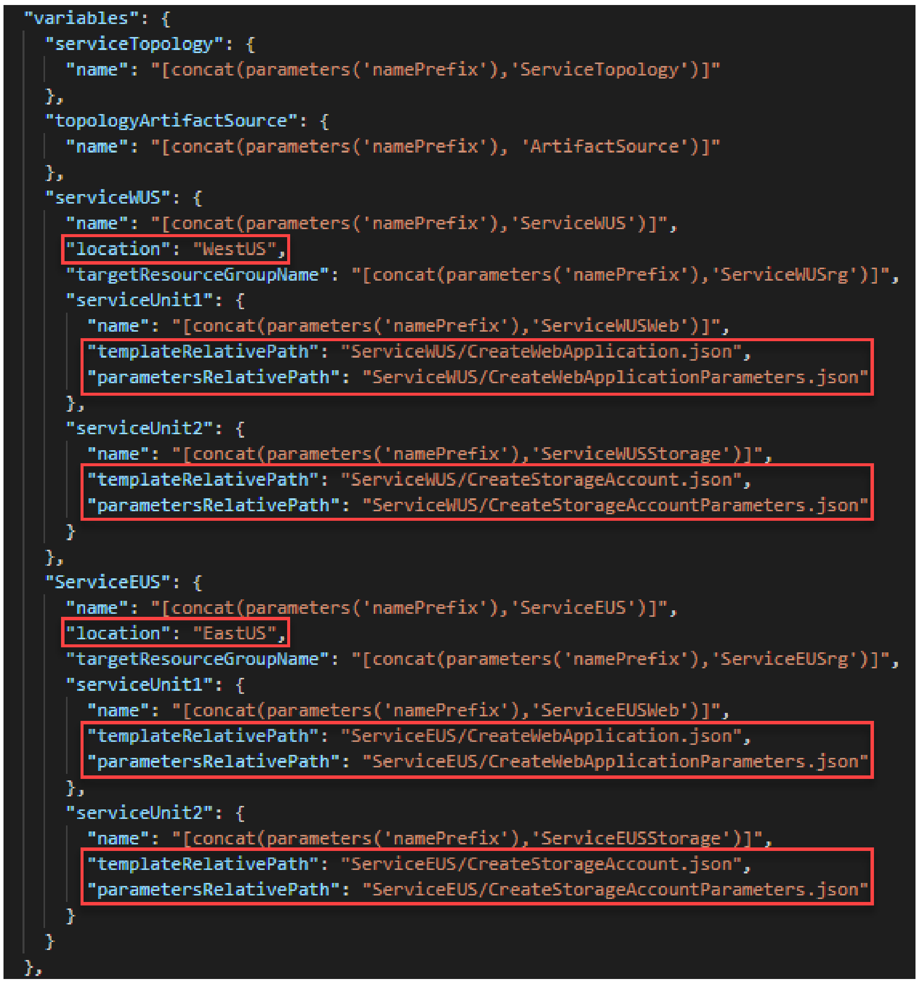
- **namePrefix**: This prefix is used to create the names for the Deployment Manager resources. For example, using the "jdoe" prefix, the service topology name is **jdoe**ServiceTopology. The resource names are defined in the variables section of this template.
- **azureResourcelocation**: To simplify the tutorial, all resources share this location unless it is specified otherwise. Currently, Azure Deployment Manager resources can only be created in either **Central US** or **East US 2**.
- **artifactSourceSASLocation**: The SAS URI to the Blob container where service unit template and parameters files are stored for deployment. See Prepare the artifacts.
- **templateArtifactRoot**: The offset path from the Blob container where the templates and parameters are stored. The default value is **templates/1.0.0.0**. Don't change this value unless you want to change the folder structure explained in Prepare the artifacts. Relative paths are used in this tutorial. The full path is constructed

by concatenating **artifactSourceSASLocation**, **templateArtifactRoot**, and **templateArtifactSourceRelativePath** (or **parametersArtifactSourceRelativePath**).

- **targetSubscriptionID**: The subscription ID to which the Deployment Manager resources are going to be deployed and billed. Use your subscription ID in this tutorial.

The variables

The variables section defines the names of the resources, the Azure locations for the two services: **Service WUS** and **Service EUS**, and the artifact paths:



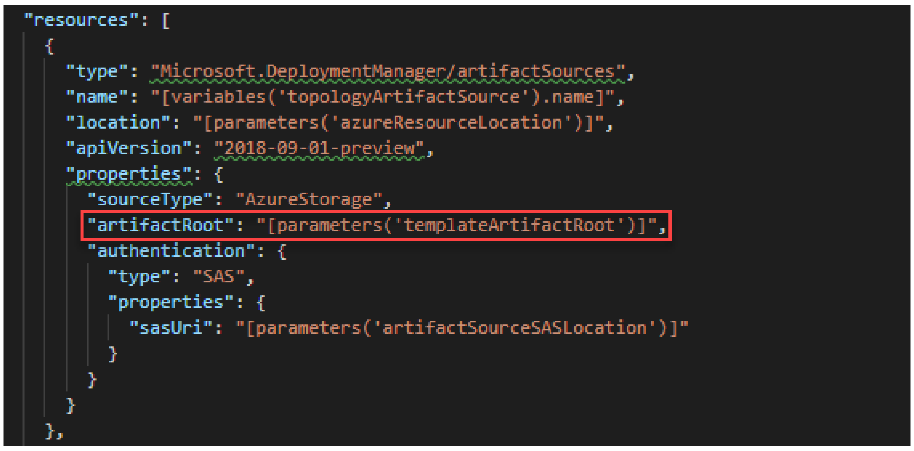
```
"variables": {
  "serviceTopology": {
    "name": "[concat(parameters('namePrefix'),'ServiceTopology')]"
  },
  "topologyArtifactSource": {
    "name": "[concat(parameters('namePrefix'), 'ArtifactSource')]"
  },
  "serviceWUS": {
    "name": "[concat(parameters('namePrefix'),'ServiceWUS')]",
    "location": "WestUS",
    "targetResourceGroupName": "[concat(parameters('namePrefix'),'ServiceWUSrg')]",
    "serviceUnit1": {
      "name": "[concat(parameters('namePrefix'),'ServiceWUSWeb')]",
      "templateRelativePath": "ServiceWUS/CreateWebApplication.json",
      "parametersRelativePath": "ServiceWUS/CreateWebApplicationParameters.json"
    },
    "serviceUnit2": {
      "name": "[concat(parameters('namePrefix'),'ServiceWUSStorage')]",
      "templateRelativePath": "ServiceWUS/CreateStorageAccount.json",
      "parametersRelativePath": "ServiceWUS/CreateStorageAccountParameters.json"
    }
  },
  "ServiceEUS": {
    "name": "[concat(parameters('namePrefix'),'ServiceEUS')]",
    "location": "EastUS",
    "targetResourceGroupName": "[concat(parameters('namePrefix'),'ServiceEUSrg')]",
    "serviceUnit1": {
      "name": "[concat(parameters('namePrefix'),'ServiceEUSWeb')]",
      "templateRelativePath": "ServiceEUS/CreateWebApplication.json",
      "parametersRelativePath": "ServiceEUS/CreateWebApplicationParameters.json"
    },
    "serviceUnit2": {
      "name": "[concat(parameters('namePrefix'),'ServiceEUSStorage')]",
      "templateRelativePath": "ServiceEUS/CreateStorageAccount.json",
      "parametersRelativePath": "ServiceEUS/CreateStorageAccountParameters.json"
    }
  }
},
```

Compare the artifact paths with the folder structure that you uploaded to the storage account. Notice the artifact paths are relative paths. The full path is constructed by concatenating **artifactSourceSASLocation**, **templateArtifactRoot**, and **templateArtifactSourceRelativePath** (or **parametersArtifactSourceRelativePath**).

The resources

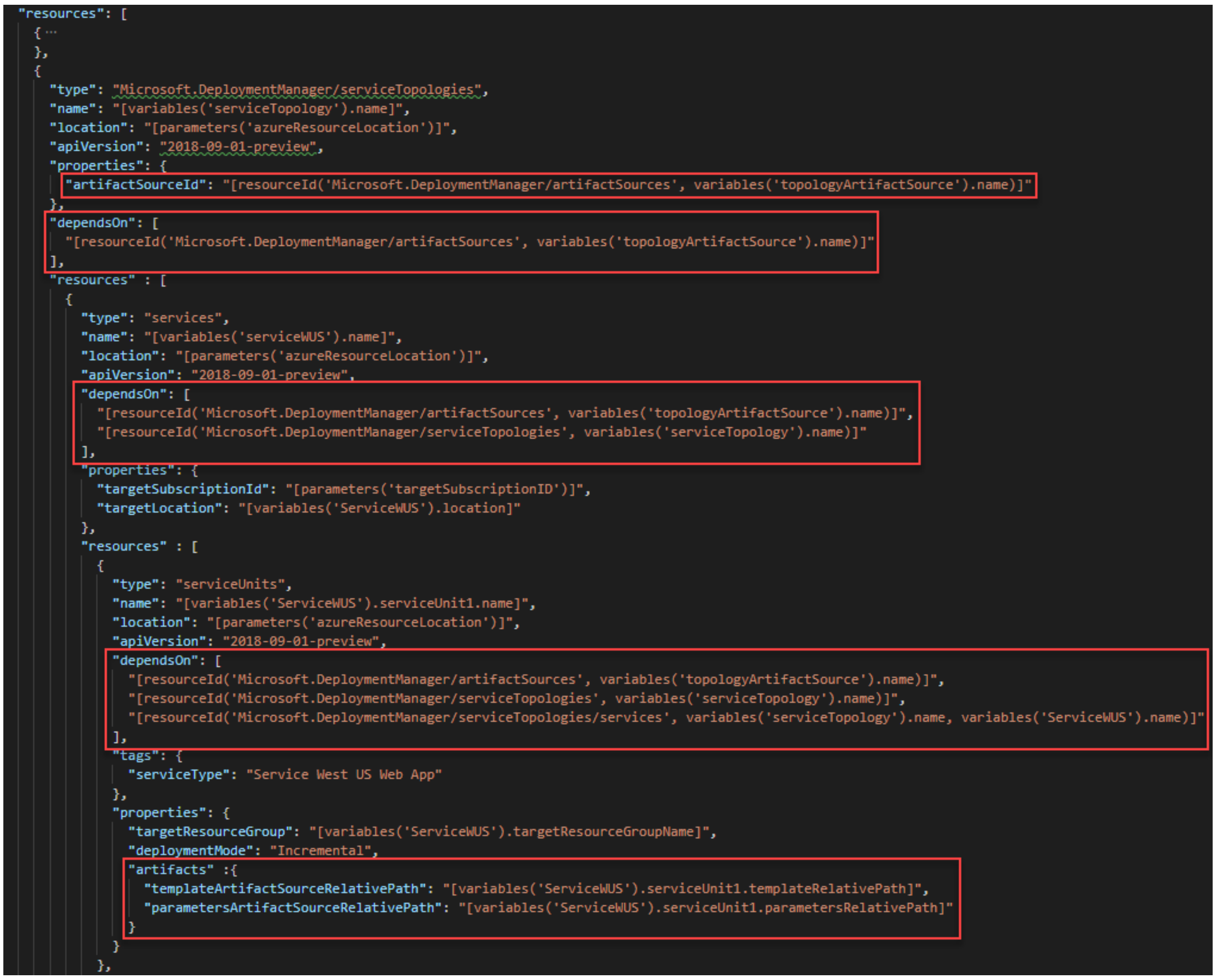
On the root level, there are two resources defined: *an artifact source*, and *a service topology*.

The artifact source definition is:



```
"resources": [
  {
    "type": "Microsoft.DeploymentManager/artifactSources",
    "name": "[variables('topologyArtifactSource').name]",
    "location": "[parameters('azureResourceLocation')]",
    "apiVersion": "2018-09-01-preview",
    "properties": {
      "sourceType": "AzureStorage",
      "artifactRoot": "[parameters('templateArtifactRoot')]",
      "authentication": {
        "type": "SAS",
        "properties": {
          "sasUri": "[parameters('artifactSourceSASLocation')]"
        }
      }
    }
  },
```

The following screenshot only shows some parts of the service topology, services, and service units definitions:



```
"resources": [
  {…
  },
  {
    "type": "Microsoft.DeploymentManager/serviceTopologies",
    "name": "[variables('serviceTopology').name]",
    "location": "[parameters('azureResourceLocation')]",
    "apiVersion": "2018-09-01-preview",
    "properties": {
      "artifactSourceId": "[resourceId('Microsoft.DeploymentManager/artifactSources', variables('topologyArtifactSource').name)]"
    },
    "dependsOn": [
      "[resourceId('Microsoft.DeploymentManager/artifactSources', variables('topologyArtifactSource').name)]"
    ],
    "resources" : [
      {
        "type": "services",
        "name": "[variables('serviceWUS').name]",
        "location": "[parameters('azureResourceLocation')]",
        "apiVersion": "2018-09-01-preview",
        "dependsOn": [
          "[resourceId('Microsoft.DeploymentManager/artifactSources', variables('topologyArtifactSource').name)]",
          "[resourceId('Microsoft.DeploymentManager/serviceTopologies', variables('serviceTopology').name)]"
        ],
        "properties": {
          "targetSubscriptionId": "[parameters('targetSubscriptionID')]",
          "targetLocation": "[variables('ServiceWUS').location]"
        },
        "resources" : [
          {
            "type": "serviceUnits",
            "name": "[variables('ServiceWUS').serviceUnit1.name]",
            "location": "[parameters('azureResourceLocation')]",
            "apiVersion": "2018-09-01-preview",
            "dependsOn": [
              "[resourceId('Microsoft.DeploymentManager/artifactSources', variables('topologyArtifactSource').name)]",
              "[resourceId('Microsoft.DeploymentManager/serviceTopologies', variables('serviceTopology').name)]",
              "[resourceId('Microsoft.DeploymentManager/serviceTopologies/services', variables('serviceTopology').name, variables('ServiceWUS').name)]"
            ],
            "tags": {
              "serviceType": "Service West US Web App"
            },
            "properties": {
              "targetResourceGroup": "[variables('ServiceWUS').targetResourceGroupName]",
              "deploymentMode": "Incremental",
              "artifacts" :{
                "templateArtifactSourceRelativePath": "[variables('ServiceWUS').serviceUnit1.templateRelativePath]",
                "parametersArtifactSourceRelativePath": "[variables('ServiceWUS').serviceUnit1.parametersRelativePath]"
              }
            }
          },
```

- **artifactSourceId** is used to associate the artifact source resource to the service topology resource.
- **dependsOn**: All the service topology resources depend on the artifact source resource.
- **artifacts** point to the template artifacts. Relative paths are used here. The full path is constructed by concatenating artifactSourceSASLocation (defined in the artifact source), artifactRoot (defined in the artifact source), and templateArtifactSourceRelativePath (or parametersArtifactSourceRelativePath).

Topology parameters file

You create a parameters file used with the topology template.

1. Open **\ADMTemplates\CreateADMServiceTopology.Parameters** in Visual Studio Code or any text editor.

2. Fill the parameter values:

- **namePrefix**: Enter a string with 4-5 characters. This prefix is used to create unique azure resource names.
- **azureResourceLocation**: If you are not familiar with Azure locations, use **centralus** in this tutorial.
- **artifactSourceSASLocation**: Enter the SAS URI to the root directory (the Blob container) where service unit template and parameters files are stored for deployment. See Prepare the artifacts.
- **templateArtifactRoot**: Unless you change the folder structure of the artifacts, use **templates/1.0.0.0** in this tutorial.
- **targetScriptionID**: Enter your Azure subscription ID.

IMPORTANT

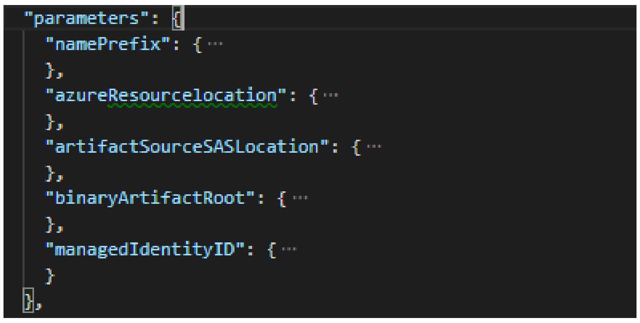
The topology template and the rollout template share some common parameters. These parameters must have the same values. These parameters are: **namePrefix**, **azureResourceLocation**, and **artifactSourceSASLocation** (both artifact sources share the same storage account in this tutorial).

Create the rollout template

Open **\ADMTemplates\CreateADMRollout.json**.

The parameters

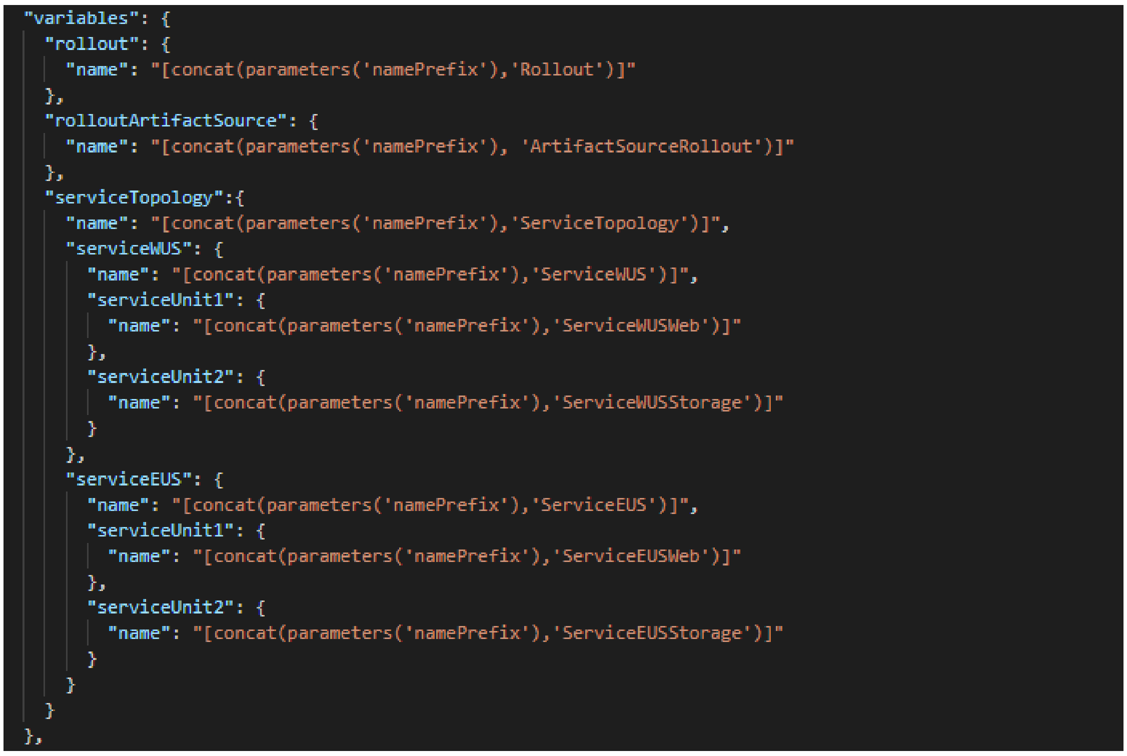
The template contains the following parameters:


```
"parameters": {
  "namePrefix": {⋯
  },
  "azureResourcelocation": {⋯
  },
  "artifactSourceSASLocation": {⋯
  },
  "binaryArtifactRoot": {⋯
  },
  "managedIdentityID": {⋯
  }
},
```

- **namePrefix**: This prefix is used to create the names for the Deployment Manager resources. For example, using the "jdoe" prefix, the rollout name is **jdoe**Rollout. The names are defined in the variables section of the template.
- **azureResourcelocation**: To simplify the tutorial, all Deployment Manager resources share this location unless it is specified otherwise. Currently, Azure Deployment Manager resources can only be created in either **Central US** or **East US 2**.
- **artifactSourceSASLocation**: The SAS URI to the root directory (the Blob container) where service unit template and parameters files are stored for deployment. See Prepare the artifacts.
- **binaryArtifactRoot**: The default value is **binaries/1.0.0.0**. Don't change this value unless you want to change the folder structure explained in Prepare the artifacts. Relative paths are used in this tutorial. The full path is constructed by concatenating **artifactSourceSASLocation**, **binaryArtifactRoot**, and the **deployPackageUri** specified in the CreateWebApplicationParameters.json. See Prepare the artifacts.
- **managedIdentityID**: The user-assigned managed identity that performs the deployment actions. See Create the user-assigned managed identity.

The variables

The variables section defines the names of the resources. Make sure the service topology name, the service names, and the service unit names match the names defined in the topology template.



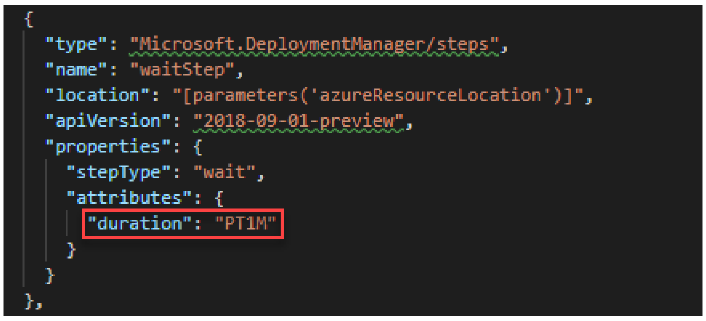
```
"variables": {
  "rollout": {
    "name": "[concat(parameters('namePrefix'),'Rollout')]"
  },
  "rolloutArtifactSource": {
    "name": "[concat(parameters('namePrefix'), 'ArtifactSourceRollout')]"
  },
  "serviceTopology":{
    "name": "[concat(parameters('namePrefix'),'ServiceTopology')]",
    "serviceWUS": {
      "name": "[concat(parameters('namePrefix'),'ServiceWUS')]",
      "serviceUnit1": {
        "name": "[concat(parameters('namePrefix'),'ServiceWUSWeb')]"
      },
      "serviceUnit2": {
        "name": "[concat(parameters('namePrefix'),'ServiceWUSStorage')]"
      }
    },
    "serviceEUS": {
      "name": "[concat(parameters('namePrefix'),'ServiceEUS')]",
      "serviceUnit1": {
        "name": "[concat(parameters('namePrefix'),'ServiceEUSWeb')]"
      },
      "serviceUnit2": {
        "name": "[concat(parameters('namePrefix'),'ServiceEUSStorage')]"
      }
    }
  }
},
```

The resources

On the root level, there are three resources defined: an artifact source, a step, and a rollout.

The artifact source definition is identical to the one defined in the topology template. See Create the service topology template for more information.

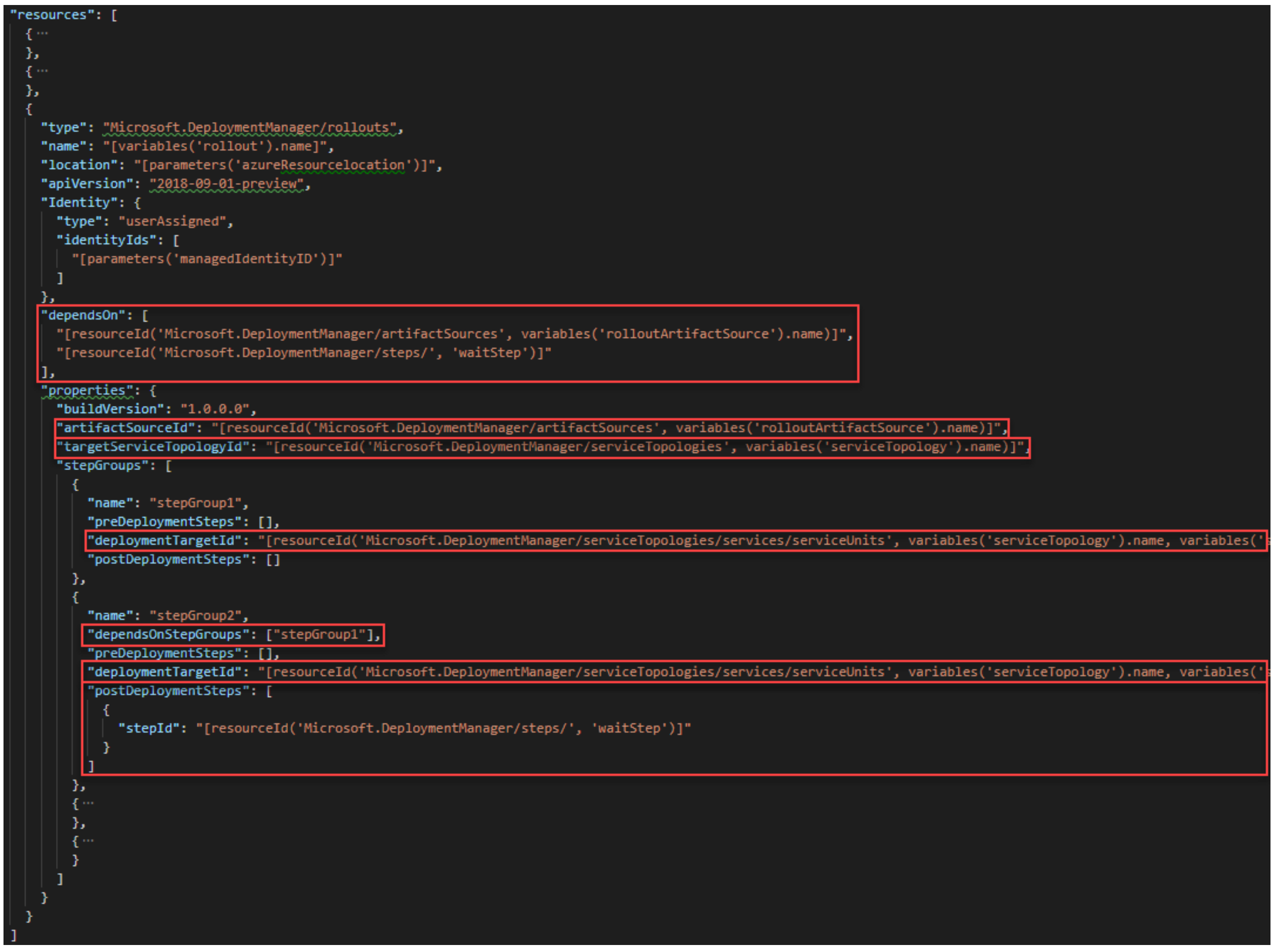
The following screenshot shows the wait step definition:



```
{
  "type": "Microsoft.DeploymentManager/steps",
  "name": "waitStep",
  "location": "[parameters('azureResourceLocation')]",
  "apiVersion": "2018-09-01-preview",
  "properties": {
    "stepType": "wait",
    "attributes": {
      "duration": "PT1M"
    }
  }
},
```

The duration is using the ISO 8601 standard. **PT1M** (capital letters are required) is an example of a 1-minute wait.

The following screenshot only shows some parts of the rollout definition:



```
"resources": [
  {⋯
  },
  {⋯
  },
  {
    "type": "Microsoft.DeploymentManager/rollouts",
    "name": "[variables('rollout').name]",
    "location": "[parameters('azureResourcelocation')]",
    "apiVersion": "2018-09-01-preview",
    "Identity": {
      "type": "userAssigned",
      "identityIds": [
        "[parameters('managedIdentityID')]"
      ]
    },
    "dependsOn": [
      "[resourceId('Microsoft.DeploymentManager/artifactSources', variables('rolloutArtifactSource').name)]",
      "[resourceId('Microsoft.DeploymentManager/steps/', 'waitStep')]"
    ],
    "properties": {
      "buildVersion": "1.0.0.0",
      "artifactSourceId": "[resourceId('Microsoft.DeploymentManager/artifactSources', variables('rolloutArtifactSource').name)]",
      "targetServiceTopologyId": "[resourceId('Microsoft.DeploymentManager/serviceTopologies', variables('serviceTopology').name)]",
      "stepGroups": [
        {
          "name": "stepGroup1",
          "preDeploymentSteps": [],
          "deploymentTargetId": "[resourceId('Microsoft.DeploymentManager/serviceTopologies/services/serviceUnits', variables('serviceTopology').name, variables('
          "postDeploymentSteps": []
        },
        {
          "name": "stepGroup2",
          "dependsOnStepGroups": ["stepGroup1"],
          "preDeploymentSteps": [],
          "deploymentTargetId": "[resourceId('Microsoft.DeploymentManager/serviceTopologies/services/serviceUnits', variables('serviceTopology').name, variables('
          "postDeploymentSteps": [
            {
              "stepId": "[resourceId('Microsoft.DeploymentManager/steps/', 'waitStep')]"
            }
          ]
        },
        {⋯
        },
        {⋯
        }
      ]
    }
  }
]
```

- **dependsOn**: The rollout resource depends on the artifact source resource, and any of the steps defined.
- **artifactSourceId**: used to associate the artifact source resource to the rollout resource.
- **targetServiceTopologyId**: used to associate the service topology resource to the rollout resource.
- **deploymentTargetId**: It is the service unit resource ID of the service topology resource.
- **preDeploymentSteps** and **postDeploymentSteps**: contains the rollout steps. In the template, a wait step is called.
- **dependsOnStepGroups**: configure the dependencies between the step groups.

Rollout parameters file

You create a parameters file used with the rollout template.

1. Open **\ADMTemplates\CreateADMRollout.Parameters** in Visual Studio Code or any text editor.

2. Fill the parameter values:

   - **namePrefix**: Enter a string with 4-5 characters. This prefix is used to create unique azure resource names.

   - **azureResourceLocation**: Currently, Azure Deployment Manager resources can only be created in either **Central US** or **East US 2**.

   - **artifactSourceSASLocation**: Enter the SAS URI to the root directory (the Blob container) where service unit template and parameters files are stored for deployment. See Prepare the artifacts.

   - **binaryArtifactRoot**: Unless you change the folder structure of the artifacts, use **binaries/1.0.0.0** in this tutorial.

   - **managedIdentityID**: Enter the user-assigned managed identity. See Create the user-assigned managed identity. The syntax is:

```
"/subscriptions/<SubscriptionID>/resourcegroups/<ResourceGroupName>/providers/Microsoft.ManagedI
dentity/userassignedidentities/<ManagedIdentityName>"
```

IMPORTANT

The topology template and the rollout template share some common parameters. These parameters must have the same values. These parameters are: **namePrefix**, **azureResourceLocation**, and **artifactSourceSASLocation** (both artifact sources share the same storage account in this tutorial).

Deploy the templates

Azure PowerShell can be used to deploy the templates.

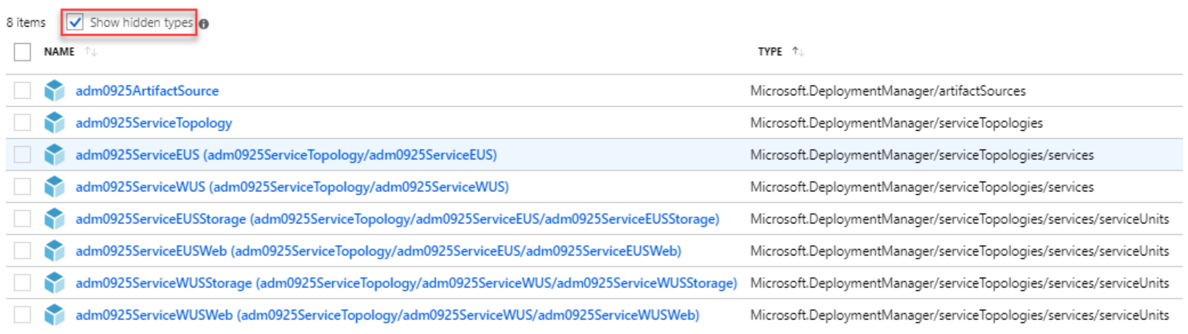
1. Run the script to deploy the service topology.

```
$resourceGroupName = "<Enter a Resource Group Name>"
$location = "Central US"
$filePath = "<Enter the File Path to the Downloaded Tutorial Files>"

# Create a resource group
New-AzResourceGroup -Name $resourceGroupName -Location "$location"

# Create the service topology
New-AzResourceGroupDeployment `
    -ResourceGroupName $resourceGroupName `
    -TemplateFile "$filePath\ADMTemplates\CreateADMServiceTopology.json" `
    -TemplateParameterFile "$filePath\ADMTemplates\CreateADMServiceTopology.Parameters.json"
```

NOTE

`New-AzResourceGroupDeployment` is an asynchronous call. The success message only means the deployment has successfully begun. To verify the deployment, see step 2 and step 4 of this procedure.

2. Verify the service topology and the underlined resources have been created successfully using the Azure portal:



8 items ✓ Show hidden types

NAME	TYPE
adm0925ArtifactSource	Microsoft.DeploymentManager/artifactSources
adm0925ServiceTopology	Microsoft.DeploymentManager/serviceTopologies
adm0925ServiceEUS (adm0925ServiceTopology/adm0925ServiceEUS)	Microsoft.DeploymentManager/serviceTopologies/services
adm0925ServiceWUS (adm0925ServiceTopology/adm0925ServiceWUS)	Microsoft.DeploymentManager/serviceTopologies/services
adm0925ServiceEUSStorage (adm0925ServiceTopology/adm0925ServiceEUS/adm0925ServiceEUSStorage)	Microsoft.DeploymentManager/serviceTopologies/services/serviceUnits
adm0925ServiceEUSWeb (adm0925ServiceTopology/adm0925ServiceEUS/adm0925ServiceEUSWeb)	Microsoft.DeploymentManager/serviceTopologies/services/serviceUnits
adm0925ServiceWUSStorage (adm0925ServiceTopology/adm0925ServiceWUS/adm0925ServiceWUSStorage)	Microsoft.DeploymentManager/serviceTopologies/services/serviceUnits
adm0925ServiceWUSWeb (adm0925ServiceTopology/adm0925ServiceWUS/adm0925ServiceWUSWeb)	Microsoft.DeploymentManager/serviceTopologies/services/serviceUnits

Show hidden types must be selected to see the resources.

3. Deploy the rollout template:

```
# Create the rollout
New-AzResourceGroupDeployment `
    -ResourceGroupName $resourceGroupName `
    -TemplateFile "$filePath\ADMTemplates\CreateADMRollout.json" `
    -TemplateParameterFile "$filePath\ADMTemplates\CreateADMRollout.Parameters.json"
```

4. Check the rollout progress using the following PowerShell script:

```
# Get the rollout status
$rolloutname = "<Enter the Rollout Name>" # "adm0925Rollout" is the rollout name used in this tutorial
Get-AzDeploymentManagerRollout `
    -ResourceGroupName $resourceGroupName `
    -Name $rolloutName `
    -Verbose
```

The Deployment Manager PowerShell cmdlets must be installed before you can run this cmdlet. See Prerequisites. The -Verbose switch can be used to see the whole output.

The following sample shows the running status:

```
VERBOSE:

Status: Succeeded
ArtifactSourceId:
/subscriptions/<AzureSubscriptionID>/resourceGroups/adm0925rg/providers/Microsoft.DeploymentManager/art
ifactSources/adm0925ArtifactSourceRollout
BuildVersion: 1.0.0.0

Operation Info:
    Retry Attempt: 0
    Skip Succeeded: False
    Start Time: 03/05/2019 15:26:13
    End Time: 03/05/2019 15:31:26
    Total Duration: 00:05:12

Service: adm0925ServiceEUS
    TargetLocation: EastUS
    TargetSubscriptionId: <AzureSubscriptionID>

    ServiceUnit: adm0925ServiceEUSStorage
        TargetResourceGroup: adm0925ServiceEUSrg

        Step: Deploy
            Status: Succeeded
            StepGroup: stepGroup3
            Operation Info:
                DeploymentName: 2F535084871E43E7A7A4CE7B45BE06510adm0925ServiceEUSStorage
                CorrelationId: 0b6f030d-7348-48ae-a578-bcd6bcafe78d
                Start Time: 03/05/2019 15:26:32
                End Time: 03/05/2019 15:27:41
                Total Duration: 00:01:08
            Resource Operations:

                Resource Operation 1:
                Name: txq6iwnyq5xle
                Type: Microsoft.Storage/storageAccounts
                ProvisioningState: Succeeded
                StatusCode: OK
                OperationId: 64A6E6EFEF1F7755

...

ResourceGroupName       : adm0925rg
BuildVersion            : 1.0.0.0
ArtifactSourceId        :
/subscriptions/<SubscriptionID>/resourceGroups/adm0925rg/providers/Microsoft.DeploymentManager/artifact
Sources/adm0925ArtifactSourceRollout
TargetServiceTopologyId :
/subscriptions/<SubscriptionID>/resourceGroups/adm0925rg/providers/Microsoft.DeploymentManager/serviceT
opologies/adm0925ServiceTopology
Status                  : Running
TotalRetryAttempts      : 0
OperationInfo           : Microsoft.Azure.Commands.DeploymentManager.Models.PSRolloutOperationInfo
Services                : {adm0925ServiceEUS, adm0925ServiceWUS}
Name                    : adm0925Rollout
Type                    : Microsoft.DeploymentManager/rollouts
Location                : centralus
Id                      :
/subscriptions/<SubscriptionID>/resourcegroups/adm0925rg/providers/Microsoft.DeploymentManager/rollouts
/adm0925Rollout
Tags                    :
```

After the rollout is deployed successfully, you shall see two more resource groups created, one for each service.

Verify the deployment

1. Open the Azure portal.
2. Browse to the newly create web applications under the new resource groups created by the rollout deployment.
3. Open the web application in a web browser. Verify the location and the version on the index.html file.

Deploy the revision

When you have a new version (1.0.0.1) for the web application. You can use the following procedure to redeploy the web application.

1. Open CreateADMRollout.Parameters.json.
2. Update **binaryArtifactRoot** to **binaries/1.0.0.1**.
3. Redeploy the rollout as instructed in Deploy the templates.
4. Verify the deployment as instructed in Verify the deployment. The web page shall show the 1.0.0.1 version.

Clean up resources

When the Azure resources are no longer needed, clean up the resources you deployed by deleting the resource group.

1. From the Azure portal, select **Resource group** from the left menu.
2. Use the **Filter by name** field to narrow down the resource groups created in this tutorial. There shall be 3-4:
    - **<namePrefix>rg**: contains the Deployment Manager resources.
    - **<namePrefix>ServiceWUSrg**: contains the resources defined by ServiceWUS.
    - **<namePrefix>ServiceEUSrg**: contains the resources defined by ServiceEUS.
    - The resource group for the user-defined managed identity.
3. Select the resource group name.
4. Select **Delete resource group** from the top menu.
5. Repeat the last two steps to delete other resource groups created by this tutorial.

Next steps

In this tutorial, you learned how to use Azure Deployment Manager. To integrate health monitoring in Azure Deployment Manager, see Tutorial: Use health check in Azure Deployment Manager.

Tutorial: Use health check in Azure Deployment Manager (Public preview)

7/28/2019 • 9 minutes to read • Edit Online

Learn how to integrate health check in Azure Deployment Manager. This tutorial is based of the Use Azure Deployment Manager with Resource Manager templates tutorial. You must complete that tutorial before you proceed with this one.

In the rollout template used in Use Azure Deployment Manager with Resource Manager templates, you used a wait step. In this tutorial, you replace the wait step with a health check step.

IMPORTANT

If your subscription is marked for Canary to test out new Azure features, you can only use Azure Deployment Manager to deploy to the Canary regions.

This tutorial covers the following tasks:

- Create a health check service simulator
- Revise the rollout template
- Deploy the topology
- Deploy the rollout with unhealthy status
- Verify the rollout deployment
- Deploy the rollout with healthy status
- Verify the rollout deployment
- Clean up resources

Additional resources:

- The Azure Deployment Manager REST API reference.
- An Azure Deployment Manager sample.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

To complete this article, you need:

- Complete Use Azure Deployment Manager with Resource Manager templates.
- Download the templates and the artifacts that is used by this tutorial.

Create a health check service simulator

In production, you typically use one or more monitoring providers. In order to make health integration as easy as possible, Microsoft has been working with some of the top service health monitoring companies to provide you with a simple copy/paste solution to integrate health checks with your deployments. For a list of these companies, see Health monitoring providers. For the purpose of this tutorial, you create an Azure Function to simulate a health monitoring service. This function takes a status code, and returns the same code. Your Azure Deployment Manager template uses the status code to determine how to proceed with the deployment.

The following two files are used for deploying the Azure Function. You don't need to download these files to go through the tutorial.

- A Resource Manager template located at https://armtutorials.blob.core.windows.net/admtutorial/deploy_hc_azure_function.json. You deploy this template to create an Azure Function.
- A zip file of the Azure Function source code, https://armtutorials.blob.core.windows.net/admtutorial/ADMHCFunction0417.zip. This zip called is called by the Resource Manager template.

To deploy the Azure function, select **Try it** to open the Azure Cloud shell, and then paste the following script into the shell window. To paste the code, right-click the shell window and then select **Paste**.

> **IMPORTANT**
>
> **projectName** in the PowerShell script is used to generate names for the Azure services that are deployed in this tutorial. Different Azure services have different requirements on the names. To ensure the deployment is successful, choose a name with less than 12 characters with only lower case letters and numbers. Save a copy of the project name. You use the same projectName through the tutorial.

```
$projectName = Read-Host -Prompt "Enter a project name that is used to generate Azure resource names"
$location = Read-Host -Prompt "Enter the location (i.e. centralus)"
$resourceGroupName = "${projectName}rg"

New-AzResourceGroup -Name $resourceGroupName -Location $location
New-AzResourceGroupDeployment -ResourceGroupName $resourceGroupName -TemplateUri
"https://armtutorials.blob.core.windows.net/admtutorial/deploy_hc_azure_function.json" -projectName
$projectName
```

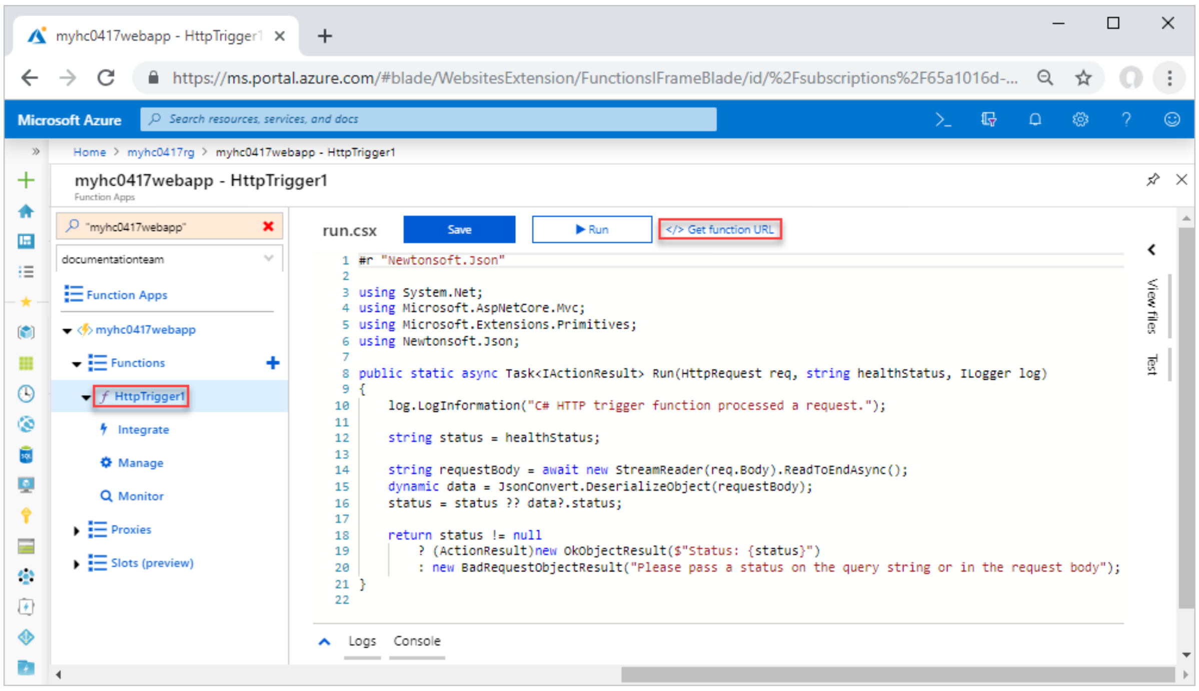
To verify and test the Azure function:

1. Open the Azure portal.

2. Open the resource group. The default name is the project name with **rg** appended.

3. Select the app service from the resource group. The default name of the app service is the project name with **webapp** appended.

4. Expand **Functions**, and then select **HttpTrigger1**.


myhc0417webapp - HttpTrigger1
https://ms.portal.azure.com/#blade/WebsitesExtension/FunctionsIFrameBlade/id/%2Fsubscriptions%2F65a1016d-...
Microsoft Azure
Search resources, services, and docs
Home > myhc0417rg > myhc0417webapp - HttpTrigger1
myhc0417webapp - HttpTrigger1
Function Apps
"myhc0417webapp"
documentationteam
Function Apps
myhc0417webapp
Functions
HttpTrigger1
Integrate
Manage
Monitor
Proxies
Slots (preview)
run.csx
Save
Run
</> Get function URL
View files
Test
Logs
Console

```
#r "Newtonsoft.Json"

using System.Net;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Primitives;
using Newtonsoft.Json;

public static async Task<IActionResult> Run(HttpRequest req, string healthStatus, ILogger log)
{
    log.LogInformation("C# HTTP trigger function processed a request.");

    string status = healthStatus;

    string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
    dynamic data = JsonConvert.DeserializeObject(requestBody);
    status = status ?? data?.status;

    return status != null
        ? (ActionResult)new OkObjectResult($"Status: {status}")
        : new BadRequestObjectResult("Please pass a status on the query string or in the request body");
}
```

5. Select **</> Get function URL**.

6. Select **Copy** to copy the URL to the clipboard. The URL is similar to:

```
https://myhc0417webapp.azurewebsites.net/api/healthStatus/{healthStatus}?
code=hc4Y1wY4AqsskAkVw6WLAN1A4E6aB0h3MbQ3YJRF3XtXgHvooaG0aw==
```

Replace `{healthStatus}` in the URL with a status code. In this tutorial, use **unhealthy** to test the unhealthy scenario, and use either **healthy** or **warning** to test the healthy scenario. Create two URLs, one with the unhealthy status, and the other with healthy status. For examples:

```
https://myhc0417webapp.azurewebsites.net/api/healthStatus/unhealthy?
code=hc4Y1wY4AqsskAkVw6WLAN1A4E6aB0h3MbQ3YJRF3XtXgHvooaG0aw==
https://myhc0417webapp.azurewebsites.net/api/healthStatus/healthy?
code=hc4Y1wY4AqsskAkVw6WLAN1A4E6aB0h3MbQ3YJRF3XtXgHvooaG0aw==
```

You need both URLs to completed this tutorial.

7. To test the health monitoring simulator, open the URLs that you created in the last step. The results for the unhealthy status shall be similar to:

```
Status: unhealthy
```

Revise the rollout template

The purpose of this section is to show you how to include a health check step in the rollout template. You don't have to create your own CreateADMRollout.json file to complete this tutorial. The revised rollout template is shared in a storage account that is used in the subsequent sections.

1. Open **CreateADMRollout.json**. This JSON file is a part of the download. See Prerequisites.

2. Add two more parameters:

```
"healthCheckUrl": {
    "type": "string",
    "metadata": {
        "description": "Specifies the health check URL."
    }
},
"healthCheckAuthAPIKey": {
    "type": "string",
    "metadata": {
        "description": "Specifies the health check Azure Function function authorization key."
    }
}
```

3. Replace the wait step resource definition with a health check step resource definition:

```
{
  "type": "Microsoft.DeploymentManager/steps",
  "apiVersion": "2018-09-01-preview",
  "name": "healthCheckStep",
   "location": "[parameters('azureResourceLocation')]",
  "properties": {
    "stepType": "healthCheck",
    "attributes": {
      "waitDuration": "PT0M",
      "maxElasticDuration": "PT0M",
      "healthyStateDuration": "PT1M",
      "type": "REST",
      "properties": {
        "healthChecks": [
          {
            "name": "appHealth",
            "request": {
              "method": "GET",
              "uri": "[parameters('healthCheckUrl')]",
              "authentication": {
                "type": "ApiKey",
                "name": "code",
                "in": "Query",
                "value": "[parameters('healthCheckAuthAPIKey')]"
              }
            },
            "response": {
              "successStatusCodes": [
                "200"
              ],
              "regex": {
                "matches": [
                  "Status: healthy",
                  "Status: warning"
                ],
                "matchQuantifier": "Any"
              }
            }
          }
        ]
      }
    }
  }
},
```

Based on the definition, the rollout proceeds if the health status is either *healthy* or *warning*.

4. Update the **dependsON** of the rollout definition to include the newly defined health check step:

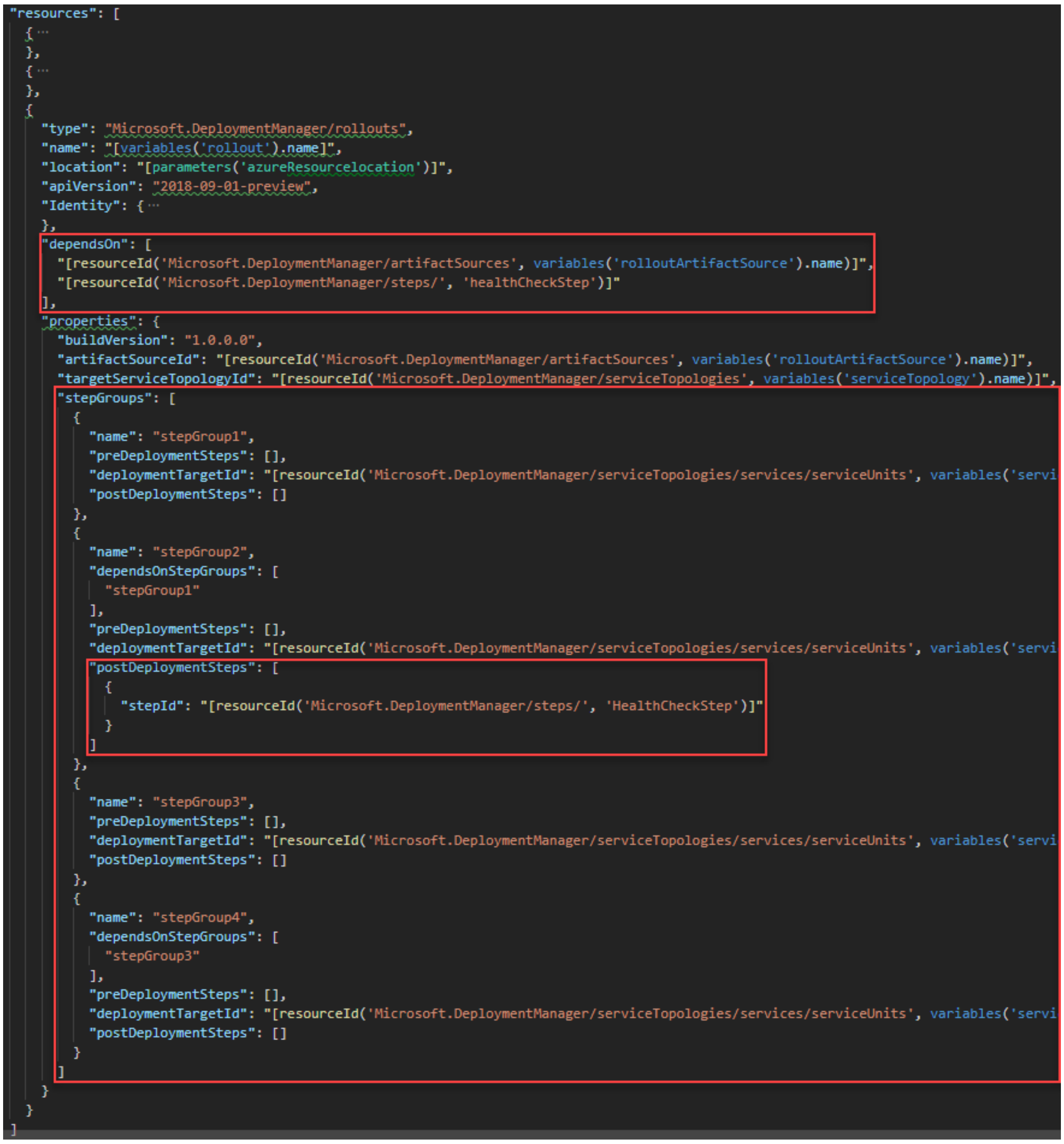
```
"dependsOn": [
    "[resourceId('Microsoft.DeploymentManager/artifactSources',
variables('rolloutArtifactSource').name)]",
    "[resourceId('Microsoft.DeploymentManager/steps/', 'healthCheckStep')]"
],
```

5. Update **stepGroups** to include the health check step. The **healthCheckStep** is called in **postDeploymentSteps** of **stepGroup2**. **stepGroup3** and **stepGroup4** are only deployed if the healthy status is either *healthy* or *warning*.

```
"stepGroups": [
    {
        "name": "stepGroup1",
        "preDeploymentSteps": [],
        "deploymentTargetId": "
[resourceId('Microsoft.DeploymentManager/serviceTopologies/services/serviceUnits',
variables('serviceTopology').name, variables('serviceTopology').serviceWUS.name,
variables('serviceTopology').serviceWUS.serviceUnit2.name)]",
        "postDeploymentSteps": []
    },
    {
        "name": "stepGroup2",
        "dependsOnStepGroups": ["stepGroup1"],
        "preDeploymentSteps": [],
        "deploymentTargetId": "
[resourceId('Microsoft.DeploymentManager/serviceTopologies/services/serviceUnits',
variables('serviceTopology').name, variables('serviceTopology').serviceWUS.name,
variables('serviceTopology').serviceWUS.serviceUnit1.name)]",
        "postDeploymentSteps": [
            {
                "stepId": "[resourceId('Microsoft.DeploymentManager/steps/', 'healthCheckStep')]"
            }
        ]
    },
    {
        "name": "stepGroup3",
        "dependsOnStepGroups": ["stepGroup2"],
        "preDeploymentSteps": [],
        "deploymentTargetId": "
[resourceId('Microsoft.DeploymentManager/serviceTopologies/services/serviceUnits',
variables('serviceTopology').name, variables('serviceTopology').serviceEUS.name,
variables('serviceTopology').serviceEUS.serviceUnit2.name)]",
        "postDeploymentSteps": []
    },
    {
        "name": "stepGroup4",
        "dependsOnStepGroups": ["stepGroup3"],
        "preDeploymentSteps": [],
        "deploymentTargetId": "
[resourceId('Microsoft.DeploymentManager/serviceTopologies/services/serviceUnits',
variables('serviceTopology').name, variables('serviceTopology').serviceEUS.name,
variables('serviceTopology').serviceEUS.serviceUnit1.name)]",
        "postDeploymentSteps": []
    }
]
```

If you compare the **stepGroup3** section before and after it is revised, this section now depends on **stepGroup2**. This is necessary when **stepGroup3** and the subsequent step groups depend on the results of health monitoring.

The following screenshot illustrates the areas modified, and how the health check step is used:



```
"resources": [
  {…
  },
  {…
  },
  {
    "type": "Microsoft.DeploymentManager/rollouts",
    "name": "[variables('rollout').name]",
    "location": "[parameters('azureResourcelocation')]",
    "apiVersion": "2018-09-01-preview",
    "Identity": {…
    },
    "dependsOn": [
      "[resourceId('Microsoft.DeploymentManager/artifactSources', variables('rolloutArtifactSource').name)]",
      "[resourceId('Microsoft.DeploymentManager/steps/', 'healthCheckStep')]"
    ],
    "properties": {
      "buildVersion": "1.0.0.0",
      "artifactSourceId": "[resourceId('Microsoft.DeploymentManager/artifactSources', variables('rolloutArtifactSource').name)]",
      "targetServiceTopologyId": "[resourceId('Microsoft.DeploymentManager/serviceTopologies', variables('serviceTopology').name)]",
      "stepGroups": [
        {
          "name": "stepGroup1",
          "preDeploymentSteps": [],
          "deploymentTargetId": "[resourceId('Microsoft.DeploymentManager/serviceTopologies/services/serviceUnits', variables('servi
          "postDeploymentSteps": []
        },
        {
          "name": "stepGroup2",
          "dependsOnStepGroups": [
            "stepGroup1"
          ],
          "preDeploymentSteps": [],
          "deploymentTargetId": "[resourceId('Microsoft.DeploymentManager/serviceTopologies/services/serviceUnits', variables('servi
          "postDeploymentSteps": [
            {
              "stepId": "[resourceId('Microsoft.DeploymentManager/steps/', 'HealthCheckStep')]"
            }
          ]
        },
        {
          "name": "stepGroup3",
          "preDeploymentSteps": [],
          "deploymentTargetId": "[resourceId('Microsoft.DeploymentManager/serviceTopologies/services/serviceUnits', variables('servi
          "postDeploymentSteps": []
        },
        {
          "name": "stepGroup4",
          "dependsOnStepGroups": [
            "stepGroup3"
          ],
          "preDeploymentSteps": [],
          "deploymentTargetId": "[resourceId('Microsoft.DeploymentManager/serviceTopologies/services/serviceUnits', variables('servi
          "postDeploymentSteps": []
        }
      ]
    }
  }
]
```

Deploy the topology

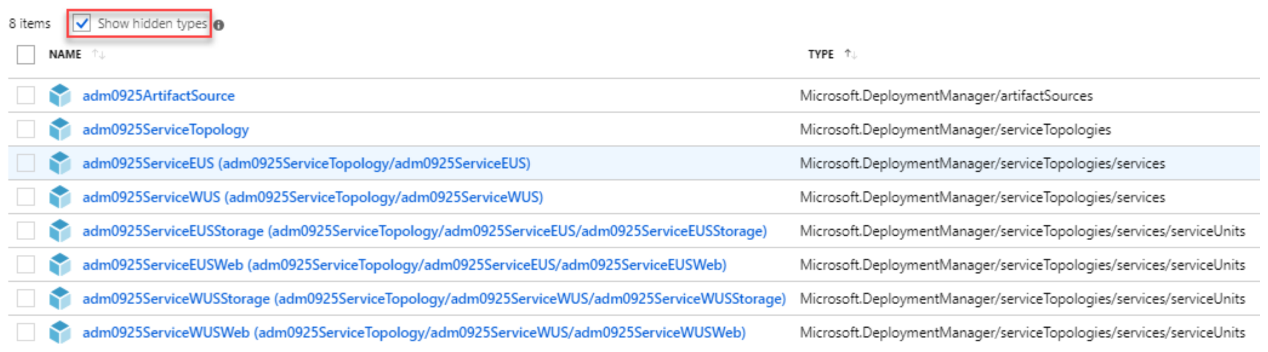
To simplify the tutorial, the topology template and artifacts are shared at the following locations so that you don't need to prepare your own copy. If you want to use your own, follow the instructions in Tutorial: Use Azure Deployment Manager with Resource Manager templates.

- Topology template: \https://armtutorials.blob.core.windows.net/admtutorial/ADMTemplates/CreateADMServiceTopology.json
- Artifacts store: \https://armtutorials.blob.core.windows.net/admtutorial/ArtifactStore

To deploy the topology, select **Try it** to open the Cloud shell, and then paste the PowerShell script.

```
$projectName = Read-Host -Prompt "Enter the same project name used earlier in this tutorial"
$location = Read-Host -Prompt "Enter the location (i.e. centralus)"
$resourceGroupName = "${projectName}rg"
$artifactLocation = "https://armtutorials.blob.core.windows.net/admtutorial/ArtifactStore?st=2019-05-
06T03%3A57%3A31Z&se=2020-05-07T03%3A57%3A00Z&sp=rl&sv=2018-03-
28&sr=c&sig=gOh%2Bkhi693rmdxiZFQ9xbKZMU1kbLJDqXw7EP4TaGlI%3D" | ConvertTo-SecureString -AsPlainText -Force

# Create the service topology
New-AzResourceGroupDeployment `
    -ResourceGroupName $resourceGroupName `
    -TemplateUri
"https://armtutorials.blob.core.windows.net/admtutorial/ADMTemplatesHC/CreateADMServiceTopology.json" `
    -namePrefix $projectName `
    -azureResourceLocation $location `
    -artifactSourceSASLocation $artifactLocation
```

Verify the service topology and the underlined resources have been created successfully using the Azure portal:


8 items ✓ Show hidden types

NAME	TYPE
adm0925ArtifactSource	Microsoft.DeploymentManager/artifactSources
adm0925ServiceTopology	Microsoft.DeploymentManager/serviceTopologies
adm0925ServiceEUS (adm0925ServiceTopology/adm0925ServiceEUS)	Microsoft.DeploymentManager/serviceTopologies/services
adm0925ServiceWUS (adm0925ServiceTopology/adm0925ServiceWUS)	Microsoft.DeploymentManager/serviceTopologies/services
adm0925ServiceEUSStorage (adm0925ServiceTopology/adm0925ServiceEUS/adm0925ServiceEUSStorage)	Microsoft.DeploymentManager/serviceTopologies/services/serviceUnits
adm0925ServiceEUSWeb (adm0925ServiceTopology/adm0925ServiceEUS/adm0925ServiceEUSWeb)	Microsoft.DeploymentManager/serviceTopologies/services/serviceUnits
adm0925ServiceWUSStorage (adm0925ServiceTopology/adm0925ServiceWUS/adm0925ServiceWUSStorage)	Microsoft.DeploymentManager/serviceTopologies/services/serviceUnits
adm0925ServiceWUSWeb (adm0925ServiceTopology/adm0925ServiceWUS/adm0925ServiceWUSWeb)	Microsoft.DeploymentManager/serviceTopologies/services/serviceUnits

Show hidden types must be selected to see the resources.

Deploy the rollout with the unhealthy status

To simplify the tutorial, the revised rollout template is shared at the following locations so that you don't need to prepare your own copy. If you want to use your own, follow the instructions in Tutorial: Use Azure Deployment Manager with Resource Manager templates.

- Topology template: \https://armtutorials.blob.core.windows.net/admtutorial/ADMTemplatesHC/CreateADMRollout.json
- Artifacts store: \https://armtutorials.blob.core.windows.net/admtutorial/ArtifactStore

Use the unhealthy status URL you created in Create a health check service simulator. For **managedIdentityID**, see Create the user-assigned managed identity.

```
$projectName = Read-Host -Prompt "Enter the same project name used earlier in this tutorial"
$location = Read-Host -Prompt "Enter the location (i.e. centralus)"
$managedIdentityID = Read-Host -Prompt "Enter a user-assigned managed identity"
$healthCheckUrl = Read-Host -Prompt "Enter the health check Azure function URL"
$healthCheckAuthAPIKey = $healthCheckUrl.Substring($healthCheckUrl.IndexOf("?code=")+6,
$healthCheckUrl.Length-$healthCheckUrl.IndexOf("?code=")-6)
$healthCheckUrl = $healthCheckUrl.Substring(0, $healthCheckUrl.IndexOf("?"))

$resourceGroupName = "${projectName}rg"
$artifactLocation = "https://armtutorials.blob.core.windows.net/admtutorial/ArtifactStore?st=2019-05-
06T03%3A57%3A31Z&se=2020-05-07T03%3A57%3A00Z&sp=rl&sv=2018-03-
28&sr=c&sig=gOh%2Bkhi693rmdxiZFQ9xbKZMU1kbLJDqXw7EP4TaGlI%3D" | ConvertTo-SecureString -AsPlainText -Force

# Create the rollout
New-AzResourceGroupDeployment `
    -ResourceGroupName $resourceGroupName `
    -TemplateUri "https://armtutorials.blob.core.windows.net/admtutorial/ADMTemplatesHC/CreateADMRollout.json"
`
    -namePrefix $projectName `
    -azureResourceLocation $location `
    -artifactSourceSASLocation $artifactLocation `
    -managedIdentityID $managedIdentityID `
    -healthCheckUrl $healthCheckUrl `
    -healthCheckAuthAPIKey $healthCheckAuthAPIKey
```

NOTE

`New-AzResourceGroupDeployment` is an asynchronous call. The success message only means the deployment has successfully begun. To verify the deployment, use `Get-AZDeploymentManagerRollout`. See the next procedure.

To check the rollout progress using the following PowerShell script:

```
$projectName = Read-Host -Prompt "Enter the same project name used earlier in this tutorial"
$resourceGroupName = "${projectName}rg"
$rolloutName = "${projectName}Rollout"

# Get the rollout status
Get-AzDeploymentManagerRollout `
    -ResourceGroupName $resourceGroupName `
    -Name $rolloutName `
    -Verbose
```

The following sample output shows the deployment failed due to the unhealthy status:

```
Service: myhc0417ServiceWUSrg
    TargetLocation: WestUS
    TargetSubscriptionId: <Subscription ID>

    ServiceUnit: myhc0417ServiceWUSWeb
        TargetResourceGroup: myhc0417ServiceWUSrg

        Step: RestHealthCheck/healthCheckStep.PostDeploy
            Status: Failed
            StepGroup: stepGroup2
            Operation Info:
                Start Time: 05/06/2019 17:58:31
                End Time: 05/06/2019 17:58:32
                Total Duration: 00:00:01
                Error:
                    Code: ResourceReportedUnhealthy
                    Message: Health checks failed as the following resources were unhealthy: '05/06/2019
17:58:32 UTC: Health check 'appHealth' failed with the following errors: Response from endpoint
'https://myhc0417webapp.azurewebsites.net/api/healthStatus/unhealthy' does not match the regex pattern(s):
'Status: healthy, Status: warning.'. Response content: "Status: unhealthy"..'.
Get-AzDeploymentManagerRollout :
Service: myhc0417ServiceWUSrg
    ServiceUnit: myhc0417ServiceWUSWeb
        Step: RestHealthCheck/healthCheckStep.PostDeploy
            Status: Failed
            StepGroup: stepGroup2
            Operation Info:
                Start Time: 05/06/2019 17:58:31
                End Time: 05/06/2019 17:58:32
                Total Duration: 00:00:01
                Error:
                    Code: ResourceReportedUnhealthy
                    Message: Health checks failed as the following resources were unhealthy: '05/06/2019
17:58:32 UTC: Health check 'appHealth' failed with the following errors: Response from endpoint
'https://myhc0417webapp.azurewebsites.net/api/healthStatus/unhealthy' does not match the regex pattern(s):
'Status: healthy, Status: warning.'. Response content: "Status: unhealthy"..'.
At line:1 char:1
+ Get-AzDeploymentManagerRollout `
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
+ CategoryInfo          : NotSpecified: (:) [Get-AzDeploymentManagerRollout], Exception
+ FullyQualifiedErrorId : RolloutFailed,Microsoft.Azure.Commands.DeploymentManager.Commands.GetRollout


ResourceGroupName       : myhc0417rg
BuildVersion            : 1.0.0.0
ArtifactSourceId        : /subscriptions/<Subscription ID>/resourceGroups/myhc0417rg/providers/Mi
                          crosoft.DeploymentManager/artifactSources/myhc0417ArtifactSourceRollout
TargetServiceTopologyId : /subscriptions/<Subscription ID>/resourceGroups/myhc0417rg/providers/Mi
                          crosoft.DeploymentManager/serviceTopologies/myhc0417ServiceTopology
Status                  : Failed
TotalRetryAttempts      : 0
Identity                : Microsoft.Azure.Commands.DeploymentManager.Models.PSIdentity
OperationInfo           : Microsoft.Azure.Commands.DeploymentManager.Models.PSRolloutOperationInfo
Services                : {myhc0417ServiceWUS, myhc0417ServiceWUSrg}
Name                    : myhc0417Rollout
Type                    : Microsoft.DeploymentManager/rollouts
Location                : centralus
Id                      : /subscriptions/<Subscription ID>/resourcegroups/myhc0417rg/providers/Mi
                          crosoft.DeploymentManager/rollouts/myhc0417Rollout
Tags                    :
```

After the rollout is completed, you shall see one additional resource group created for West US.

Deploy the rollout with the healthy status

Repeat this section to redeploy the rollout with the healthy status URL. After the rollout is completed, you shall see one more resource group created for East US.

Verify the deployment

1. Open the Azure portal.
2. Browse to the newly create web applications under the new resource groups created by the rollout deployment.
3. Open the web application in a web browser. Verify the location and the version on the index.html file.

Clean up resources

When the Azure resources are no longer needed, clean up the resources you deployed by deleting the resource group.

1. From the Azure portal, select **Resource group** from the left menu.
2. Use the **Filter by name** field to narrow down the resource groups created in this tutorial. There shall be 3-4:
   - **<namePrefix>rg**: contains the Deployment Manager resources.
   - **<namePrefix>ServiceWUSrg**: contains the resources defined by ServiceWUS.
   - **<namePrefix>ServiceEUSrg**: contains the resources defined by ServiceEUS.
   - The resource group for the user-defined managed identity.
3. Select the resource group name.
4. Select **Delete resource group** from the top menu.
5. Repeat the last two steps to delete other resource groups created by this tutorial.

Next steps

In this tutorial, you learned how to use the health check feature of Azure Deployment Manager. To learn more, see Azure Resource Manager documentation.

Tutorial: Troubleshoot Resource Manager template deployments

2/14/2019 • 3 minutes to read • Edit Online

Learn how to troubleshoot Resource Manager template deployment errors. In this tutorial, you set up two errors in a template, and learn how to use the activity logs and deployment history to resolve the issues.

There are two types of errors that are related to template deployment:

- **Validation errors** arise from scenarios that can be determined before deployment. They include syntax errors in your template, or trying to deploy resources that would exceed your subscription quotas.
- **Deployment errors** arise from conditions that occur during the deployment process. They include trying to access a resource that is being deployed in parallel.

Both types of errors return an error code that you use to troubleshoot the deployment. Both types of errors appear in the activity log. However, validation errors don't appear in your deployment history because the deployment never started.

This tutorial covers the following tasks:

- Create a problematic template
- Troubleshoot validation errors
- Troubleshoot deployment errors
- Clean up resources

If you don't have an Azure subscription, create a free account before you begin.

> **NOTE**
>
> This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

Prerequisites

To complete this article, you need:

- Visual Studio Code with Resource Manager Tools extension.

Create a problematic template

Open a template called Create a standard storage account from Azure QuickStart Templates, and setup two template issues.

1. From Visual Studio Code, select **File**>**Open File**.
2. In **File name**, paste the following URL:

```
https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-storage-account-
create/azuredeploy.json
```

3. Select **Open** to open the file.
4. Change the **apiVersion** line to the following line:

   ```
   "apiVersion1": "2018-07-02",
   ```

   - **apiVersion1** is invalid element name. It is a validation error.
   - The API version shall be "2018-07-01". It is a deployment error.
5. Select **File**>**Save As** to save the file as **azuredeploy.json** to your local computer.

Troubleshoot the validation error

Refer to the Deploy the template section to deploy the template.

You shall get an error from the shell similar to:

```
New-AzResourceGroupDeployment : 4:29:24 PM - Error: Code=InvalidRequestContent; Message=The request content was
invalid and could not be deserialized: 'Could not find member 'apiVersion1' on object of type
'TemplateResource'. Path 'properties.template.resources[0].apiVersion1', line 36, position 24.'.
```

The error message indicates the problem is with **apiVersion1**.

Use Visual Studio Code to correct the problem by changing **apiVersion1** to **apiVersion**, and then save the template.

Troubleshoot the deployment error

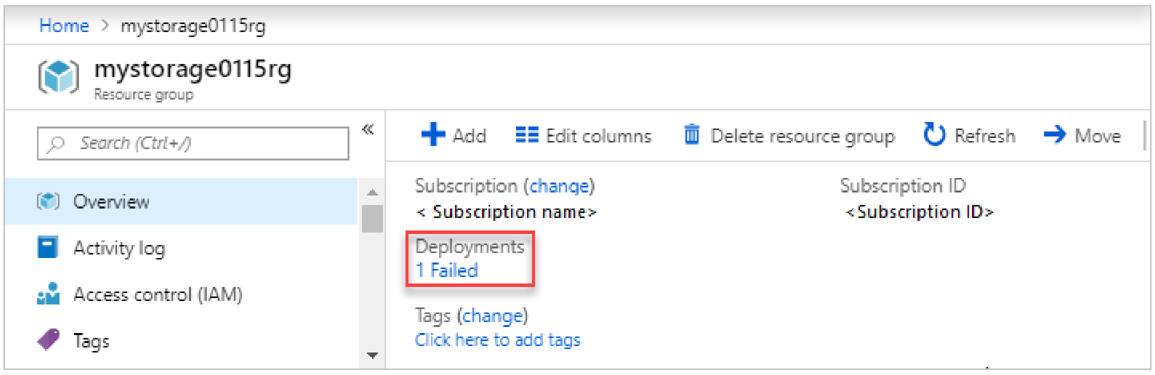
Refer to the Deploy the template section to deploy the template.

You shall get an error from the shell similar to:

```
New-AzResourceGroupDeployment : 4:48:50 PM - Resource Microsoft.Storage/storageAccounts 'storeqii7x2rce77dc'
failed with message '{
  "error": {
    "code": "NoRegisteredProviderFound",
    "message": "No registered resource provider found for location 'centralus' and API version '2018-07-02' for
type 'storageAccounts'. The supported api-versions are '2018-07-01, 2018-03-01-preview, 2018-02-01, 2017-10-01,
2017-06-01, 2016-12-01, 2016-05-01, 2016-01-01, 2015-06-15, 2015-05-01-preview'. The supported locations are
'eastus, eastus2, westus, westeurope, eastasia, southeastasia, japaneast, japanwest, northcentralus,
southcentralus, centralus, northeurope, brazilsouth, australiaeast, australiasoutheast, southindia,
centralindia, westindia, canadaeast, canadacentral, westus2, westcentralus, uksouth, ukwest, koreacentral,
koreasouth, francecentral'."
  }
}'
```

The deployment error can be found from the Azure portal using the following procedure:

1. Sign in to the Azure portal.
2. Open the resource group by selecting **Resource groups** and then the resource group name. You shall see **1 Failed** under **Deployment**.



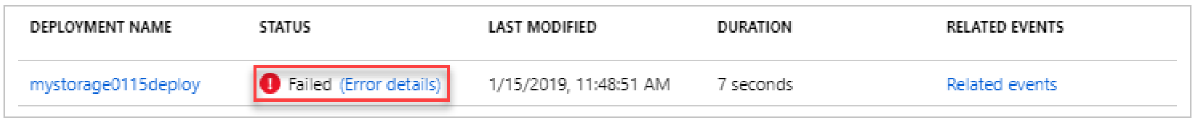
Subscription (change)
< Subscription name>

Subscription ID
<Subscription ID>

Deployments
1 Failed

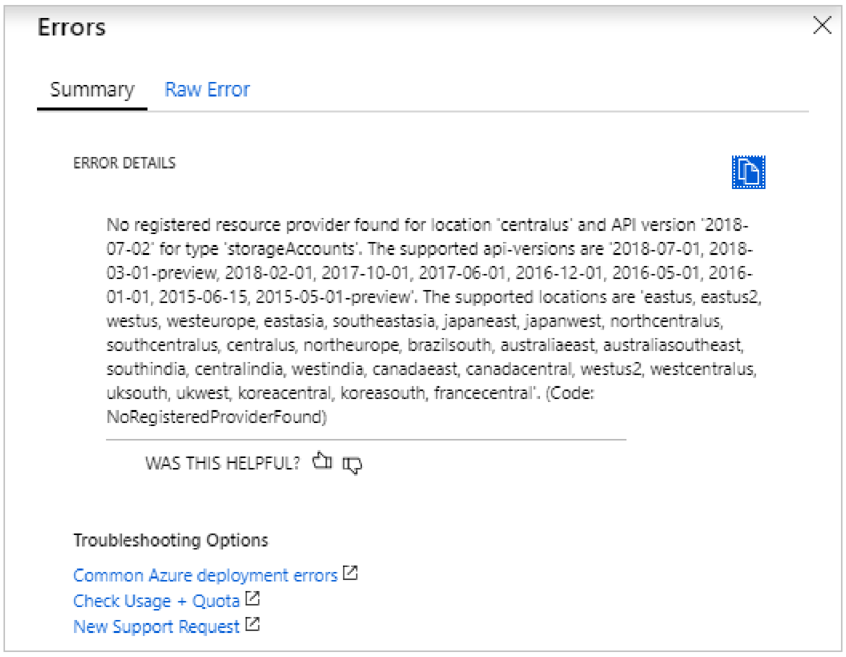
Tags (change)
Click here to add tags

3. Select **Error details**.



DEPLOYMENT NAME	STATUS	LAST MODIFIED	DURATION	RELATED EVENTS
mystorage0115deploy	Failed (Error details)	1/15/2019, 11:48:51 AM	7 seconds	Related events

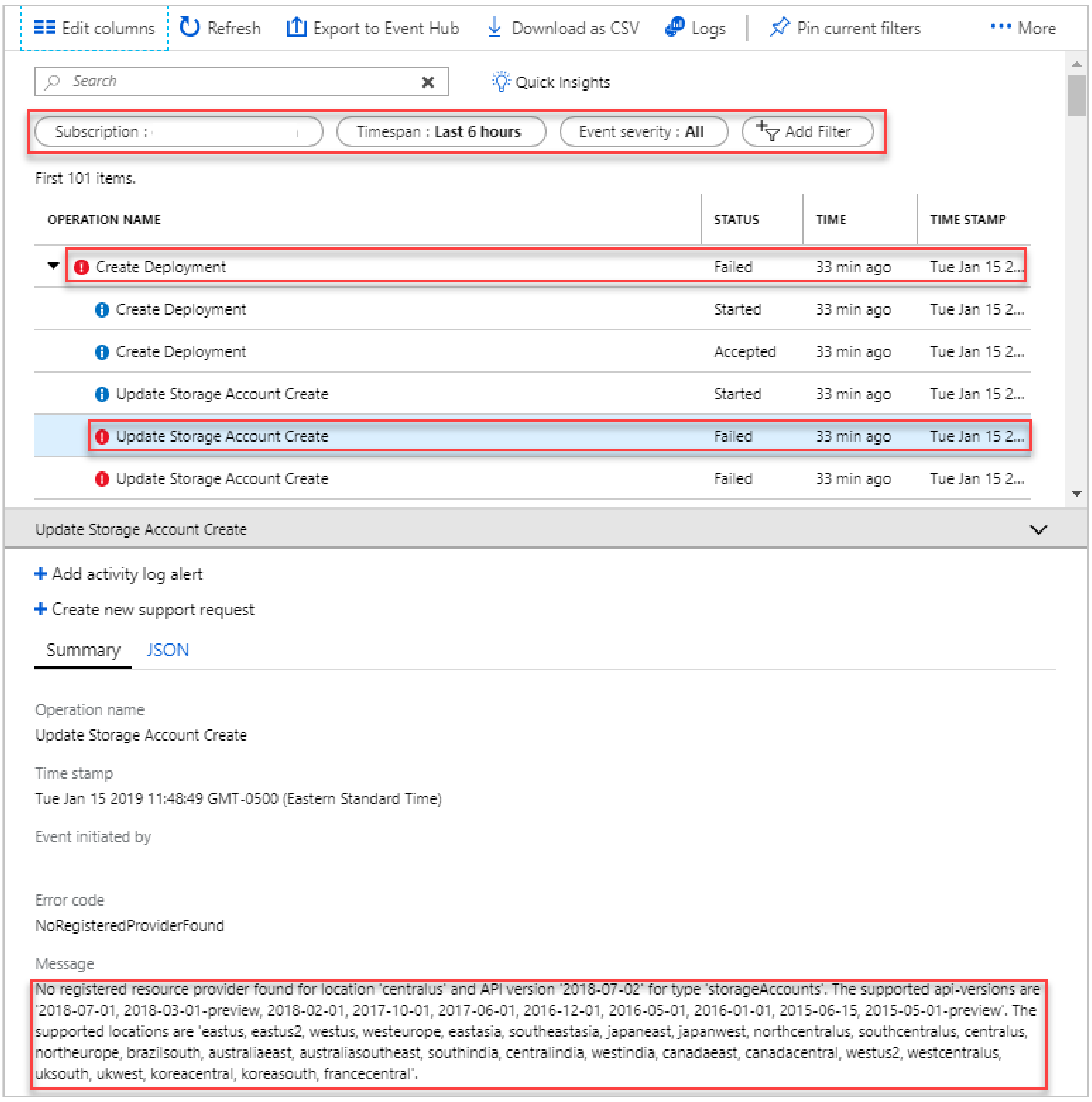
The error message is the same as the one shown earlier:



No registered resource provider found for location 'centralus' and API version '2018-07-02' for type 'storageAccounts'. The supported api-versions are '2018-07-01, 2018-03-01-preview, 2018-02-01, 2017-10-01, 2017-06-01, 2016-12-01, 2016-05-01, 2016-01-01, 2015-06-15, 2015-05-01-preview'. The supported locations are 'eastus, eastus2, westus, westeurope, eastasia, southeastasia, japaneast, japanwest, northcentralus, southcentralus, centralus, northeurope, brazilsouth, australiaeast, australiasoutheast, southindia, centralindia, westindia, canadaeast, canadacentral, westus2, westcentralus, uksouth, ukwest, koreacentral, koreasouth, francecentral'. (Code: NoRegisteredProviderFound)

You can also find the error from the activity logs:

1. Sign in to the Azure portal.
2. Select **Monitor** > **Activity log**.
3. Use the filters to find the log.



First 101 items.

OPERATION NAME	STATUS	TIME	TIME STAMP
Create Deployment	Failed	33 min ago	Tue Jan 15 2...
Create Deployment	Started	33 min ago	Tue Jan 15 2...
Create Deployment	Accepted	33 min ago	Tue Jan 15 2...
Update Storage Account Create	Started	33 min ago	Tue Jan 15 2...
Update Storage Account Create	Failed	33 min ago	Tue Jan 15 2...
Update Storage Account Create	Failed	33 min ago	Tue Jan 15 2...

Update Storage Account Create

Operation name
Update Storage Account Create

Time stamp
Tue Jan 15 2019 11:48:49 GMT-0500 (Eastern Standard Time)

Event initiated by

Error code
NoRegisteredProviderFound

Message
No registered resource provider found for location 'centralus' and API version '2018-07-02' for type 'storageAccounts'. The supported api-versions are '2018-07-01, 2018-03-01-preview, 2018-02-01, 2017-10-01, 2017-06-01, 2016-12-01, 2016-05-01, 2016-01-01, 2015-06-15, 2015-05-01-preview'. The supported locations are 'eastus, eastus2, westus, westeurope, eastasia, southeastasia, japaneast, japanwest, northcentralus, southcentralus, centralus, northeurope, brazilsouth, australiaeast, australiasoutheast, southindia, centralindia, westindia, canadaeast, canadacentral, westus2, westcentralus, uksouth, ukwest, koreacentral, koreasouth, francecentral'.

Use Visual Studio Code to correct the problem, and then redeploy the template.

For a list of common errors, see Troubleshoot common Azure deployment errors with Azure Resource Manager.

Clean up resources

When the Azure resources are no longer needed, clean up the resources you deployed by deleting the resource group.

1. From the Azure portal, select **Resource group** from the left menu.
2. Enter the resource group name in the **Filter by name** field.
3. Select the resource group name. You shall see a total of six resources in the resource group.
4. Select **Delete resource group** from the top menu.

Next steps

In this tutorial, you learned how to troubleshoot Resource Manager template deployment errors. For more information, see Troubleshoot common Azure deployment errors with Azure Resource Manager.

Azure Resource Manager templates for management features

11/16/2018 • 2 minutes to read • Edit Online

The following table includes links to Azure Resource Manager templates for features provided by Resource Manager.

Role assignments	
Assign role for resource group	Assigns a built-in role to a user for an existing resource group.
Assign role for existing virtual machine	Assigns a built-in role to a user for an existing VM.
Assign role for several virtual machines	Assigns built-in roles to users for more than one virtual machine.
Assign role for Azure subscription	Assigns a role to a user for an Azure subscription.
Role definition	
Create custom role definition	Creates a new role definition in an Azure subscription.
Resource lock	
Lock resource group	Creates a resource group, and applies a **DoNotDelete** lock to the resource group. Assigns the contributor role to a user.

Understand the structure and syntax of Azure Resource Manager templates

6/18/2019 • 24 minutes to read • Edit Online

This article describes the structure of an Azure Resource Manager template. It presents the different sections of a template and the properties that are available in those sections. The template consists of JSON and expressions that you can use to construct values for your deployment.

This article is intended for users who have some familiarity with Resource Manager templates. It provides detailed information about the structure and syntax of the template. If you want an introduction to creating a template, see Create your first Azure Resource Manager template.

Template format

In its simplest structure, a template has the following elements:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "",
  "apiProfile": "",
  "parameters": {  },
  "variables": {  },
  "functions": [  ],
  "resources": [  ],
  "outputs": {  }
}
```

ELEMENT NAME	REQUIRED	DESCRIPTION
$schema	Yes	Location of the JSON schema file that describes the version of the template language. For resource group deployments, use: `https://schema.management.azure.com/schemas/` `01-01/deploymentTemplate.json#` For subscription deployments, use: `https://schema.management.azure.com/schemas/` `05-01/subscriptionDeploymentTemplate.json#`
contentVersion	Yes	Version of the template (such as 1.0.0.0). You can provide any value for this element. Use this value to document significant changes in your template. When deploying resources using the template, this value can be used to make sure that the right template is being used.

ELEMENT NAME	REQUIRED	DESCRIPTION
apiProfile	No	An API version that serves as a collection of API versions for resource types. Use this value to avoid having to specify API versions for each resource in the template. When you specify an API profile version and don't specify an API version for the resource type, Resource Manager uses the API version for that resource type that is defined in the profile. The API profile property is especially helpful when deploying a template to different environments, such as Azure Stack and global Azure. Use the API profile version to make sure your template automatically uses versions that are supported in both environments. For a list of the current API profile versions and the resources API versions defined in the profile, see API Profile. For more information, see Track versions using API profiles.
parameters	No	Values that are provided when deployment is executed to customize resource deployment.
variables	No	Values that are used as JSON fragments in the template to simplify template language expressions.
functions	No	User-defined functions that are available within the template.
resources	Yes	Resource types that are deployed or updated in a resource group or subscription.
outputs	No	Values that are returned after deployment.

Each element has properties you can set. This article describes the sections of the template in greater detail.

Syntax

The basic syntax of the template is JSON. However, you can use expressions to extend the JSON values available within the template. Expressions start and end with brackets: `[` and `]`, respectively. The value of the expression is evaluated when the template is deployed. An expression can return a string, integer, boolean, array, or object. The following example shows an expression in the default value of a parameter:

```
"parameters": {
  "location": {
    "type": "string",
    "defaultValue": "[resourceGroup().location]"
  }
},
```

Within the expression, the syntax `resourceGroup()` calls one of the functions that Resource Manager provides for use within a template. Just like in JavaScript, function calls are formatted as `functionName(arg1,arg2,arg3)`. The syntax `.location` retrieves one property from the object returned by that function.

Template functions and their parameters are case-insensitive. For example, Resource Manager resolves **variables('var1')** and **VARIABLES('VAR1')** as the same. When evaluated, unless the function expressly modifies case (such as toUpper or toLower), the function preserves the case. Certain resource types may have case requirements irrespective of how functions are evaluated.

To have a literal string start with a left bracket `[` and end with a right bracket `]`, but not have it interpreted as an expression, add an extra bracket to start the string with `[[`. For example, the variable:

```
"demoVar1": "[[test value]"
```

Resolves to `[test value]`.

However, if the literal string doesn't end with a bracket, don't escape the first bracket. For example, the variable:

```
"demoVar2": "[test] value"
```

Resolves to `[test] value`.

To pass a string value as a parameter to a function, use single quotes.

```
"name": "[concat('storage', uniqueString(resourceGroup().id))]"
```

To escape double quotes in an expression, such as adding a JSON object in the template, use the backslash.

```
"tags": {
    "CostCenter": "{\"Dept\":\"Finance\",\"Environment\":\"Production\"}"
},
```

A template expression can't exceed 24,576 characters.

For the full list of template functions, see Azure Resource Manager template functions.

Parameters

In the parameters section of the template, you specify which values you can input when deploying the resources. These parameter values enable you to customize the deployment by providing values that are tailored for a particular environment (such as dev, test, and production). You don't have to provide parameters in your template, but without parameters your template would always deploy the same resources with the same names, locations, and properties.

You're limited to 256 parameters in a template. You can reduce the number of parameters by using objects that contain multiple properties, as shown in this article.

Available properties

The available properties for a parameter are:

```
"parameters": {
  "<parameter-name>" : {
    "type" : "<type-of-parameter-value>",
    "defaultValue": "<default-value-of-parameter>",
    "allowedValues": [ "<array-of-allowed-values>" ],
    "minValue": <minimum-value-for-int>,
    "maxValue": <maximum-value-for-int>,
    "minLength": <minimum-length-for-string-or-array>,
    "maxLength": <maximum-length-for-string-or-array-parameters>,
    "metadata": {
      "description": "<description-of-the parameter>"
    }
  }
}
```

ELEMENT NAME	REQUIRED	DESCRIPTION
parameterName	Yes	Name of the parameter. Must be a valid JavaScript identifier.
type	Yes	Type of the parameter value. The allowed types and values are **string**, **securestring**, **int**, **bool**, **object**, **secureObject**, and **array**.
defaultValue	No	Default value for the parameter, if no value is provided for the parameter.
allowedValues	No	Array of allowed values for the parameter to make sure that the right value is provided.
minValue	No	The minimum value for int type parameters, this value is inclusive.
maxValue	No	The maximum value for int type parameters, this value is inclusive.
minLength	No	The minimum length for string, secure string, and array type parameters, this value is inclusive.
maxLength	No	The maximum length for string, secure string, and array type parameters, this value is inclusive.
description	No	Description of the parameter that is displayed to users through the portal. For more information, see Comments in templates.

Define and use a parameter

The following example shows a simple parameter definition. It defines the name of the parameter, and specifies that it takes a string value. The parameter only accepts values that make sense for its intended use. It specifies a default value when no value is provided during deployment. Finally, the parameter includes a description of its use.

```
"parameters": {
  "storageSKU": {
    "type": "string",
    "allowedValues": [
      "Standard_LRS",
      "Standard_ZRS",
      "Standard_GRS",
      "Standard_RAGRS",
      "Premium_LRS"
    ],
    "defaultValue": "Standard_LRS",
    "metadata": {
      "description": "The type of replication to use for the storage account."
    }
  }
}
```

In the template, you reference the value for the parameter with the following syntax:

```
"resources": [
  {
    "type": "Microsoft.Storage/storageAccounts",
    "sku": {
      "name": "[parameters('storageSKU')]"
    },
    ...
  }
]
```

Template functions with parameters

When specifying the default value for a parameter, you can use most template functions. You can use another parameter value to build a default value. The following template demonstrates the use of functions in the default value:

```
"parameters": {
  "siteName": {
    "type": "string",
    "defaultValue": "[concat('site', uniqueString(resourceGroup().id))]",
    "metadata": {
      "description": "The site name. To use the default value, do not specify a new value."
    }
  },
  "hostingPlanName": {
    "type": "string",
    "defaultValue": "[concat(parameters('siteName'),'-plan')]",
    "metadata": {
      "description": "The host name. To use the default value, do not specify a new value."
    }
  }
}
```

You can't use the `reference` function in the parameters section. Parameters are evaluated before deployment so the `reference` function can't get the runtime state of a resource.

Objects as parameters

It can be easier to organize related values by passing them in as an object. This approach also reduces the number of parameters in the template.

Define the parameter in your template and specify a JSON object instead of a single value during deployment.

```
"parameters": {
  "VNetSettings": {
    "type": "object",
    "defaultValue": {
      "name": "VNet1",
      "location": "eastus",
      "addressPrefixes": [
        {
          "name": "firstPrefix",
          "addressPrefix": "10.0.0.0/22"
        }
      ],
      "subnets": [
        {
          "name": "firstSubnet",
          "addressPrefix": "10.0.0.0/24"
        },
        {
          "name": "secondSubnet",
          "addressPrefix": "10.0.1.0/24"
        }
      ]
    }
  }
},
```

Then, reference the subproperties of the parameter by using the dot operator.

```
"resources": [
  {
    "apiVersion": "2015-06-15",
    "type": "Microsoft.Network/virtualNetworks",
    "name": "[parameters('VNetSettings').name]",
    "location": "[parameters('VNetSettings').location]",
    "properties": {
      "addressSpace":{
        "addressPrefixes": [
          "[parameters('VNetSettings').addressPrefixes[0].addressPrefix]"
        ]
      },
      "subnets":[
        {
          "name":"[parameters('VNetSettings').subnets[0].name]",
          "properties": {
            "addressPrefix": "[parameters('VNetSettings').subnets[0].addressPrefix]"
          }
        },
        {
          "name":"[parameters('VNetSettings').subnets[1].name]",
          "properties": {
            "addressPrefix": "[parameters('VNetSettings').subnets[1].addressPrefix]"
          }
        }
      ]
    }
  }
]
```

Parameter example templates

These example templates demonstrate some scenarios for using parameters. Deploy them to test how parameters are handled in different scenarios.

TEMPLATE	DESCRIPTION
parameters with functions for default values	Demonstrates how to use template functions when defining default values for parameters. The template doesn't deploy any resources. It constructs parameter values and returns those values.
parameter object	Demonstrates using an object for a parameter. The template doesn't deploy any resources. It constructs parameter values and returns those values.

Variables

In the variables section, you construct values that can be used throughout your template. You don't need to define variables, but they often simplify your template by reducing complex expressions.

Available definitions

The following example shows the available options for defining a variable:

```
"variables": {
  "<variable-name>": "<variable-value>",
  "<variable-name>": {
    <variable-complex-type-value>
  },
  "<variable-object-name>": {
    "copy": [
      {
        "name": "<name-of-array-property>",
        "count": <number-of-iterations>,
        "input": <object-or-value-to-repeat>
      }
    ]
  },
  "copy": [
    {
      "name": "<variable-array-name>",
      "count": <number-of-iterations>,
      "input": <object-or-value-to-repeat>
    }
  ]
}
```

For information about using `copy` to create several values for a variable, see Variable iteration.

Define and use a variable

The following example shows a variable definition. It creates a string value for a storage account name. It uses several template functions to get a parameter value, and concatenates it to a unique string.

```
"variables": {
  "storageName": "[concat(toLower(parameters('storageNamePrefix')), uniqueString(resourceGroup().id))]"
},
```

You use the variable when defining the resource.

```
"resources": [
  {
    "name": "[variables('storageName')]",
    "type": "Microsoft.Storage/storageAccounts",
    ...
```

Configuration variables

You can use complex JSON types to define related values for an environment.

```
"variables": {
  "environmentSettings": {
    "test": {
      "instanceSize": "Small",
      "instanceCount": 1
    },
    "prod": {
      "instanceSize": "Large",
      "instanceCount": 4
    }
  }
},
```

In parameters, you create a value that indicates which configuration values to use.

```
"parameters": {
  "environmentName": {
    "type": "string",
    "allowedValues": [
      "test",
      "prod"
    ]
  }
},
```

You retrieve the current settings with:

```
"[variables('environmentSettings')[parameters('environmentName')].instanceSize]"
```

Variable example templates

These example templates demonstrate some scenarios for using variables. Deploy them to test how variables are handled in different scenarios.

TEMPLATE	DESCRIPTION
variable definitions	Demonstrates the different types of variables. The template doesn't deploy any resources. It constructs variable values and returns those values.
configuration variable	Demonstrates the use of a variable that defines configuration values. The template doesn't deploy any resources. It constructs variable values and returns those values.
network security rules and parameter file	Constructs an array in the correct format for assigning security rules to a network security group.

Functions

Within your template, you can create your own functions. These functions are available for use in your template. Typically, you define complicated expression that you don't want to repeat throughout your template. You create the user-defined functions from expressions and functions that are supported in templates.

When defining a user function, there are some restrictions:

- The function can't access variables.
- The function can only use parameters that are defined in the function. When you use the parameters function within a user-defined function, you're restricted to the parameters for that function.
- The function can't call other user-defined functions.
- The function can't use the reference function.
- Parameters for the function can't have default values.

Your functions require a namespace value to avoid naming conflicts with template functions. The following example shows a function that returns a storage account name:

```
"functions": [
  {
    "namespace": "contoso",
    "members": {
      "uniqueName": {
        "parameters": [
          {
            "name": "namePrefix",
            "type": "string"
          }
        ],
        "output": {
          "type": "string",
          "value": "[concat(toLower(parameters('namePrefix')), uniqueString(resourceGroup().id))]"
        }
      }
    }
  }
],
```

You call the function with:

```
"resources": [
  {
    "name": "[contoso.uniqueName(parameters('storageNamePrefix'))]",
    "type": "Microsoft.Storage/storageAccounts",
    "apiVersion": "2016-01-01",
    "sku": {
      "name": "Standard_LRS"
    },
    "kind": "Storage",
    "location": "South Central US",
    "tags": {},
    "properties": {}
  }
]
```

Resources

In the resources section, you define the resources that are deployed or updated.

Available properties

You define resources with the following structure:

```
"resources": [
  {
      "condition": "<true-to-deploy-this-resource>",
      "apiVersion": "<api-version-of-resource>",
      "type": "<resource-provider-namespace/resource-type-name>",
      "name": "<name-of-the-resource>",
      "location": "<location-of-resource>",
      "tags": {
          "<tag-name1>": "<tag-value1>",
          "<tag-name2>": "<tag-value2>"
      },
      "comments": "<your-reference-notes>",
      "copy": {
          "name": "<name-of-copy-loop>",
          "count": <number-of-iterations>,
          "mode": "<serial-or-parallel>",
          "batchSize": <number-to-deploy-serially>
      },
      "dependsOn": [
          "<array-of-related-resource-names>"
      ],
      "properties": {
          "<settings-for-the-resource>",
          "copy": [
              {
                  "name": ,
                  "count": ,
                  "input": {}
              }
          ]
      },
      "sku": {
          "name": "<sku-name>",
          "tier": "<sku-tier>",
          "size": "<sku-size>",
          "family": "<sku-family>",
          "capacity": <sku-capacity>
      },
      "kind": "<type-of-resource>",
      "plan": {
          "name": "<plan-name>",
          "promotionCode": "<plan-promotion-code>",
          "publisher": "<plan-publisher>",
          "product": "<plan-product>",
          "version": "<plan-version>"
      },
      "resources": [
          "<array-of-child-resources>"
      ]
  }
]
```

ELEMENT NAME	REQUIRED	DESCRIPTION
condition	No	Boolean value that indicates whether the resource will be provisioned during this deployment. When `true`, the resource is created during deployment. When `false`, the resource is skipped for this deployment. See condition.
apiVersion	Yes	Version of the REST API to use for creating the resource. To determine available values, see template reference.

ELEMENT NAME	REQUIRED	DESCRIPTION
type	Yes	Type of the resource. This value is a combination of the namespace of the resource provider and the resource type (such as **Microsoft.Storage/storageAccounts**). To determine available values, see template reference. For a child resource, the format of the type depends on whether it's nested within the parent resource or defined outside of the parent resource. See child resources.
name	Yes	Name of the resource. The name must follow URI component restrictions defined in RFC3986. In addition, Azure services that expose the resource name to outside parties validate the name to make sure it isn't an attempt to spoof another identity. For a child resource, the format of the name depends on whether it's nested within the parent resource or defined outside of the parent resource. See child resources.
location	Varies	Supported geo-locations of the provided resource. You can select any of the available locations, but typically it makes sense to pick one that is close to your users. Usually, it also makes sense to place resources that interact with each other in the same region. Most resource types require a location, but some types (such as a role assignment) don't require a location.
tags	No	Tags that are associated with the resource. Apply tags to logically organize resources across your subscription.
comments	No	Your notes for documenting the resources in your template. For more information, see Comments in templates.
copy	No	If more than one instance is needed, the number of resources to create. The default mode is parallel. Specify serial mode when you don't want all or the resources to deploy at the same time. For more information, see Create several instances of resources in Azure Resource Manager.

ELEMENT NAME	REQUIRED	DESCRIPTION
dependsOn	No	Resources that must be deployed before this resource is deployed. Resource Manager evaluates the dependencies between resources and deploys them in the correct order. When resources aren't dependent on each other, they're deployed in parallel. The value can be a comma-separated list of a resource names or resource unique identifiers. Only list resources that are deployed in this template. Resources that aren't defined in this template must already exist. Avoid adding unnecessary dependencies as they can slow your deployment and create circular dependencies. For guidance on setting dependencies, see Defining dependencies in Azure Resource Manager templates.
properties	No	Resource-specific configuration settings. The values for the properties are the same as the values you provide in the request body for the REST API operation (PUT method) to create the resource. You can also specify a copy array to create several instances of a property. To determine available values, see template reference.
sku	No	Some resources allow values that define the SKU to deploy. For example, you can specify the type of redundancy for a storage account.
kind	No	Some resources allow a value that defines the type of resource you deploy. For example, you can specify the type of Cosmos DB to create.
plan	No	Some resources allow values that define the plan to deploy. For example, you can specify the marketplace image for a virtual machine.
resources	No	Child resources that depend on the resource being defined. Only provide resource types that are permitted by the schema of the parent resource. Dependency on the parent resource isn't implied. You must explicitly define that dependency. See child resources.

Condition

When you must decide during deployment whether to create a resource, use the `condition` element. The value for this element resolves to true or false. When the value is true, the resource is created. When the value is false, the resource isn't created. The value can only be applied to the whole resource.

Typically, you use this value when you want to create a new resource or use an existing one. For example, to specify whether a new storage account is deployed or an existing storage account is used, use:

```
{
    "condition": "[equals(parameters('newOrExisting'),'new')]",
    "type": "Microsoft.Storage/storageAccounts",
    "name": "[variables('storageAccountName')]",
    "apiVersion": "2017-06-01",
    "location": "[resourceGroup().location]",
    "sku": {
        "name": "[variables('storageAccountType')]"
    },
    "kind": "Storage",
    "properties": {}
}
```

For a complete example template that uses the `condition` element, see VM with a new or existing Virtual Network, Storage, and Public IP.

If you use a reference or list function with a resource that is conditionally deployed, the function is evaluated even if the resource isn't deployed. You get an error if the function refers to a resource that doesn't exist. Use the if function to make sure the function is only evaluated for conditions when the resource is deployed. See the if function for a sample template that uses if and reference with a conditionally deployed resource.

Resource names

Generally, you work with three types of resource names in Resource Manager:

- Resource names that must be unique.
- Resource names that aren't required to be unique, but you choose to provide a name that can help you identify the resource.
- Resource names that can be generic.

Provide a **unique resource name** for any resource type that has a data access endpoint. Some common resource types that require a unique name include:

- Azure Storage[1]
- Web Apps feature of Azure App Service
- SQL Server
- Azure Key Vault
- Azure Cache for Redis
- Azure Batch
- Azure Traffic Manager
- Azure Search
- Azure HDInsight

[1] Storage account names also must be lowercase, 24 characters or less, and not have any hyphens.

When setting the name, you can either manually create a unique name or use the uniqueString() function to generate a name. You also might want to add a prefix or suffix to the **uniqueString** result. Modifying the unique name can help you more easily identify the resource type from the name. For example, you can generate a unique name for a storage account by using the following variable:

```
"variables": {
  "storageAccountName": "[concat(uniqueString(resourceGroup().id),'storage')]"
}
```

For some resource types, you might want to provide a **name for identification**, but the name doesn't have to be unique. For these resource types, provide a name that describes it use or characteristics.

```
"parameters": {
  "vmName": {
    "type": "string",
    "defaultValue": "demoLinuxVM",
    "metadata": {
      "description": "The name of the VM to create."
    }
  }
}
```

For resource types that you mostly access through a different resource, you can use a **generic name** that is hard-coded in the template. For example, you can set a standard, generic name for firewall rules on a SQL server:

```
{
  "type": "firewallrules",
  "name": "AllowAllWindowsAzureIps",
  ...
}
```

Resource location

When deploying a template, you must provide a location for each resource. Different resource types are supported in different locations. To get the supported locations for a resource type, see Azure resource providers and types.

Use a parameter to specify the location for resources, and set the default value to `resourceGroup().location`.

The following example shows a storage account that is deployed to a location specified as a parameter:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "storageAccountType": {
      "type": "string",
      "defaultValue": "Standard_LRS",
      "allowedValues": [
        "Standard_LRS",
        "Standard_GRS",
        "Standard_ZRS",
        "Premium_LRS"
      ],
      "metadata": {
        "description": "Storage Account type"
      }
    },
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]",
      "metadata": {
        "description": "Location for all resources."
      }
    }
  },
  "variables": {
    "storageAccountName": "[concat('storage', uniquestring(resourceGroup().id))]"
  },
  "resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "name": "[variables('storageAccountName')]",
      "location": "[parameters('location')]",
      "apiVersion": "2018-07-01",
      "sku": {
        "name": "[parameters('storageAccountType')]"
      },
      "kind": "StorageV2",
      "properties": {}
    }
  ],
  "outputs": {
    "storageAccountName": {
      "type": "string",
      "value": "[variables('storageAccountName')]"
    }
  }
}
```

Child resources

Within some resource types, you can also define an array of child resources. Child resources are resources that only exist within the context of another resource. For example, a SQL database can't exist without a SQL server so the database is a child of the server. You can define the database within the definition for the server.

```
{
  "apiVersion": "2015-05-01-preview",
  "type": "Microsoft.Sql/servers",
  "name": "exampleserver",
  ...
  "resources": [
    {
      "apiVersion": "2017-10-01-preview",
      "type": "databases",
      "name": "exampledatabase",
      ...
    }
  ]
}
```

But, you don't have to define the database within the server. You can define the child resource at the top level. You might use this approach if the parent resource isn't deployed in the same template, or if want to use `copy` to create more than one child resource. With this approach, you must provide the full resource type, and include the parent resource name in the child resource name.

```
{
  "apiVersion": "2015-05-01-preview",
  "type": "Microsoft.Sql/servers",
  "name": "exampleserver",
  "resources": [
  ],
  ...
},
{
  "apiVersion": "2017-10-01-preview",
  "type": "Microsoft.Sql/servers/databases",
  "name": "exampleserver/exampledatabase",
  ...
}
```

The values you provide for type and name vary based on whether the child resource is defined within the parent resource or outside of the parent resource.

When nested in the parent resource, use:

```
"type": "{child-resource-type}",
"name": "{child-resource-name}",
```

When defined outside of the parent resource, use:

```
"type": "{resource-provider-namespace}/{parent-resource-type}/{child-resource-type}",
"name": "{parent-resource-name}/{child-resource-name}",
```

When nested, the type is set to `databases` but its full resource type is still `Microsoft.Sql/servers/databases`. You don't provide `Microsoft.Sql/servers/` because it's assumed from the parent resource type. The child resource name is set to `exampledatabase` but the full name includes the parent name. You don't provide `exampleserver` because it's assumed from the parent resource.

When constructing a fully qualified reference to a resource, the order to combine segments from the type and name isn't simply a concatenation of the two. Instead, after the namespace, use a sequence of *type/name* pairs from least specific to most specific:

```
{resource-provider-namespace}/{parent-resource-type}/{parent-resource-name}[/{child-resource-type}/{child-resource-name}]*
```

For example:

`Microsoft.Compute/virtualMachines/myVM/extensions/myExt` is correct
`Microsoft.Compute/virtualMachines/extensions/myVM/myExt` is not correct

Outputs

In the Outputs section, you specify values that are returned from deployment. Typically, you return values from resources that were deployed.

Available properties

The following example shows the structure of an output definition:

```
"outputs": {
  "<outputName>" : {
    "condition": "<boolean-value-whether-to-output-value>",
    "type" : "<type-of-output-value>",
    "value": "<output-value-expression>"
  }
}
```

ELEMENT NAME	REQUIRED	DESCRIPTION
outputName	Yes	Name of the output value. Must be a valid JavaScript identifier.
condition	No	Boolean value that indicates whether this output value is returned. When `true`, the value is included in the output for the deployment. When `false`, the output value is skipped for this deployment. When not specified, the default value is `true`.
type	Yes	Type of the output value. Output values support the same types as template input parameters. If you specify **securestring** for the output type, the value isn't displayed in the deployment history and can't be retrieved from another template. To use a secret value in more than one template, store the secret in a Key Vault and reference the secret in the parameter file. For more information, see Use Azure Key Vault to pass secure parameter value during deployment.
value	Yes	Template language expression that is evaluated and returned as output value.

Define and use output values

The following example shows how to return the resource ID for a public IP address:

```
"outputs": {
  "resourceID": {
    "type": "string",
    "value": "[resourceId('Microsoft.Network/publicIPAddresses', parameters('publicIPAddresses_name'))]"
  }
}
```

The next example shows how to conditionally return the resource ID for a public IP address based on whether a new one was deployed:

```
"outputs": {
  "resourceID": {
    "condition": "[equals(parameters('publicIpNewOrExisting'), 'new')]",
    "type": "string",
    "value": "[resourceId('Microsoft.Network/publicIPAddresses', parameters('publicIPAddresses_name'))]"
  }
}
```

For a simple example of conditional output, see conditional output template.

After the deployment, you can retrieve the value with script. For PowerShell, use:

```
(Get-AzResourceGroupDeployment -ResourceGroupName <resource-group-name> -Name <deployment-
name>).Outputs.resourceID.value
```

For Azure CLI, use:

```
az group deployment show -g <resource-group-name> -n <deployment-name> --query properties.outputs.resourceID.value
```

You can retrieve the output value from a linked template by using the reference function. To get an output value from a linked template, retrieve the property value with syntax like: `"[reference('deploymentName').outputs.propertyName.value]"`.

When getting an output property from a linked template, the property name can't include a dash.

The following example shows how to set the IP address on a load balancer by retrieving a value from a linked template.

```
"publicIPAddress": {
  "id": "[reference('linkedTemplate').outputs.resourceID.value]"
}
```

You can't use the `reference` function in the outputs section of a nested template. To return the values for a deployed resource in a nested template, convert your nested template to a linked template.

Output example templates

TEMPLATE	DESCRIPTION
Copy variables	Creates complex variables and outputs those values. Doesn't deploy any resources.
Public IP address	Creates a public IP address and outputs the resource ID.
Load balancer	Links to the preceding template. Uses the resource ID in the output when creating the load balancer.

Comments and metadata

You have a few options for adding comments and metadata to your template.

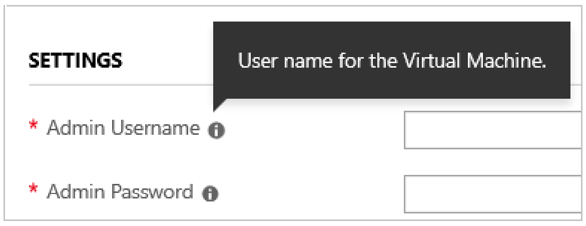
You can add a `metadata` object almost anywhere in your template. Resource Manager ignores the object, but your JSON editor may warn you that the property isn't valid. In the object, define the properties you need.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "metadata": {
    "comments": "This template was developed for demonstration purposes.",
    "author": "Example Name"
  },
```

For **parameters**, add a `metadata` object with a `description` property.

```
"parameters": {
  "adminUsername": {
    "type": "string",
    "metadata": {
      "description": "User name for the Virtual Machine."
    }
  },
```

When deploying the template through the portal, the text you provide in the description is automatically used as a tip for that parameter.



For **resources**, add a `comments` element or a metadata object. The following example shows both a comments element and a metadata object.

```
"resources": [
  {
    "comments": "Storage account used to store VM disks",
    "apiVersion": "2018-07-01",
    "type": "Microsoft.Storage/storageAccounts",
    "name": "[concat('storage', uniqueString(resourceGroup().id))]",
    "location": "[parameters('location')]",
    "metadata": {
      "comments": "These tags are needed for policy compliance."
    },
    "tags": {
      "Dept": "[parameters('deptName')]",
      "Environment": "[parameters('environment')]"
    },
    "sku": {
      "name": "Standard_LRS"
    },
    "kind": "Storage",
    "properties": {}
  }
]
```

For **outputs**, add a metadata object to the output value.

```
"outputs": {
  "hostname": {
    "type": "string",
    "value": "[reference(variables('publicIPAddressName')).dnsSettings.fqdn]",
    "metadata": {
      "comments": "Return the fully qualified domain name"
    }
  },
```

You can't add a metadata object to user-defined functions.

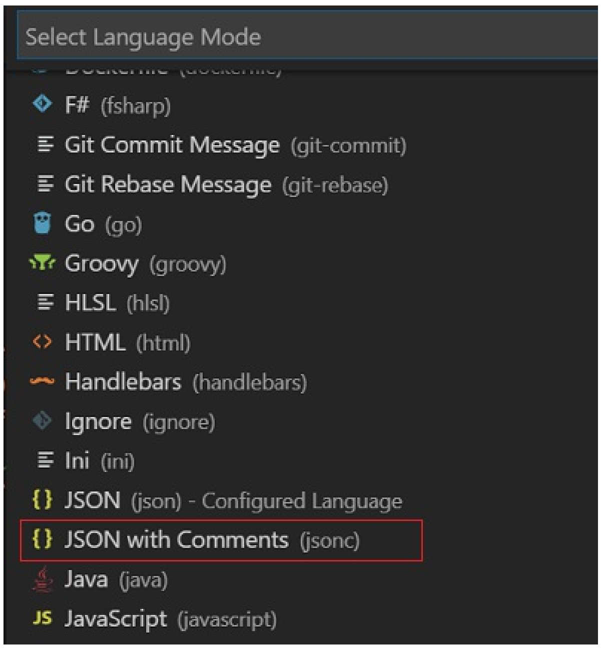
For inline comments, you can use `//` but this syntax doesn't work with all tools. You can't use Azure CLI to deploy the template with inline comments. And, you can't use the portal template editor to work on templates with inline comments. If you add this style of comment, be sure the tools you use support inline JSON comments.

```
{
  "type": "Microsoft.Compute/virtualMachines",
  "name": "[variables('vmName')]", // to customize name, change it in variables
  "location": "[parameters('location')]", //defaults to resource group location
  "apiVersion": "2018-10-01",
  "dependsOn": [ // storage account and network interface must be deployed first
    "[resourceId('Microsoft.Storage/storageAccounts/', variables('storageAccountName'))]",
    "[resourceId('Microsoft.Network/networkInterfaces/', variables('nicName'))]"
  ],
```

In VS Code, you can set the language mode to JSON with comments. The inline comments are no longer marked as invalid. To change the mode:

1. Open language mode selection (Ctrl+K M)

2. Select **JSON with Comments**.



Quickstarts and tutorials

Use the following quickstarts and tutorials to learn how to develop resource manager templates:

- Quickstarts

TITLE	DESCRIPTION
Use the Azure portal	Generate a template using the portal, and understand the process of editing and deploying the template.
Use Visual Studio Code	Use Visual Studio Code to create and edit templates, and how to use the Azure Cloud shell to deploy templates.
Use Visual Studio	Use Visual Studio to create, edit, and deploy templates.

- Tutorials

TITLE	DESCRIPTION
Utilize template reference	Utilize the template reference documentation to develop templates. In the tutorial, you find the storage account schema, and use the information to create an encrypted storage account.
Create multiple instances	Create multiple instances of Azure resources. In the tutorial, you create multiple instances of storage account.
Set resource deployment order	Define resource dependencies. In the tutorial, you create a virtual network, a virtual machine, and the dependent Azure resources. You learn how the dependencies are defined.
Use conditions	Deploy resources based on some parameter values. In the tutorial, you define a template to create a new storage account or use an existing storage account based on the value of a parameter.

TITLE	DESCRIPTION
Integrate key vault	Retrieve secrets/passwords from Azure Key Vault. In the tutorial, you create a virtual machine. The virtual machine administrator password is retrieved from a Key Vault.
Create linked templates	Modularize templates, and call other templates from a template. In the tutorial, you create a virtual network, a virtual machine, and the dependent resources. The dependent storage account is defined in a linked template.
Deploy virtual machine extensions	Perform post-deployment tasks by using extensions. In the tutorial, you deploy a customer script extension to install web server on the virtual machine.
Deploy SQL extensions	Perform post-deployment tasks by using extensions. In the tutorial, you deploy a customer script extension to install web server on the virtual machine.
Secure artifacts	Secure the artifacts needed to complete the deployments. In the tutorial, you learn how to secure the artifact used in the Deploy SQL extensions tutorial.
Use safe deployment practices	Use Azure Deployment manager.
Tutorial: Troubleshoot Resource Manager template deployments	Troubleshoot template deployment issues.

These tutorials can be used individually, or as a series to learn the major Resource Manager template development concepts.

Next steps

- To view complete templates for many different types of solutions, see the Azure Quickstart Templates.
- For details about the functions you can use from within a template, see Azure Resource Manager Template Functions.
- To combine several templates during deployment, see Using linked templates with Azure Resource Manager.
- For recommendations about creating templates, see Azure Resource Manager template best practices.
- For recommendations on creating Resource Manager templates that you can use across all Azure environments and Azure Stack, see Develop Azure Resource Manager templates for cloud consistency.

Azure Resource Manager template best practices

7/12/2019 • 9 minutes to read • Edit Online

This article gives recommendations about how to construct your Resource Manager template. These recommendations help you avoid common problems when using a template to deploy a solution.

For recommendations about how to govern your Azure subscriptions, see Azure enterprise scaffold: Prescriptive subscription governance.

For recommendations about how to build templates that work in all Azure cloud environments, see Develop Azure Resource Manager templates for cloud consistency.

Template limits

Limit the size of your template to 4 MB, and each parameter file to 64 KB. The 4-MB limit applies to the final state of the template after it has been expanded with iterative resource definitions, and values for variables and parameters.

You're also limited to:

- 256 parameters
- 256 variables
- 800 resources (including copy count)
- 64 output values
- 24,576 characters in a template expression

You can exceed some template limits by using a nested template. For more information, see Using linked templates when deploying Azure resources. To reduce the number of parameters, variables, or outputs, you can combine several values into an object. For more information, see Objects as parameters.

Resource group

When you deploy resources to a resource group, the resource group stores metadata about the resources. The metadata is stored in the location of the resource group.

If the resource group's region is temporarily unavailable, you can't update resources in the resource group because the metadata is unavailable. The resources in other regions will still function as expected, but you can't update them. To minimize risk, locate your resource group and resources in the same region.

Parameters

The information in this section can be helpful when you work with parameters.

General recommendations for parameters

- Minimize your use of parameters. Instead, use variables or literal values for properties that don't need to be specified during deployment.
- Use camel case for parameter names.
- Use parameters for settings that vary according to the environment, like SKU, size, or capacity.
- Use parameters for resource names that you want to specify for easy identification.

- Provide a description of every parameter in the metadata:

```
"parameters": {
     "storageAccountType": {
         "type": "string",
         "metadata": {
             "description": "The type of the new storage account created to store the VM disks."
         }
     }
}
```

- Define default values for parameters that aren't sensitive. By specifying a default value, it's easier to deploy the template, and users of your template see an example of an appropriate value. Any default value for a parameter must be valid for all users in the default deployment configuration.

```
"parameters": {
      "storageAccountType": {
          "type": "string",
          "defaultValue": "Standard_GRS",
          "metadata": {
              "description": "The type of the new storage account created to store the VM disks."
          }
      }
}
```

- To specify an optional parameter, don't use an empty string as a default value. Instead, use a literal value or a language expression to construct a value.

```
"storageAccountName": {
   "type": "string",
   "defaultValue": "[concat('storage', uniqueString(resourceGroup().id))]",
   "metadata": {
     "description": "Name of the storage account"
   }
},
```

- Don't use a parameter for the API version for a resource type. Resource properties and values can vary by version number. IntelliSense in a code editor can't determine the correct schema when the API version is set to a parameter. Instead, hard-code the API version in the template.

- Use `allowedValues` sparingly. Use it only when you must make sure some values aren't included in the permitted options. If you use `allowedValues` too broadly, you might block valid deployments by not keeping your list up-to-date.

- When a parameter name in your template matches a parameter in the PowerShell deployment command, Resource Manager resolves this naming conflict by adding the postfix **FromTemplate** to the template parameter. For example, if you include a parameter named **ResourceGroupName** in your template, it conflicts with the **ResourceGroupName** parameter in the New-AzResourceGroupDeployment cmdlet. During deployment, you're prompted to provide a value for **ResourceGroupNameFromTemplate**.

Security recommendations for parameters

- Always use parameters for user names and passwords (or secrets).

- Use `securestring` for all passwords and secrets. If you pass sensitive data in a JSON object, use the `secureObject` type. Template parameters with secure string or secure object types can't be read after resource deployment.

```
"parameters": {
     "secretValue": {
         "type": "securestring",
         "metadata": {
             "description": "The value of the secret to store in the vault."
         }
     }
}
```

- Don't provide default values for user names, passwords, or any value that requires a `secureString` type.
- Don't provide default values for properties that increase the attack surface area of the application.

Location recommendations for parameters

- Use a parameter to specify the location for resources, and set the default value to `resourceGroup().location`. Providing a location parameter enables users of the template to specify a location that they have permission to deploy to.

```
"parameters": {
   "location": {
     "type": "string",
     "defaultValue": "[resourceGroup().location]",
     "metadata": {
       "description": "The location in which the resources should be deployed."
     }
   }
},
```

- Don't specify `allowedValues` for the location parameter. The locations you specify might not be available in all clouds.
- Use the location parameter value for resources that are likely to be in the same location. This approach minimizes the number of times users are asked to provide location information.
- For resources that aren't available in all locations, use a separate parameter or specify a literal location value.

Variables

The following information can be helpful when you work with variables:

- Use variables for values that you need to use more than once in a template. If a value is used only once, a hard-coded value makes your template easier to read.
- Use variables for values that you construct from a complex arrangement of template functions. Your template is easier to read when the complex expression only appears in variables.
- Don't use variables for `apiVersion` on a resource. The API version determines the schema of the resource. Often, you can't change the version without changing the properties for the resource.
- You can't use the reference function in the **variables** section of the template. The **reference** function derives its value from the resource's runtime state. However, variables are resolved during the initial parsing of the template. Construct values that need the **reference** function directly in the **resources** or **outputs** section of the template.
- Include variables for resource names that must be unique.
- Use a copy loop in variables to create a repeated pattern of JSON objects.
- Remove unused variables.

Resource dependencies

When deciding what dependencies to set, use the following guidelines:

- Use the **reference** function and pass in the resource name to set an implicit dependency between resources that need to share a property. Don't add an explicit `dependsOn` element when you've already defined an implicit dependency. This approach reduces the risk of having unnecessary dependencies.
- Set a child resource as dependent on its parent resource.
- Resources with the condition element set to false are automatically removed from the dependency order. Set the dependencies as if the resource is always deployed.
- Let dependencies cascade without setting them explicitly. For example, your virtual machine depends on a virtual network interface, and the virtual network interface depends on a virtual network and public IP addresses. Therefore, the virtual machine is deployed after all three resources, but don't explicitly set the virtual machine as dependent on all three resources. This approach clarifies the dependency order and makes it easier to change the template later.
- If a value can be determined before deployment, try deploying the resource without a dependency. For example, if a configuration value needs the name of another resource, you might not need a dependency. This guidance doesn't always work because some resources verify the existence of the other resource. If you receive an error, add a dependency.

Resources

The following information can be helpful when you work with resources:

- To help other contributors understand the purpose of the resource, specify **comments** for each resource in the template:

  ```
  "resources": [
    {
        "name": "[variables('storageAccountName')]",
        "type": "Microsoft.Storage/storageAccounts",
        "apiVersion": "2016-01-01",
        "location": "[resourceGroup().location]",
        "comments": "This storage account is used to store the VM disks.",
        ...
    }
  ]
  ```

- If you use a *public endpoint* in your template (such as an Azure Blob storage public endpoint), *don't hard-code* the namespace. Use the **reference** function to dynamically retrieve the namespace. You can use this approach to deploy the template to different public namespace environments without manually changing the endpoint in the template. Set the API version to the same version that you're using for the storage account in your template:

  ```
  "osDisk": {
      "name": "osdisk",
      "vhd": {
          "uri": "[concat(reference(concat('Microsoft.Storage/storageAccounts/',
  variables('storageAccountName')), '2016-01-01').primaryEndpoints.blob,
  variables('vmStorageAccountContainerName'), '/',variables('OSDiskName'),'.vhd')]"
      }
  }
  ```

  If the storage account is deployed in the same template that you're creating, you don't need to specify the provider namespace when you reference the resource. The following example shows the simplified syntax:

```
"osDisk": {
     "name": "osdisk",
     "vhd": {
         "uri": "[concat(reference(variables('storageAccountName'), '2016-01-01').primaryEndpoints.blob, 
variables('vmStorageAccountContainerName'), '/',variables('OSDiskName'),'.vhd')]"
     }
}
```

If you have other values in your template that are configured to use a public namespace, change these values to reflect the same **reference** function. For example, you can set the **storageUri** property of the virtual machine diagnostics profile:

```
"diagnosticsProfile": {
     "bootDiagnostics": {
         "enabled": "true",
         "storageUri": "[reference(concat('Microsoft.Storage/storageAccounts/', 
variables('storageAccountName')), '2016-01-01').primaryEndpoints.blob]"
     }
}
```

You also can reference an existing storage account that is in a different resource group:

```
"osDisk": {
     "name": "osdisk",
     "vhd": {
         "uri":"[concat(reference(resourceId(parameters('existingResourceGroup'), 
'Microsoft.Storage/storageAccounts/', parameters('existingStorageAccountName')), '2016-01-
01').primaryEndpoints.blob,  variables('vmStorageAccountContainerName'), '/', 
variables('OSDiskName'),'.vhd')]"
     }
}
```

- Assign public IP addresses to a virtual machine only when an application requires it. To connect to a virtual machine (VM) for debugging, or for management or administrative purposes, use inbound NAT rules, a virtual network gateway, or a jumpbox.

  For more information about connecting to virtual machines, see:

  - Run VMs for an N-tier architecture in Azure
  - Set up WinRM access for VMs in Azure Resource Manager
  - Allow external access to your VM by using the Azure portal
  - Allow external access to your VM by using PowerShell
  - Allow external access to your Linux VM by using Azure CLI
- The **domainNameLabel** property for public IP addresses must be unique. The **domainNameLabel** value must be between 3 and 63 characters long, and follow the rules specified by this regular expression: `^[a-z][a-z0-9-]{1,61}[a-z0-9]$`. Because the **uniqueString** function generates a string that is 13 characters long, the **dnsPrefixString** parameter is limited to 50 characters:

```
"parameters": {
     "dnsPrefixString": {
         "type": "string",
         "maxLength": 50,
         "metadata": {
             "description": "The DNS label for the public IP address. It must be lowercase. It should
match the following regular expression, or it will raise an error: ^[a-z][a-z0-9-]{1,61}[a-z0-9]$"
         }
     }
},
"variables": {
     "dnsPrefix": "[concat(parameters('dnsPrefixString'),uniquestring(resourceGroup().id))]"
}
```

- When you add a password to a custom script extension, use the **commandToExecute** property in the **protectedSettings** property:

```
"properties": {
     "publisher": "Microsoft.Azure.Extensions",
     "type": "CustomScript",
     "typeHandlerVersion": "2.0",
     "autoUpgradeMinorVersion": true,
     "settings": {
         "fileUris": [
             "[concat(variables('template').assets, '/lamp-app/install_lamp.sh')]"
         ]
     },
     "protectedSettings": {
         "commandToExecute": "[concat('sh install_lamp.sh ', parameters('mySqlPassword'))]"
     }
}
```

NOTE

To ensure that secrets are encrypted when they are passed as parameters to VMs and extensions, use the **protectedSettings** property of the relevant extensions.

Outputs

If you use a template to create public IP addresses, include an outputs section that returns details of the IP address and the fully qualified domain name (FQDN). You can use output values to easily retrieve details about public IP addresses and FQDNs after deployment.

```
"outputs": {
    "fqdn": {
        "value": "[reference(parameters('publicIPAddresses_name')).dnsSettings.fqdn]",
        "type": "string"
    },
    "ipaddress": {
        "value": "[reference(parameters('publicIPAddresses_name')).ipAddress]",
        "type": "string"
    }
}
```

Next steps

- For information about the structure of the Resource Manager template file, see Understand the structure and syntax of Azure Resource Manager Templates.

- For recommendations about how to build templates that work in all Azure cloud environments, see Develop Azure Resource Manager templates for cloud consistency.

Develop Azure Resource Manager templates for cloud consistency

4/19/2019 • 26 minutes to read • Edit Online

IMPORTANT

Using this Azure feature from PowerShell requires the `AzureRM` module installed. This is an older module only available for Windows PowerShell 5.1 that no longer receives new features. The `Az` and `AzureRM` modules are **not** compatible when installed for the same versions of PowerShell. If you need both versions:

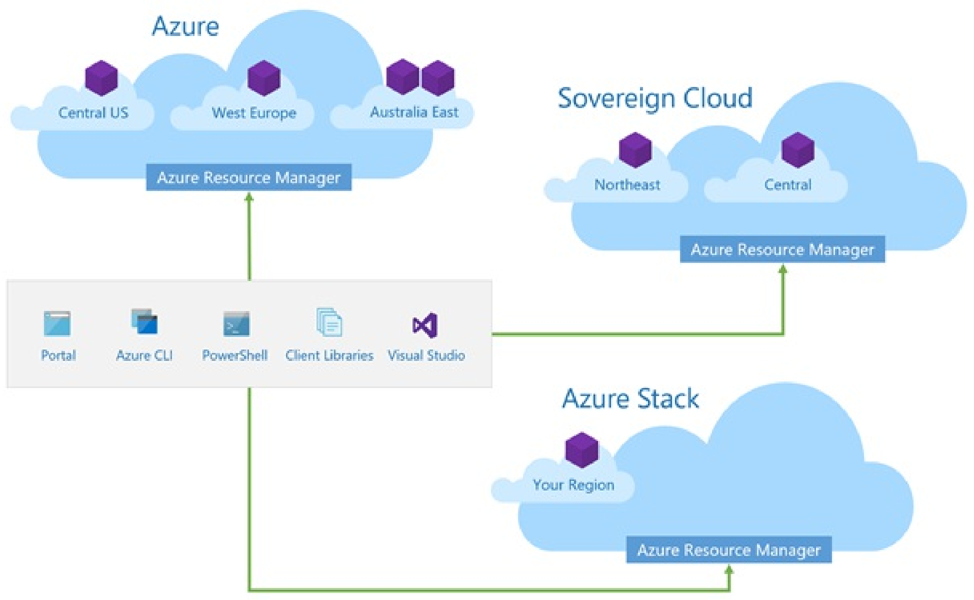
1. Uninstall the Az module from a PowerShell 5.1 session.
2. Install the AzureRM module from a PowerShell 5.1 session.
3. Download and install PowerShell Core 6.x or later.
4. Install the Az module in a PowerShell Core session.

A key benefit of Azure is consistency. Development investments for one location are reusable in another. A template makes your deployments consistent and repeatable across environments, including the global Azure, Azure sovereign clouds, and Azure Stack. To reuse templates across clouds, however, you need to consider cloud-specific dependencies as this guide explains.

Microsoft offers intelligent, enterprise-ready cloud services in many locations, including:

- The global Azure platform supported by a growing network of Microsoft-managed datacenters in regions around the world.
- Isolated sovereign clouds like Azure Germany, Azure Government, and Azure China (Azure operated by 21Vianet). Sovereign clouds provide a consistent platform with most of the same great features that global Azure customers have access to.
- Azure Stack, a hybrid cloud platform that lets you deliver Azure services from your organization's datacenter. Enterprises can set up Azure Stack in their own datacenters, or consume Azure Services from service providers, running Azure Stack in their facilities (sometimes known as hosted regions).

At the core of all these clouds, Azure Resource Manager provides an API that allows a wide variety of user interfaces to communicate with the Azure platform. This API gives you powerful infrastructure-as-code capabilities. Any type of resource that is available on the Azure cloud platform can be deployed and configured with Azure Resource Manager. With a single template, you can deploy and configure your complete application to an operational end state.



The consistency of global Azure, the sovereign clouds, hosted clouds, and a cloud in your datacenter helps you benefit from Azure Resource Manager. You can reuse your development investments across these clouds when you set up template-based resource deployment and configuration.

However, even though the global, sovereign, hosted, and hybrid clouds provide consistent services, not all clouds are identical. As a result, you can create a template with dependencies on features available only in a specific cloud.

The rest of this guide describes the areas to consider when planning to develop new or updating existing Azure Resource Manager templates for Azure Stack. In general, your checklist should include the following items:

- Verify that the functions, endpoints, services, and other resources in your template are available in the target deployment locations.
- Store nested templates and configuration artifacts in accessible locations, ensuring access across clouds.
- Use dynamic references instead of hard-coding links and elements.
- Ensure the template parameters you use work in the target clouds.
- Verify that resource-specific properties are available the target clouds.

For an introduction to Azure Resource Manger templates, see Template deployment.

Ensure template functions work

The basic syntax of a Resource Manager template is JSON. Templates use a superset of JSON, extending the syntax with expressions and functions. The template language processor is frequently updated to support additional template functions. For a detailed explanation of the available template functions, see Azure Resource Manager template functions.

New template functions that are introduced to Azure Resource Manager aren't immediately available in the sovereign clouds or Azure Stack. To deploy a template successfully, all functions referenced in the template must be available in the target cloud.

Azure Resource Manager capabilities will always be introduced to global Azure first. You can use the following PowerShell script to verify whether newly introduced template functions are also available in Azure Stack:

1. Make a clone of the GitHub repository: https://github.com/marcvaneijk/arm-template-functions.
2. Once you have a local clone of the repository, connect to the destination's Azure Resource Manager with PowerShell.

3. Import the psm1 module and execute the Test-AzureRmureRmTemplateFunctions cmdlet:

```
# Import the module
Import-module <path to local clone>\AzTemplateFunctions.psm1

# Execute the Test-AzureRmTemplateFunctions cmdlet
Test-AzureRmTemplateFunctions -path <path to local clone>
```

The script deploys multiple, minimized templates, each containing only unique template functions. The output of the script reports the supported and unavailable template functions.

Working with linked artifacts

A template can contain references to linked artifacts and contain a deployment resource that links to another template. The linked templates (also referred to as nested template) are retrieved by Resource Manager at runtime. A template can also contain references to artifacts for virtual machine (VM) extensions. These artifacts are retrieved by the VM extension running inside of the VM for configuration of the VM extension during the template deployment.

The following sections describe considerations for cloud consistency when developing templates that include artifacts that are external to the main deployment template.

Use nested templates across regions

Templates can be decomposed into small, reusable templates, each of which has a specific purpose and can be reused across deployment scenarios. To execute a deployment, you specify a single template known as the main or master template. It specifies the resources to deploy, such as virtual networks, VMs, and web apps. The main template can also contain a link to another template, meaning you can nest templates. Likewise, a nested template can contain links to other templates. You can nest up to five levels deep.

The following code shows how the templateLink parameter refers to a nested template:

```
"resources": [
  {
     "apiVersion": "2017-05-10",
     "name": "linkedTemplate",
     "type": "Microsoft.Resources/deployments",
     "properties": {
       "mode": "incremental",
       "templateLink": {
          "uri":"https://mystorageaccount.blob.core.windows.net/AzureTemplates/vNet.json",
          "contentVersion":"1.0.0.0"
       }
     }
  }
]
```

Azure Resource Manager evaluates the main template at runtime and retrieves and evaluates each nested template. After all nested templates are retrieved, the template is flattened, and further processing is initiated.

Make linked templates accessible across clouds

Consider where and how to store any linked templates you use. At runtime, Azure Resource Manager fetches—and therefore requires direct access to—any linked templates. A common practice is to use GitHub to store the nested templates. A GitHub repository can contain files that are accessible publicly through a URL. Although this technique works well for the public cloud and the sovereign clouds, an Azure Stack environment might be located on a corporate network or on a disconnected remote location, without any outbound Internet access. In those cases, Azure Resource Manager would fail to retrieve the nested templates.

A better practice for cross-cloud deployments is to store your linked templates in a location that is accessible for the target cloud. Ideally all deployment artifacts are maintained in and deployed from a continuous integration/continuous development (CI/CD) pipeline. Alternatively, you can store nested templates in a blob storage container, from which Azure Resource Manager can retrieve them.

Since the blob storage on each cloud uses a different endpoint fully qualified domain name (FQDN), configure the template with the location of the linked templates with two parameters. Parameters can accept user input at deployment time. Templates are typically authored and shared by multiple people, so a best practice is to use a standard name for these parameters. Naming conventions help make templates more reusable across regions, clouds, and authors.

In the following code, `_artifactsLocation` is used to point to a single location, containing all deployment-related artifacts. Notice that a default value is provided. At deployment time, if no input value is specified for `_artifactsLocation`, the default value is used. The `_artifactsLocationSasToken` is used as input for the `sasToken`. The default value should be an empty string for scenarios where the `_artifactsLocation` isn't secured — for example, a public GitHub repository.

```
"parameters": {
  "_artifactsLocation": {
    "type": "string",
    "metadata": {
      "description": "The base URI where artifacts required by this template are located."
    },
    "defaultValue": "https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/201-vm-custom-
script-windows/"
  },
  "_artifactsLocationSasToken": {
    "type": "securestring",
    "metadata": {
      "description": "The sasToken required to access _artifactsLocation."
    },
    "defaultValue": ""
  }
}
```

Throughout the template, links are generated by combining the base URI (from the `_artifactsLocation` parameter) with an artifact-relative path and the `_artifactsLocationSasToken`. The following code shows how to specify the link to the nested template using the uri template function:

```
"resources": [
  {
    "name": "shared",
    "type": "Microsoft.Resources/deployments",
    "apiVersion": "2015-01-01",
    "properties": {
      "mode": "Incremental",
      "templateLink": {
        "uri": "[uri(parameters('_artifactsLocation'), concat('nested/vnet.json',
parameters('_artifactsLocationSasToken')))]",
        "contentVersion": "1.0.0.0"
      }
    }
  }
]
```

By using this approach, the default value for the `_artifactsLocation` parameter is used. If the linked templates need to be retrieved from a different location, the parameter input can be used at deployment time to override the default value—no change to the template itself is needed.

Use _artifactsLocation instead of hardcoding links

Besides being used for nested templates, the URL in the `_artifactsLocation` parameter is used as a base for all related artifacts of a deployment template. Some VM extensions include a link to a script stored outside the template. For these extensions, you should not hardcode the links. For example, the Custom Script and PowerShell DSC extensions may link to an external script on GitHub as shown:

```
"properties": {
  "publisher": "Microsoft.Compute",
  "type": "CustomScriptExtension",
  "typeHandlerVersion": "1.9",
  "autoUpgradeMinorVersion": true,
  "settings": {
    "fileUris": [
      "https://raw.githubusercontent.com/Microsoft/dotnet-core-sample-templates/master/dotnet-core-music-windows/scripts/configure-music-app.ps1"
    ]
  }
}
```

Hardcoding the links to the script potentially prevents the template from deploying successfully to another location. During configuration of the VM resource, the VM agent running inside the VM initiates a download of all the scripts linked in the VM extension, and then stores the scripts on the VM's local disk. This approach functions like the nested template links explained earlier in the "Use nested templates across regions" section.

Resource Manager retrieves nested templates at runtime. For VM extensions, the retrieval of any external artifacts is performed by the VM agent. Besides the different initiator of the artifact retrieval, the solution in the template definition is the same. Use the _artifactsLocation parameter with a default value of the base path where all the artifacts are stored (including the VM extension scripts) and the `_artifactsLocationSasToken` parameter for the input for the sasToken.

```
"parameters": {
  "_artifactsLocation": {
    "type": "string",
    "metadata": {
      "description": "The base URI where artifacts required by this template are located."
    },
    "defaultValue": "https://raw.githubusercontent.com/Microsoft/dotnet-core-sample-templates/master/dotnet-core-music-windows/"
  },
  "_artifactsLocationSasToken": {
    "type": "securestring",
    "metadata": {
      "description": "The sasToken required to access _artifactsLocation."
    },
    "defaultValue": ""
  }
}
```

To construct the absolute URI of an artifact, the preferred method is to use the uri template function, instead of the concat template function. By replacing hardcoded links to the scripts in the VM extension with the uri template function, this functionality in the template is configured for cloud consistency.

```
"properties": {
  "publisher": "Microsoft.Compute",
  "type": "CustomScriptExtension",
  "typeHandlerVersion": "1.9",
  "autoUpgradeMinorVersion": true,
  "settings": {
    "fileUris": [
      "[uri(parameters('_artifactsLocation'), concat('scripts/configure-music-app.ps1',
parameters('_artifactsLocationSasToken')))]"
    ]
  }
}
```

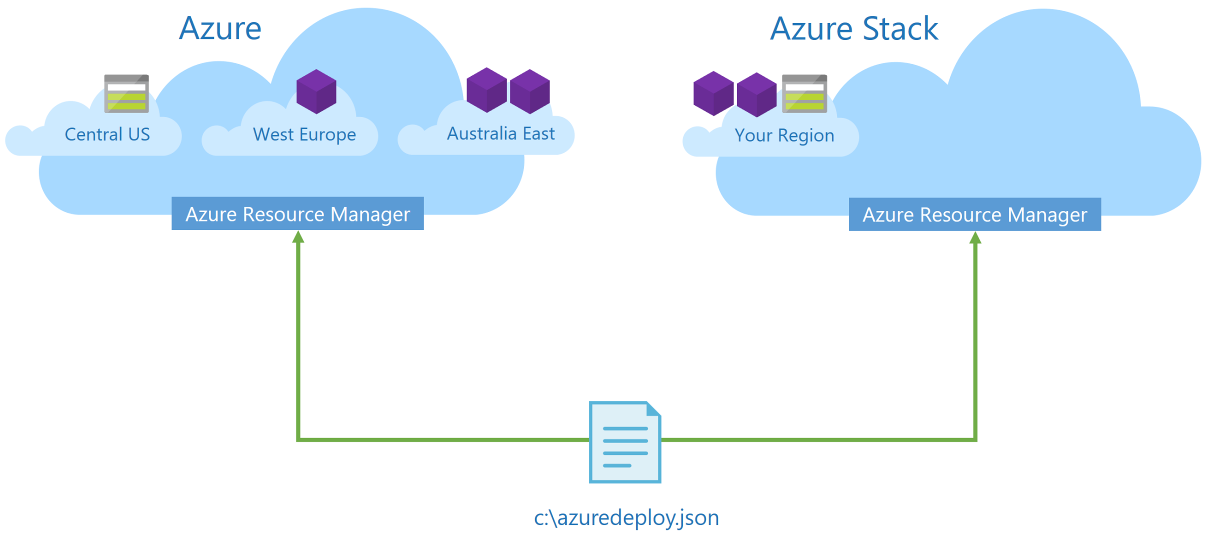
With this approach, all deployment artifacts, including configuration scripts, can be stored in the same location with the template itself. To change the location of all the links, you only need to specify a different base URL for the *artifactsLocation parameters*.

Factor in differing regional capabilities

With the agile development and continuous flow of updates and new services introduced to Azure, regions can differ in availability of services or updates. After rigorous internal testing, new services or updates to existing services are usually introduced to a small audience of customers participating in a validation program. After successful customer validation, the services or updates are made available within a subset of Azure regions, then introduced to more regions, rolled out to the sovereign clouds, and potentially made available for Azure Stack customers as well.

Knowing that Azure regions and clouds may differ in their available services, you can make some proactive decisions about your templates. A good place to start is by examining the available resource providers for a cloud. A resource provider tells you the set of resources and operations that are available for an Azure service.

A template deploys and configures resources. A resource type is provided by a resource provider. For example, the compute resource provider (Microsoft.Compute), provides multiple resource types such as virtualMachines and availabilitySets. Each resource provider provides an API to Azure Resource Manager defined by a common contract, enabling a consistent, unified authoring experience across all resource providers. However, a resource provider that is available in global Azure may not be available in a sovereign cloud or an Azure Stack region.



To verify the resource providers that are available in a given cloud, run the following script in the Azure command line interface (CLI):

```
az provider list --query "[].{Provider:namespace, Status:registrationState}" --out table
```

You can also use the following PowerShell cmdlet to see available resource providers:

```
Get-AzureRmResourceProvider -ListAvailable | Select-Object ProviderNamespace, RegistrationState
```

Verify the version of all resource types

A set of properties is common for all resource types, but each resource also has its own specific properties. New features and related properties are added to existing resource types at times through a new API version. A resource in a template has its own API version property - `apiVersion`. This versioning ensures that an existing resource configuration in a template is not affected by changes on the platform.

New API versions introduced to existing resource types in global Azure might not immediately be available in all regions, sovereign clouds, or Azure Stack. To view a list of the available resource providers, resource types, and API versions for a cloud, you can use Resource Explorer in Azure portal. Search for Resource Explorer in the All Services menu. Expand the Providers node in Resource Explorer to return all the available resource providers, their resource types, and API versions in that cloud.

To list the available API version for all resource types in a given cloud in Azure CLI, run the following script:

```
az provider list --query "[].{namespace:namespace, resourceType:resourceType[]}"
```

You can also use the following PowerShell cmdlet:

```
Get-AzureRmResourceProvider | select-object ProviderNamespace -ExpandProperty ResourceTypes | ft
ProviderNamespace, ResourceTypeName, ApiVersions
```

Refer to resource locations with a parameter

A template is always deployed into a resource group that resides in a region. Besides the deployment itself, each resource in a template also has a location property that you use to specify the region to deploy in. To develop your template for cloud consistency, you need a dynamic way to refer to resource locations, because each Azure Stack can contain unique location names. Usually resources are deployed in the same region as the resource group, but to support scenarios such as cross-region application availability, it can be useful to spread resources across regions.

Even though you could hardcode the region names when specifying the resource properties in a template, this approach doesn't guarantee that the template can be deployed to other Azure Stack environments, because the region name most likely doesn't exist there.

To accommodate different regions, add an input parameter location to the template with a default value. The default value will be used if no value is specified during deployment.

The template function `[resourceGroup()]` returns an object that contains the following key/value pairs:

```
{
  "id": "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}",
  "name": "{resourceGroupName}",
  "location": "{resourceGroupLocation}",
  "tags": {
  },
  "properties": {
    "provisioningState": "{status}"
  }
}
```

By referencing the location key of the object in the defaultValue of the input parameter, Azure Resource Manager will, at runtime, replace the `[resourceGroup().location]` template function with the name of the location of the resource group the template is deployed to.

```
"parameters": {
  "location": {
    "type": "string",
    "metadata": {
      "description": "Location the resources will be deployed to."
    },
    "defaultValue": "[resourceGroup().location]"
  }
},
"resources": [
  {
    "name": "storageaccount1",
    "type": "Microsoft.Storage/storageAccounts",
    "apiVersion": "2015-06-15",
    "location": "[parameters('location')]",
    ...
```

With this template function, you can deploy your template to any cloud without even knowing the region names in advance. In addition, a location for a specific resource in the template can differ from the resource group location. In this case, you can configure it by using additional input parameters for that specific resource, while the other resources in the same template still use the initial location input parameter.

Track versions using API profiles

It can be very challenging to keep track of all the available resource providers and related API versions that are present in Azure Stack. For example, at the time of writing, the latest API version for **Microsoft.Compute/availabilitySets** in Azure is `2018-04-01`, while the available API version common to Azure and Azure Stack is `2016-03-30`. The common API version for **Microsoft.Storage/storageAccounts** shared among all Azure and Azure Stack locations is `2016-01-01`, while the latest API version in Azure is `2018-02-01`.

For this reason, Resource Manager introduced the concept of API profiles to templates. Without API profiles, each resource in a template is configured with an `apiVersion` element that describes the API version for that specific resource.

```json
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "location": {
            "type": "string",
            "metadata": {
                "description": "Location the resources will be deployed to."
            },
            "defaultValue": "[resourceGroup().location]"
        }
    },
    "variables": {},
    "resources": [
        {
            "name": "mystorageaccount",
            "type": "Microsoft.Storage/storageAccounts",
            "apiVersion": "2016-01-01",
            "location": "[parameters('location')]",
            "properties": {
                "accountType": "Standard_LRS"
            }
        },
        {
            "name": "myavailabilityset",
            "type": "Microsoft.Compute/availabilitySets",
            "apiVersion": "2016-03-30",
            "location": "[parameters('location')]",
            "properties": {
                "platformFaultDomainCount": 2,
                "platformUpdateDomainCount": 2
            }
        }
    ],
    "outputs": {}
}
```

An API profile version acts as an alias for a single API version per resource type common to Azure and Azure Stack. Instead of specifying an API version for each resource in a template, you specify only the API profile version in a new root element called `apiProfile` and omit the `apiVersion` element for the individual resources.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "apiProfile": "2018–03-01-hybrid",
    "parameters": {
        "location": {
            "type": "string",
            "metadata": {
                "description": "Location the resources will be deployed to."
            },
            "defaultValue": "[resourceGroup().location]"
        }
    },
    "variables": {},
    "resources": [
        {
            "name": "mystorageaccount",
            "type": "Microsoft.Storage/storageAccounts",
            "location": "[parameters('location')]",
            "properties": {
                "accountType": "Standard_LRS"
            }
        },
        {
            "name": "myavailabilityset",
            "type": "Microsoft.Compute/availabilitySets",
            "location": "[parameters('location')]",
            "properties": {
                "platformFaultDomainCount": 2,
                "platformUpdateDomainCount": 2
            }
        }
    ],
    "outputs": {}
}
```

The API profile ensures that the API versions are available across locations, so you do not have to manually verify the apiVersions that are available in a specific location. To ensure the API versions referenced by your API profile are present in an Azure Stack environment, the Azure Stack operators must keep the solution up-to-date based on the policy for support. If a system is more than six months out of date, it is considered out of compliance, and the environment must be updated.

The API profile isn't a required element in a template. Even if you add the element, it will only be used for resources for which no `apiVersion` is specified. This element allows for gradual changes but doesn't require any changes to existing templates.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "apiProfile": "2018-03-01-hybrid",
    "parameters": {
        "location": {
            "type": "string",
            "metadata": {
                "description": "Location the resources will be deployed to."
            },
            "defaultValue": "[resourceGroup().location]"
        }
    },
    "variables": {},
    "resources": [
        {
            "name": "mystorageaccount",
            "type": "Microsoft.Storage/storageAccounts",
            "apiVersion": "2016-01-01",
            "location": "[parameters('location')]",
            "properties": {
                "accountType": "Standard_LRS"
            }
        },
        {
            "name": "myavailabilityset",
            "type": "Microsoft.Compute/availabilitySets",
            "location": "[parameters('location')]",
            "properties": {
                "platformFaultDomainCount": 2,
                "platformUpdateDomainCount": 2
            }
        }
    ],
    "outputs": {}
}
```

Check endpoint references

Resources can have references to other services on the platform. For example, a public IP can have a public DNS name assigned to it. The public cloud, the sovereign clouds, and Azure Stack solutions have their own distinct endpoint namespaces. In most cases, a resource requires only a prefix as input in the template. During runtime, Azure Resource Manager appends the endpoint value to it. Some endpoint values need to be explicitly specified in the template.

NOTE

To develop templates for cloud consistency, don't hardcode endpoint namespaces.

The following two examples are common endpoint namespaces that need to be explicitly specified when creating a resource:

- Storage accounts (blob, queue, table and file)
- Connection strings for databases and Azure Cache for Redis

Endpoint namespaces can also be used in the output of a template as information for the user when the deployment completes. The following are common examples:

- Storage accounts (blob, queue, table and file)
- Connection strings (MySql, SQLServer, SQLAzure, Custom, NotificationHub, ServiceBus, EventHub, ApiHub,

DocDb, RedisCache, PostgreSQL)
- Traffic Manager
- domainNameLabel of a public IP address
- Cloud services

In general, avoid hardcoded endpoints in a template. The best practice is to use the reference template function to retrieve the endpoints dynamically. For example, the endpoint most commonly hardcoded is the endpoint namespace for storage accounts. Each storage account has a unique FQDN that is constructed by concatenating the name of the storage account with the endpoint namespace. A blob storage account named mystorageaccount1 results in different FQDNs depending on the cloud:

- **mystorageaccount1.blob.core.windows.net** when created on the global Azure cloud.
- **mystorageaccount1.blob.core.chinacloudapi.cn** when created in the Azure China cloud.

The following reference template function retrieves the endpoint namespace from the storage resource provider:

```
"diskUri":"[concat(reference(concat('Microsoft.Storage/storageAccounts/', variables('storageAccountName')),
'2015-06-15').primaryEndpoints.blob, 'container/myosdisk.vhd')]"
```

By replacing the hardcoded value of the storage account endpoint with the `reference` template function, you can use the same template to deploy to different environments successfully without making any changes to the endpoint reference.

Refer to existing resources by unique ID

You can also refer to an existing resource from the same or another resource group, and within the same subscription or another subscription, within the same tenant in the same cloud. To retrieve the resource properties, you must use the unique identifier for the resource itself. The `resourceId` template function retrieves the unique ID of a resource such as SQL Server as the following code shows:

```
"outputs": {
  "resourceId":{
    "type": "string",
    "value": "[resourceId('otherResourceGroup', 'Microsoft.Sql/servers', parameters('serverName'))]"
  }
}
```

You can then use the `resourceId` function inside the `reference` template function to retrieve the properties of a database. The return object contains the `fullyQualifiedDomainName` property that holds the full endpoint value. This value is retrieved at runtime and provides the cloud environment-specific endpoint namespace. To define the connection string without hardcoding the endpoint namespace, you can refer to the property of the return object directly in the connection string as shown:

```
"[concat('Server=tcp:', reference(resourceId('sql', 'Microsoft.Sql/servers', parameters('test')), '2015-05-01-
preview').fullyQualifiedDomainName, ',1433;Initial Catalog=', parameters('database'),';User ID=',
parameters('username'), ';Password=', parameters('pass'), ';Encrypt=True;')]"
```

Consider resource properties

Specific resources within Azure Stack environments have unique properties you must consider in your template.

Ensure VM images are available

Azure provides a rich selection of VM images. These images are created and prepared for deployment by Microsoft and partners. The images form the foundation for VMs on the platform. However, a cloud-consistent template should refer to available parameters only — in particular, the publisher, offer, and SKU of the VM images available to the global Azure, Azure sovereign clouds, or an Azure Stack solution.

To retrieve a list of the available VM images in a location, run the following Azure CLI command:

```
az vm image list -all
```

You can retrieve the same list with the Azure PowerShell cmdlet Get-AzureRmVMImagePublisher and specify the location you want with the `-Location` parameter. For example:

```
Get-AzureRmVMImagePublisher -Location "West Europe" | Get-AzureRmVMImageOffer | Get-AzureRmVMImageSku | Get-
AzureRmVMImage
```

This command takes a couple of minutes to return all the available images in the West Europe region of the global Azure cloud.

If you made these VM images available to Azure Stack, all the available storage would be consumed. To accommodate even the smallest scale unit, Azure Stack allows you to select the images you want to add to an environment.

The following code sample shows a consistent approach to refer to the publisher, offer, and SKU parameters in your Azure Resource Manager templates:

```
"storageProfile": {
    "imageReference": {
    "publisher": "MicrosoftWindowsServer",
    "offer": "WindowsServer",
    "sku": "2016-Datacenter",
    "version": "latest"
    }
}
```

Check local VM sizes

To develop your template for cloud consistency, you need to make sure the VM size you want is available in all target environments. VM sizes are a grouping of performance characteristics and capabilities. Some VM sizes depend on the hardware that the VM runs on. For example, if you want to deploy a GPU-optimized VM, the hardware that runs the hypervisor needs to have the hardware GPUs.

When Microsoft introduces a new size of VM that has certain hardware dependencies, the VM size is usually made available first in a small subset of regions in the Azure cloud. Later, it is made available to other regions and clouds. To make sure the VM size exists in each cloud you deploy to, you can retrieve the available sizes with the following Azure CLI command:

```
az vm list-sizes --location "West Europe"
```

For Azure PowerShell, use:

```
Get-AzureRmVMSize -Location "West Europe"
```

For a full list of available services, see Products available by region.

Check use of Azure Managed Disks in Azure Stack

Managed disks handle the storage for an Azure tenant. Instead of explicitly creating a storage account and specifying the URI for a virtual hard disk (VHD), you can use managed disks to implicitly perform these actions when you deploy a VM. Managed disks enhance availability by placing all the disks from VMs in the same availability set into different storage units. Additionally, existing VHDs can be converted from Standard to Premium storage with significantly less downtime.

Although managed disks are on the roadmap for Azure Stack, they are currently not supported. Until they are, you can develop cloud-consistent templates for Azure Stack by explicitly specifying VHDs using the `vhd` element in the template for the VM resource as shown:

```
"storageProfile": {
  "imageReference": {
    "publisher": "MicrosoftWindowsServer",
    "offer": "WindowsServer",
    "sku": "[parameters('windowsOSVersion')]",
    "version": "latest"
  },
  "osDisk": {
    "name": "osdisk",
    "vhd": {
      "uri": "[concat(reference(resourceId('Microsoft.Storage/storageAccounts/',
variables('storageAccountName')), '2015-06-15').primaryEndpoints.blob, 'vhds/osdisk.vhd')]"
    },
    "caching": "ReadWrite",
    "createOption": "FromImage"
  }
}
```

In contrast, to specify a managed disk configuration in a template, remove the `vhd` element from the disk configuration.

```
"storageProfile": {
  "imageReference": {
    "publisher": "MicrosoftWindowsServer",
    "offer": "WindowsServer",
    "sku": "[parameters('windowsOSVersion')]",
    "version": "latest"
  },
  "osDisk": {
    "caching": "ReadWrite",
    "createOption": "FromImage"
  }
}
```

The same changes also apply data disks.

Verify that VM extensions are available in Azure Stack

Another consideration for cloud consistency is the use of virtual machine extensions to configure the resources inside a VM. Not all VM extensions are available in Azure Stack. A template can specify the resources dedicated to the VM extension, creating dependencies and conditions within the template.

For example, if you want to configure a VM running Microsoft SQL Server, the VM extension can configure SQL Server as part the template deployment. Consider what happens if the deployment template also contains an application server configured to create a database on the VM running SQL Server. Besides also using a VM extension for the application servers, you can configure the dependency of the application server on the successful return of the SQL Server VM extension resource. This approach ensures the VM running SQL Server is configured and available when the application server is instructed to create the database.

The declarative approach of the template allows you to define the end state of the resources and their inter-dependencies, while the platform takes care of the logic required for the dependencies.

Check that VM extensions are available

Many types of VM extensions exist. When developing template for cloud consistency, make sure to use only the extensions that are available in all the regions the template targets.

To retrieve a list of the VM extensions that are available for a specific region (in this example, `myLocation`), run the following Azure CLI command:

```
az vm extension image list --location myLocation
```

You can also execute the Azure PowerShell Get-AzureRmVmImagePublisher cmdlet and use `-Location` to specify the location of the virtual machine image. For example:

```
Get-AzureRmVmImagePublisher -Location myLocation | Get-AzureRmVMExtensionImageType | Get-
AzureRmVMExtensionImage | Select Type, Version
```

Ensure that versions are available

Since VM extensions are first-party Resource Manager resources, they have their own API versions. As the following code shows, the VM extension type is a nested resource in the Microsoft.Compute resource provider.

```
{
    "apiVersion": "2015-06-15",
    "type": "Microsoft.Compute/virtualMachines/extensions",
    "name": "myExtension",
    "location": "[parameters('location')]",
    ...
```

The API version of the VM extension resource must be present in all the locations you plan to target with your template. The location dependency works like the resource provider API version availability discussed earlier in the "Verify the version of all resource types" section.

To retrieve a list of the available API versions for the VM extension resource, use the Get-AzureRmResourceProvider cmdlet with the **Microsoft.Compute** resource provider as shown:

```
Get-AzureRmResourceProvider -ProviderNamespace "Microsoft.Compute" | Select-Object -ExpandProperty
ResourceTypes | Select ResourceTypeName, Locations, ApiVersions | where {$_.ResourceTypeName -eq
"virtualMachines/extensions"}
```

You can also use VM extensions in virtual machine scale sets. The same location conditions apply. To develop your template for cloud consistency, make sure the API versions are available in all the locations you plan on deploying to. To retrieve the API versions of the VM extension resource for scale sets, use the same cmdlet as before, but specify the virtual machine scale sets resource type as shown:

```
Get-AzureRmResourceProvider -ProviderNamespace "Microsoft.Compute" | Select-Object -ExpandProperty
ResourceTypes | Select ResourceTypeName, Locations, ApiVersions | where {$_.ResourceTypeName -eq
"virtualMachineScaleSets/extensions"}
```

Each specific extension is also versioned. This version is shown in the `typeHandlerVersion` property of the VM extension. Make sure that the version specified in the `typeHandlerVersion` element of your template's VM extensions are available in the locations where you plan to deploy the template. For example, the following code specifies version 1.7:

```
{
    "name": "MyCustomScriptExtension",
    "type": "extensions",
    "apiVersion": "2016-03-30",
    "location": "[parameters('location')]",
    "dependsOn": [
        "[concat('Microsoft.Compute/virtualMachines/myVM', copyindex())]"
    ],
    "properties": {
        "publisher": "Microsoft.Compute",
        "type": "CustomScriptExtension",
        "typeHandlerVersion": "1.7",
        ...
```

To retrieve a list of the available versions for a specific VM extension, use the Get-AzureRmVMExtensionImage cmdlet. The following example retrieves the available versions for the PowerShell DSC (Desired State Configuration) VM extension from **myLocation**:

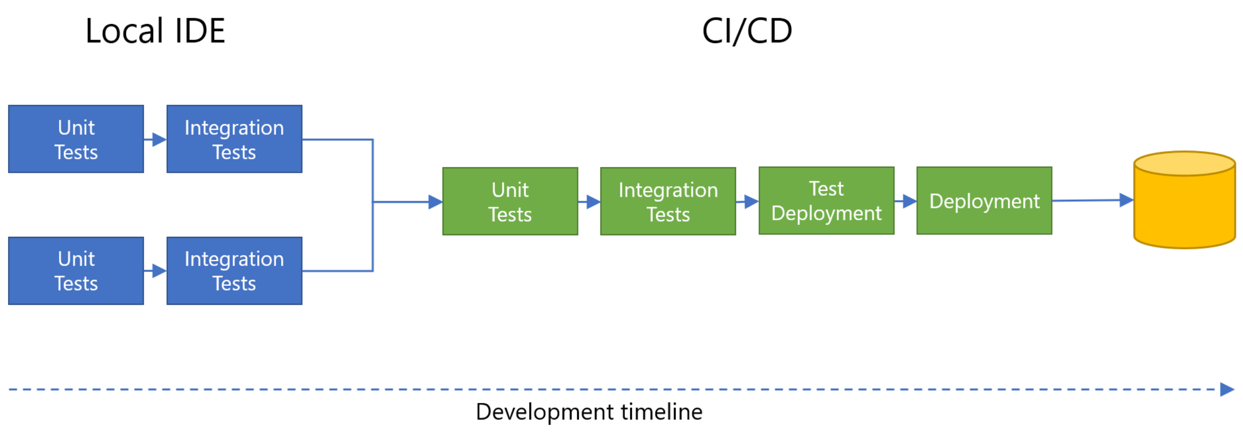
```
Get-AzureRmVMExtensionImage -Location myLocation -PublisherName Microsoft.PowerShell -Type DSC | FT
```

To get a list of publishers, use the Get-AzureRmVmImagePublisher command. To request type, use the Get-AzureRmVMExtensionImageType commend.

Tips for testing and automation

It's a challenge to keep track of all related settings, capabilities, and limitations while authoring a template. The common approach is to develop and test templates against a single cloud before other locations are targeted. However, the earlier that tests are performed in the authoring process, the less troubleshooting and code rewriting your development team will have to do. Deployments that fail because of location dependencies can be time-consuming to troubleshoot. That's why we recommend automated testing as early as possible in the authoring cycle. Ultimately, you'll need less development time and fewer resources, and your cloud-consistent artifacts will become even more valuable.

The following image shows a typical example of a development process for a team using an integrated development environment (IDE). At different stages in the timeline, different test types are executed. Here, two developers are working on the same solution, but this scenario applies equally to a single developer or a large team. Each developer typically creates a local copy of a central repository, enabling each one to work on the local copy without impacting the others who may be working on the same files.



Consider the following tips for testing and automation:

- Do make use of testing tools. For example, Visual Studio Code and Visual Studio include IntelliSense and other features that can help you validate your templates.

- To improve the code quality during development on the local IDE, perform static code analysis with unit tests and integration tests.
- For an even better experience during initial development, unit tests and integration tests should only warn when an issue is found and proceed with the tests. That way, you can identify the issues to addressed and prioritize the order of the changes, also referred to as test-driven deployment (TDD).
- Be aware that some tests can be performed without being connected to Azure Resource Manager. Others, like testing template deployment, require Resource Manager to perform certain actions that cannot be performed offline.
- Testing a deployment template against the validation API isn't equal to an actual deployment. Also, even if you deploy a template from a local file, any references to nested templates in the template are retrieved by Resource Manager directly, and artifacts referenced by VM extensions are retrieved by the VM agent running inside the deployed VM.

Next steps

- Azure Resource Manager template considerations
- Best practices for Azure Resource Manager templates

Azure Resource Manager deployment modes

7/2/2019 • 3 minutes to read • Edit Online

When deploying your resources, you specify that the deployment is either an incremental update or a complete update. The primary difference between these two modes is how Resource Manager handles existing resources in the resource group that aren't in the template. The default mode is incremental.

For both modes, Resource Manager tries to create all resources specified in the template. If the resource already exists in the resource group and its settings are unchanged, no operation is taken for that resource. If you change the property values for a resource, the resource is updated with those new values. If you try to update the location or type of an existing resource, the deployment fails with an error. Instead, deploy a new resource with the location or type that you need.

Complete mode

In complete mode, Resource Manager **deletes** resources that exist in the resource group but aren't specified in the template. Resources that are specified in the template, but not deployed because a condition evaluates to false, aren't deleted.

Be careful using complete mode with copy loops. Any resources that aren't specified in the template after resolving the copy loop are deleted.

There are some differences in how resource types handle complete mode deletions. Parent resources are automatically deleted when not in a template that's deployed in complete mode. Some child resources aren't automatically deleted when not in the template. However, these child resources are deleted if the parent resource is deleted.

For example, if your resource group contains a DNS zone (Microsoft.Network/dnsZones resource type) and a CNAME record (Microsoft.Network/dnsZones/CNAME resource type), the DNS zone is the parent resource for the CNAME record. If you deploy with complete mode and don't include the DNS zone in your template, the DNS zone and the CNAME record are both deleted. If you include the DNS zone in your template but don't include the CNAME record, the CNAME isn't deleted.

For a list of how resource types handle deletion, see Deletion of Azure resources for complete mode deployments.

If the resource group is locked, complete mode doesn't delete the resources.

> **NOTE**
>
> Only root-level templates support the complete deployment mode. For linked or nested templates, you must use incremental mode.
>
> Subscription level deployments don't support complete mode.
>
> Currently, the portal doesn't support complete mode.

Incremental mode

In incremental mode, Resource Manager **leaves unchanged** resources that exist in the resource group but aren't specified in the template.

However, when redeploying an existing resource in incremental mode, the outcome is a different. Specify all properties for the resource, not just the ones you're updating. A common misunderstanding is to think properties that are not specified are left unchanged. If you don't specify certain properties, Resource Manager interprets the update as overwriting those values.

Example result

To illustrate the difference between incremental and complete modes, consider the following scenario.

Resource Group contains:

- Resource A
- Resource B
- Resource C

Template contains:

- Resource A
- Resource B
- Resource D

When deployed in **incremental** mode, the resource group has:

- Resource A
- Resource B
- Resource C
- Resource D

When deployed in **complete** mode, Resource C is deleted. The resource group has:

- Resource A
- Resource B
- Resource D

Set deployment mode

To set the deployment mode when deploying with PowerShell, use the `Mode` parameter.

```
New-AzResourceGroupDeployment `
  -Mode Complete `
  -Name ExampleDeployment `
  -ResourceGroupName ExampleResourceGroup `
  -TemplateFile c:\MyTemplates\storage.json
```

To set the deployment mode when deploying with Azure CLI, use the `mode` parameter.

```
az group deployment create \
  --name ExampleDeployment \
  --mode Complete \
  --resource-group ExampleGroup \
  --template-file storage.json \
  --parameters storageAccountType=Standard_GRS
```

The following example shows a linked template set to incremental deployment mode:

```
"resources": [
  {
      "apiVersion": "2017-05-10",
      "name": "linkedTemplate",
      "type": "Microsoft.Resources/deployments",
      "properties": {
          "mode": "Incremental",
          <nested-template-or-external-template>
      }
  }
]
```

Next steps

- To learn about creating Resource Manager templates, see Authoring Azure Resource Manager templates.
- To learn about deploying resources, see Deploy an application with Azure Resource Manager template.
- To view the operations for a resource provider, see Azure REST API.

Azure Resource Manager resource group deletion

6/18/2019 • 2 minutes to read • Edit Online

This article describes how Azure Resource Manager orders the deletion of resources when you delete a resource group.

Determine order of deletion

When you delete a resource group, Resource Manager determines the order to delete resources. It uses the following order:

1. All the child (nested) resources are deleted.
2. Resources that manage other resources are deleted next. A resource can have the `managedBy` property set to indicate that a different resource manages it. When this property is set, the resource that manages the other resource is deleted before the other resources.
3. The remaining resources are deleted after the previous two categories.

Resource deletion

After the order is determined, Resource Manager issues a DELETE operation for each resource. It waits for any dependencies to finish before proceeding.

For synchronous operations, the expected successful response codes are:

- 200
- 204
- 404

For asynchronous operations, the expected successful response is 202. Resource Manager tracks the location header or the azure-async operation header to determine the status of the asynchronous delete operation.

Errors

When a delete operation returns an error, Resource Manager retries the DELETE call. Retries happen for the 5xx, 429 and 408 status codes. By default, the time period for retry is 15 minutes.

After deletion

Resource Manager issues a GET call on each resource that it tried to delete. The response of this GET call is expected to be 404. When Resource Manager gets a 404, it considers the deletion to have completed successfully. Resource Manager removes the resource from its cache.

However, if the GET call on the resource returns a 200 or 201, Resource Manager recreates the resource.

Errors

If the GET operation returns an error, Resource Manager retries the GET for the following error code:

- Less than 100
- 408
- 429
- Greater than 500

For other error codes, Resource Manager fails the deletion of the resource.

Next steps

- To understand Resource Manager concepts, see Azure Resource Manager overview.
- For deletion commands, see PowerShell, Azure CLI, and REST API.

Enable safe deployment practices with Azure Deployment Manager (Public preview)

6/11/2019 • 7 minutes to read • Edit Online

To deploy your service across many regions and make sure it's running as expected in each region, you can use Azure Deployment Manager to coordinate a staged rollout of the service. Just as you would for any Azure deployment, you define the resources for your service in Resource Manager templates. After creating the templates, you use Deployment Manager to describe the topology for your service and how it should be rolled out.

Deployment Manager is a feature of Resource Manager. It expands your capabilities during deployment. Use Deployment Manager when you have a complex service that needs to be deployed to several regions. By staging the rollout of your service, you can find potential problems before it has been deployed to all regions. If you don't need the extra precautions of a staged rollout, use the standard deployment options for Resource Manager. Deployment Manager seamlessly integrates with all existing third-party tools that support Resource Manager deployments, such as continuous integration and continuous delivery (CI/CD) offerings.

Azure Deployment Manager is in preview. Help us improve the feature by providing feedback.

To use Deployment Manager, you need to create four files:

- Topology template
- Rollout template
- Parameter file for topology
- Parameter file for rollout

You deploy the topology template before deploying the rollout template.

Additional resources:

- The Azure Deployment Manager REST API reference.
- Tutorial: Use Azure Deployment Manager with Resource Manager templates.
- Tutorial: Use health check in Azure Deployment Manager.
- An Azure Deployment Manager sample.

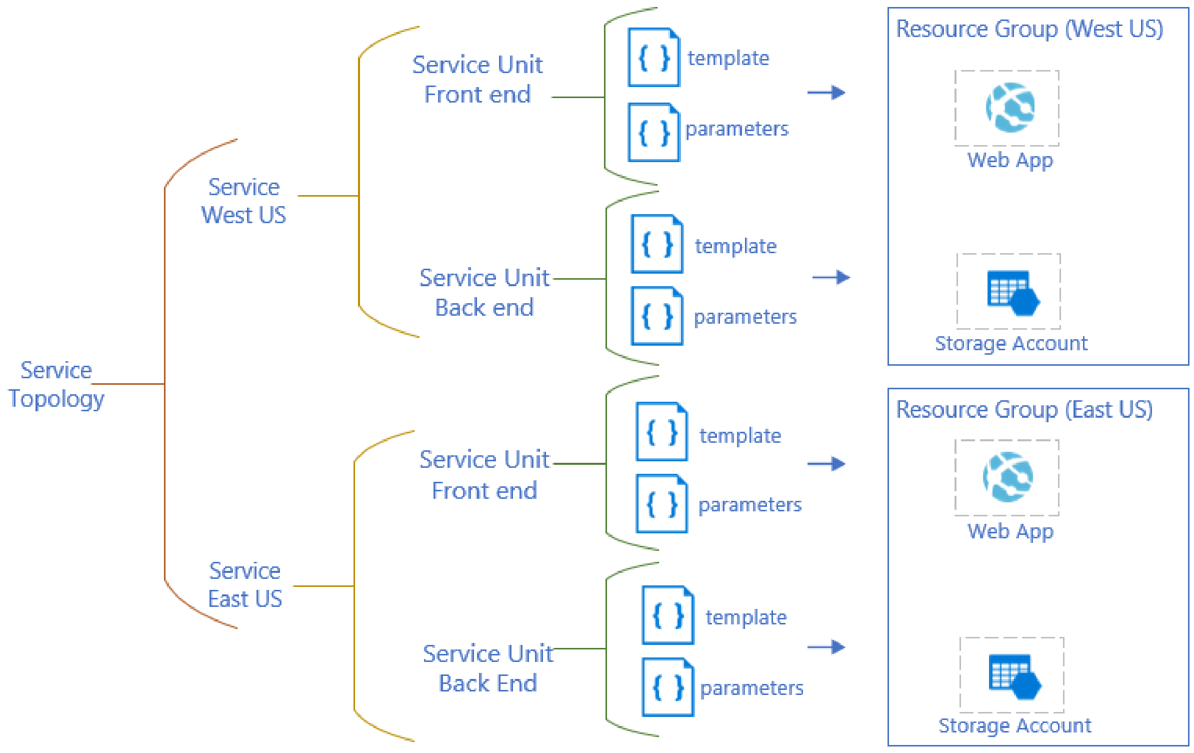
Identity and access

With Deployment Manager, a user-assigned managed identity performs the deployment actions. You create this identity before starting your deployment. It must have access to the subscription you're deploying the service to, and sufficient permission to complete the deployment. For information about the actions granted through roles, see Built-in roles for Azure resources.

The identity must reside in the same location as the rollout.

Topology template

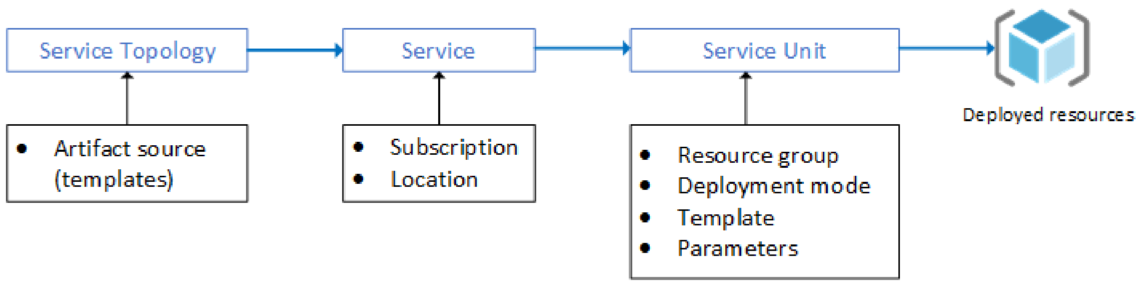
The topology template describes the Azure resources that make up your service and where to deploy them. The following image shows the topology for an example service:



The topology template includes the following resources:

- Artifact source - where your Resource Manager templates and parameters are stored
- Service topology - points to artifact source
  - Services - specifies location and Azure subscription ID
    - Service units - specifies resource group, deployment mode, and path to template and parameter file

To understand what happens at each level, it's helpful to see which values you provide.



Artifact source for templates

In your topology template, you create an artifact source that holds the templates and parameters files. The artifact source is a way to pull the files for deployment. You'll see another artifact source for binaries later in this article.

The following example shows the general format of the artifact source.

```
{
 "type": "Microsoft.DeploymentManager/artifactSources",
    "name": "<artifact-source-name>",
 "location": "<artifact-source-location>",
 "apiVersion": "2018-09-01-preview",
 "properties": {
  "sourceType": "AzureStorage",
  "artifactRoot": "<root-folder-for-templates>",
  "authentication": {
   "type": "SAS",
   "properties": {
    "sasUri": "<SAS-URI-for-storage-container>"
   }
  }
 }
}
```

For more information, see artifactSources template reference.

Service topology

The following example shows the general format of the service topology resource. You provide the resource ID of the artifact source that holds the templates and parameter files. The service topology includes all service resources. To make sure the artifact source is available, the service topology depends on it.

```
{
    "type": "Microsoft.DeploymentManager/serviceTopologies",
 "name": "<topology-name>",
 "location": "<topology-location>",
 "apiVersion": "2018-09-01-preview",
 "properties": {
  "artifactSourceId": "<resource-ID-artifact-source>"
 },
 "dependsOn": [
  "<artifact-source>"
 ],
 "resources": [
  {
   "type": "services",
      ...
        }
    ]
}
```

For more information, see serviceTopologies template reference.

Services

The following example shows the general format of the services resource. In each service, you provide the location and Azure subscription ID to use for deploying your service. To deploy to several regions, you define a service for each region. The service depends on the service topology.

```
{
    "type": "services",
 "name": "<service-name>",
 "location": "<service-location>",
 "apiVersion": "2018-09-01-preview",
 "dependsOn": [
     "<service-topology>"
 ],
 "properties": {
  "targetSubscriptionId": "<subscription-ID>",
  "targetLocation": "<location-of-deployed-service>"
 },
 "resources": [
  {
   "type": "serviceUnits",
           ...
        }
    ]
}
```

For more information, see services template reference.

Service Units

The following example shows the general format of the service units resource. In each service unit, you specify the resource group, the deployment mode to use for deployment, and the path to the template and parameter file. If you specify a relative path for the template and parameters, the full path is constructed from the root folder in the artifacts source. You can specify an absolute path for the template and parameters, but you lose the ability to easily version your releases. The service unit depends on the service.

```
{
 "type": "serviceUnits",
 "name": "<service-unit-name>",
 "location": "<service-unit-location>",
 "apiVersion": "2018-09-01-preview",
 "dependsOn": [
  "<service>"
 ],
 "tags": {
  "serviceType": "Service West US Web App"
 },
 "properties": {
  "targetResourceGroup": "<resource-group-name>",
  "deploymentMode": "Incremental",
  "artifacts": {
   "templateArtifactSourceRelativePath": "<relative-path-to-template>",
   "parametersArtifactSourceRelativePath": "<relative-path-to-parameter-file>"
  }
 }
}
```

Each template should include the related resources that you want to deploy in one step. For example, a service unit could have a template that deploys all of the resources for your service's front end.

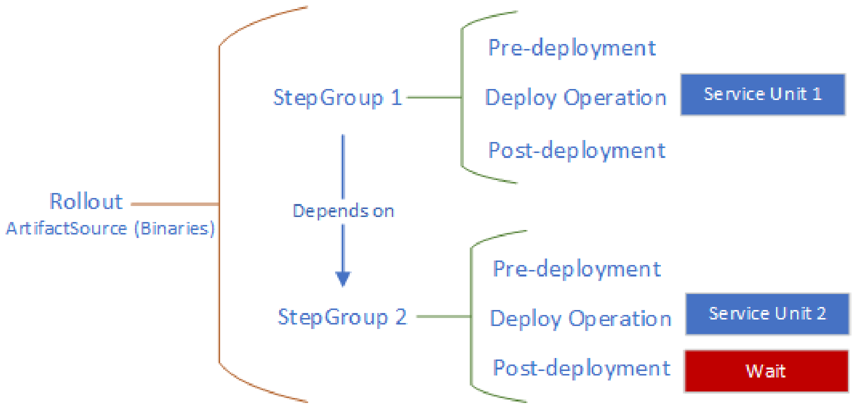
For more information, see serviceUnits template reference.

Rollout template

The rollout template describes the steps to take when deploying your service. You specify the service topology to use and define the order for deploying service units. It includes an artifact source for storing binaries for the deployment. In your rollout template, you define the following hierarchy:

- Artifact source
- Step
- Rollout
  - Step groups
    - Deployment operations

The following image shows the hierarchy of the rollout template:



Each rollout can have many step groups. Each step group has one deployment operation that points to a service unit in the service topology.

Artifact source for binaries

In the rollout template, you create an artifact source for the binaries you need to deploy to the service. This artifact source is similar to the artifact source for templates, except that it contains the scripts, web pages, compiled code, or other files needed by your service.

Steps

You can define a step to perform either before or after your deployment operation. Currently, only the `wait` step and the 'healthCheck' step are available.

The wait step pauses the deployment before continuing. It allows you to verify that your service is running as expected before deploying the next service unit. The following example shows the general format of a wait step.

```
{
    "apiVersion": "2018-09-01-preview",
    "type": "Microsoft.DeploymentManager/steps",
    "name": "waitStep",
     "location": "<step-location>",
    "properties": {
        "stepType": "wait",
        "attributes": {
          "duration": "PT1M"
        }
    }
},
```

The duration property uses ISO 8601 standard. The preceding example specifies a one-minute wait.

For more information about the health check step, see Introduce health integration rollout to Azure Deployment Manager and Tutorial: Use health check in Azure Deployment Manager.

For more information, see steps template reference.

Rollouts

To make sure the artifact source is available, the rollout depends on it. The rollout defines steps groups for each service unit that is deployed. You can define actions to take before or after deployment. For example, you can specify that the deployment wait after the service unit has been deployed. You can define the order of the step groups.

The identity object specifies the user-assigned managed identity that performs the deployment actions.

The following example shows the general format of the rollout.

```
{
 "type": "Microsoft.DeploymentManager/rollouts",
 "name": "<rollout-name>",
 "location": "<rollout-location>",
 "apiVersion": "2018-09-01-preview",
 "Identity": {
  "type": "userAssigned",
  "identityIds": [
   "<managed-identity-ID>"
  ]
 },
    "dependsOn": [
  "<artifact-source>"
    ],
 "properties": {
  "buildVersion": "1.0.0.0",
  "artifactSourceId": "<artifact-source-ID>",
  "targetServiceTopologyId": "<service-topology-ID>",
  "stepGroups": [
   {
    "name": "stepGroup1",
                "dependsOnStepGroups": ["<step-group-name>"],
    "preDeploymentSteps": ["<step-ID>"],
    "deploymentTargetId":
     "<service-unit-ID>",
    "postDeploymentSteps": ["<step-ID>"]
   },
            ...
        ]
    }
}
```

For more information, see rollouts template reference.

Parameter file

You create two parameter files. One parameter file is used when deploying the service topology, and the other is used for the rollout deployment. There are some values that you need to make sure are the same in both parameter files.

containerRoot variable

With versioned deployments, the path to your artifacts changes with each new version. The first time you run a deployment the path might be `https://<base-uri-blob-container>/binaries/1.0.0.0`. The second time it might be `https://<base-uri-blob-container>/binaries/1.0.0.1`. Deployment Manager simplifies getting the correct root path for the current deployment by using the `$containerRoot` variable. This value changes with each version and isn't known before deployment.

Use the `$containerRoot` variable in the parameter file for template to deploy the Azure resources. At deployment time, this variable is replaced with the actual values from the rollout.

For example, during rollout you create an artifact source for the binary artifacts.

```
{
 "type": "Microsoft.DeploymentManager/artifactSources",
    "name": "[variables('rolloutArtifactSource').name]",
 "location": "[parameters('azureResourceLocation')]",
 "apiVersion": "2018-09-01-preview",
 "properties": {
  "sourceType": "AzureStorage",
        "artifactRoot": "[parameters('binaryArtifactRoot')]",
     "authentication" :
  {
   "type": "SAS",
   "properties": {
    "sasUri": "[parameters('artifactSourceSASLocation')]"
   }
  }
 }
},
```

Notice the `artifactRoot` and `sasUri` properties. The artifact root might be set to a value like `binaries/1.0.0.0`. The SAS URI is the URI to your storage container with a SAS token for access. Deployment Manager automatically constructs the value of the `$containerRoot` variable. It combines those values in the format `<container>/<artifactRoot>`.

Your template and parameter file need to know the correct path for getting the versioned binaries. For example, to deploy files for a web app, create the following parameter file with the $containerRoot variable. You must use two backslashes (`\\`) for the path because the first is an escape character.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "deployPackageUri": {
            "value": "$containerRoot\\helloWorldWebAppWUS.zip"
        }
    }
}
```

Then, use that parameter in your template:

```
{
    "name": "MSDeploy",
    "type": "extensions",
    "location": "[parameters('location')]",
    "apiVersion": "2015-08-01",
    "dependsOn": [
        "[concat('Microsoft.Web/sites/', parameters('WebAppName'))]"
    ],
    "tags": {
        "displayName": "WebAppMSDeploy"
    },
    "properties": {
        "packageUri": "[parameters('deployPackageURI')]"
    }
}
```

You manage versioned deployments by creating new folders and passing in that root during rollout. The path flows through to the template that deploys the resources.

Next steps

In this article, you learned about Deployment Manager. Proceed to the next article to learn how to deploy with Deployment Manager.

Tutorial: Use Azure Deployment Manager with Resource Manager templates

Introduce health integration rollout to Azure Deployment Manager (Public preview)

7/5/2019 • 5 minutes to read • Edit Online

Azure Deployment Manager allows you to perform staged rollouts of Azure Resource Manager resources. The resources are deployed region by region in an ordered fashion. The integrated health check of Azure Deployment Manager can monitor rollouts, and automatically stop problematic rollouts, so that you can troubleshoot and reduce the scale of the impact. This feature can reduce service unavailability caused by regressions in updates.

Health monitoring providers

In order to make health integration as easy as possible, Microsoft has been working with some of the top service health monitoring companies to provide you with a simple copy/paste solution to integrate health checks with your deployments. If you're not already using a health monitor, these are great solutions to start with:

Datadog, the leading monitoring and analytics platform for modern cloud environments. See how Datadog integrates with Azure Deployment Manager.	Site24x7, the all-in-one private and public cloud services monitoring solution. See how Site24x7 integrates with Azure Deployment Manager.	Wavefront, the monitoring and analytics platform for multi-cloud application environments. See how Wavefront integrates with Azure Deployment Manager.

How service health is determined

Health monitoring providers offer several mechanisms for monitoring services and alerting you of any service health issues. Azure Monitor is an example of one such offering. Azure Monitor can be used to create alerts when certain thresholds are exceeded. For example, your memory and CPU utilization spike beyond expected levels when you deploy a new update to your service. When notified, you can take corrective actions.

These health providers typically offer REST APIs so that the status of your service's monitors can be examined programmatically. The REST APIs can either come back with a simple healthy/unhealthy signal (determined by the HTTP response code), and/or with detailed information about the signals it is receiving.

The new *healthCheck* step in Azure Deployment Manager allows you to declare HTTP codes that indicate a healthy service, or, for more complex REST results, you can even specify regular expressions that, if they match, indicate a healthy response.

The flow to getting setup with Azure Deployment Manager health checks:

1. Create your health monitors via a health service provider of your choice.
2. Create one or more healthCheck steps as part of your Azure Deployment Manager rollout. Fill out the healthCheck steps with the following information:

   a. The URI for the REST API for your health monitors (as defined by your health service provider).
   b. Authentication information. Currently only API-key style authentication is supported.
   c. HTTP status codes or regular expressions that define a healthy response. Note that you may provide regular expressions, which ALL must match for the response to be considered healthy, or you may provide expressions of which ANY must match for the response to be considered healthy. Both methods are supported.

The following Json is an example:

```
{
  "type": "Microsoft.DeploymentManager/steps",
  "apiVersion": "2018-09-01-preview",
  "name": "healthCheckStep",
   "location": "[parameters('azureResourceLocation')]",
  "properties": {
    "stepType": "healthCheck",
    "attributes": {
      "waitDuration": "PT0M",
      "maxElasticDuration": "PT0M",
      "healthyStateDuration": "PT1M",
      "type": "REST",
      "properties": {
        "healthChecks": [
          {
            "name": "appHealth",
            "request": {
              "method": "GET",
              "uri": "[parameters('healthCheckUrl')]",
              "authentication": {
                "type": "ApiKey",
                "name": "code",
                "in": "Query",
                "value": "[parameters('healthCheckAuthAPIKey')]"
              }
            },
            "response": {
              "successStatusCodes": [
                "200"
              ],
              "regex": {
                "matches": [
                  "Status: healthy",
                  "Status: warning"
                ],
                "matchQuantifier": "Any"
              }
            }
          }
        ]
      }
    }
  }
},
```

3. Invoke the healthCheck steps at the appropriate time in your Azure Deployment Manager rollout. In the following example, a health check step is invoked in **postDeploymentSteps** of **stepGroup2**.

```
"stepGroups": [
    {
        "name": "stepGroup1",
        "preDeploymentSteps": [],
        "deploymentTargetId": "
[resourceId('Microsoft.DeploymentManager/serviceTopologies/services/serviceUnits',
variables('serviceTopology').name, variables('serviceTopology').serviceWUS.name,
variables('serviceTopology').serviceWUS.serviceUnit2.name)]",
        "postDeploymentSteps": []
    },
    {
        "name": "stepGroup2",
        "dependsOnStepGroups": ["stepGroup1"],
        "preDeploymentSteps": [],
        "deploymentTargetId": "
[resourceId('Microsoft.DeploymentManager/serviceTopologies/services/serviceUnits',
variables('serviceTopology').name, variables('serviceTopology').serviceWUS.name,
variables('serviceTopology').serviceWUS.serviceUnit1.name)]",
        "postDeploymentSteps": [
            {
                "stepId": "[resourceId('Microsoft.DeploymentManager/steps/', 'healthCheckStep')]"
            }
        ]
    },
    {
        "name": "stepGroup3",
        "dependsOnStepGroups": ["stepGroup2"],
        "preDeploymentSteps": [],
        "deploymentTargetId": "
[resourceId('Microsoft.DeploymentManager/serviceTopologies/services/serviceUnits',
variables('serviceTopology').name, variables('serviceTopology').serviceEUS.name,
variables('serviceTopology').serviceEUS.serviceUnit2.name)]",
        "postDeploymentSteps": []
    },
    {
        "name": "stepGroup4",
        "dependsOnStepGroups": ["stepGroup3"],
        "preDeploymentSteps": [],
        "deploymentTargetId": "
[resourceId('Microsoft.DeploymentManager/serviceTopologies/services/serviceUnits',
variables('serviceTopology').name, variables('serviceTopology').serviceEUS.name,
variables('serviceTopology').serviceEUS.serviceUnit1.name)]",
        "postDeploymentSteps": []
    }
]
```

To walk through an example, see Tutorial: Use health check in Azure Deployment Manager.

Phases of a health check

At this point Azure Deployment Manager knows how to query for the health of your service and at what phases in your rollout to do so. However, Azure Deployment Manager also allows for deep configuration of the timing of these checks. A healthCheck step is executed in 3 sequential phases, all of which have configurable durations:

1. Wait

    a. After a deployment operation is completed, VMs may be rebooting, reconfiguring based on new data, or even being started for the first time. It also takes time for services to start emitting health signals to be aggregated by the health monitoring provider into something useful. During this tumultuous process, it may not make sense to check for service health since the update has not yet reached a steady state. Indeed, the service may be oscillating between healthy and unhealthy states as the resources settle.

    b. During the Wait phase, service health is not monitored. This is used to allow the deployed resources the time to bake before beginning the health check process.

2. Elastic

    a. Since it is impossible to know in all cases how long resources will take to bake before they become stable, the Elastic phase allows for a flexible time period between when the resources are potentially unstable and when they are required to maintain a healthy steady state.
    b. When the Elastic phase begins, Azure Deployment Manager begins polling the provided REST endpoint for service health periodically. The polling interval is configurable.
    c. If the health monitor comes back with signals indicating that the service is unhealthy, these signals are ignored, the Elastic phase continues, and polling continues.
    d. As soon as the health monitor comes back with signals indicating that the service is healthy, the Elastic phase ends and the HealthyState phase begins.
    e. Thus, the duration specified for the Elastic phase is the maximum amount of time that can be spent polling for service health before a healthy response is considered mandatory.

3. HealthyState

    a. During the HealthyState phase, service health is continually polled at the same interval as the Elastic phase.
    b. The service is expected to maintain healthy signals from the health monitoring provider for the entire specified duration.
    c. If at any point an unhealthy response is detected, Azure Deployment Manager will stop the entire rollout and return the REST response carrying the unhealthy service signals.
    d. Once the HealthyState duration has ended, the healthCheck is complete, and deployment continues to the next step.

Next steps

In this article, you learned about how to integrate health monitoring in Azure Deployment Manager. Proceed to the next article to learn how to deploy with Deployment Manager.

Tutorial: integrate health check in Azure Deployment Manager

Azure Resource Manager vs. classic deployment: Understand deployment models and the state of your resources

6/18/2019 • 10 minutes to read • Edit Online

NOTE

The information provided in this article is only used when you migrate from the classic deployment to the Azure Resource Manager deployment.

In this article, you learn about Azure Resource Manager and classic deployment models. The Resource Manager and classic deployment models represent two different ways of deploying and managing your Azure solutions. You work with them through two different API sets, and the deployed resources can contain important differences. The two models are not compatible with each other. This article describes those differences.

To simplify the deployment and management of resources, Microsoft recommends that you use Resource Manager for all new resources. If possible, Microsoft recommends that you redeploy existing resources through Resource Manager.

If you are new to Resource Manager, you may want to first review the terminology defined in the Azure Resource Manager overview.

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

History of the deployment models

Azure originally provided only the classic deployment model. In this model, each resource existed independently; there was no way to group related resources together. Instead, you had to manually track which resources made up your solution or application, and remember to manage them in a coordinated approach. To deploy a solution, you had to either create each resource individually through the portal or create a script that deployed all the resources in the correct order. To delete a solution, you had to delete each resource individually. You could not easily apply and update access control policies for related resources. Finally, you could not apply tags to resources to label them with terms that help you monitor your resources and manage billing.

In 2014, Azure introduced Resource Manager, which added the concept of a resource group. A resource group is a container for resources that share a common lifecycle. The Resource Manager deployment model provides several benefits:

- You can deploy, manage, and monitor all the services for your solution as a group, rather than handling these services individually.
- You can repeatedly deploy your solution throughout its lifecycle and have confidence your resources are deployed in a consistent state.
- You can apply access control to all resources in your resource group, and those policies are automatically

applied when new resources are added to the resource group.
- You can apply tags to resources to logically organize all the resources in your subscription.
- You can use JavaScript Object Notation (JSON) to define the infrastructure for your solution. The JSON file is known as a Resource Manager template.
- You can define the dependencies between resources so they are deployed in the correct order.

When Resource Manager was added, all resources were retroactively added to default resource groups. If you create a resource through classic deployment now, the resource is automatically created within a default resource group for that service, even though you did not specify that resource group at deployment. However, just existing within a resource group does not mean that the resource has been converted to the Resource Manager model.

Understand support for the models

There are three scenarios to be aware of:

1. Cloud Services does not support Resource Manager deployment model.
2. Virtual machines, storage accounts, and virtual networks support both Resource Manager and classic deployment models.
3. All other Azure services support Resource Manager.

For virtual machines, storage accounts, and virtual networks, if the resource was created through classic deployment, you must continue to operate on it through classic operations. If the virtual machine, storage account, or virtual network was created through Resource Manager deployment, you must continue using Resource Manager operations. This distinction can get confusing when your subscription contains a mix of resources created through Resource Manager and classic deployment. This combination of resources can create unexpected results because the resources do not support the same operations.

In some cases, a Resource Manager command can retrieve information about a resource created through classic deployment, or can perform an administrative task such as moving a classic resource to another resource group. But, these cases should not give the impression that the type supports Resource Manager operations. For example, suppose you have a resource group that contains a virtual machine that was created with classic deployment. If you run the following Resource Manager PowerShell command:

```
Get-AzResource -ResourceGroupName ExampleGroup -ResourceType Microsoft.ClassicCompute/virtualMachines
```

It returns the virtual machine:

```
Name              : ExampleClassicVM
ResourceId        :
/subscriptions/{guid}/resourceGroups/ExampleGroup/providers/Microsoft.ClassicCompute/virtualMachines/ExampleCla
ssicVM
ResourceName      : ExampleClassicVM
ResourceType      : Microsoft.ClassicCompute/virtualMachines
ResourceGroupName : ExampleGroup
Location          : westus
SubscriptionId    : {guid}
```

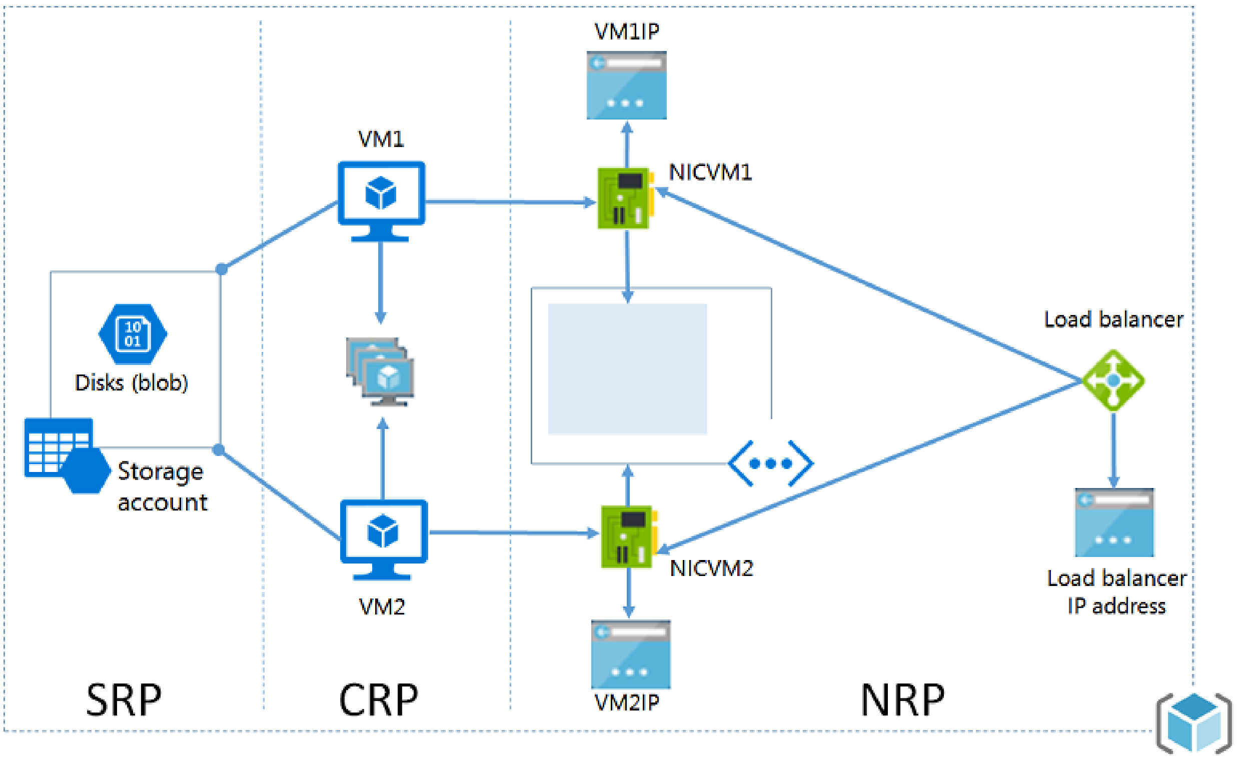
However, the Resource Manager cmdlet **Get-AzVM** only returns virtual machines deployed through Resource Manager. The following command does not return the virtual machine created through classic deployment.

```
Get-AzVM -ResourceGroupName ExampleGroup
```

Only resources created through Resource Manager support tags. You cannot apply tags to classic resources.

Changes for compute, network, and storage

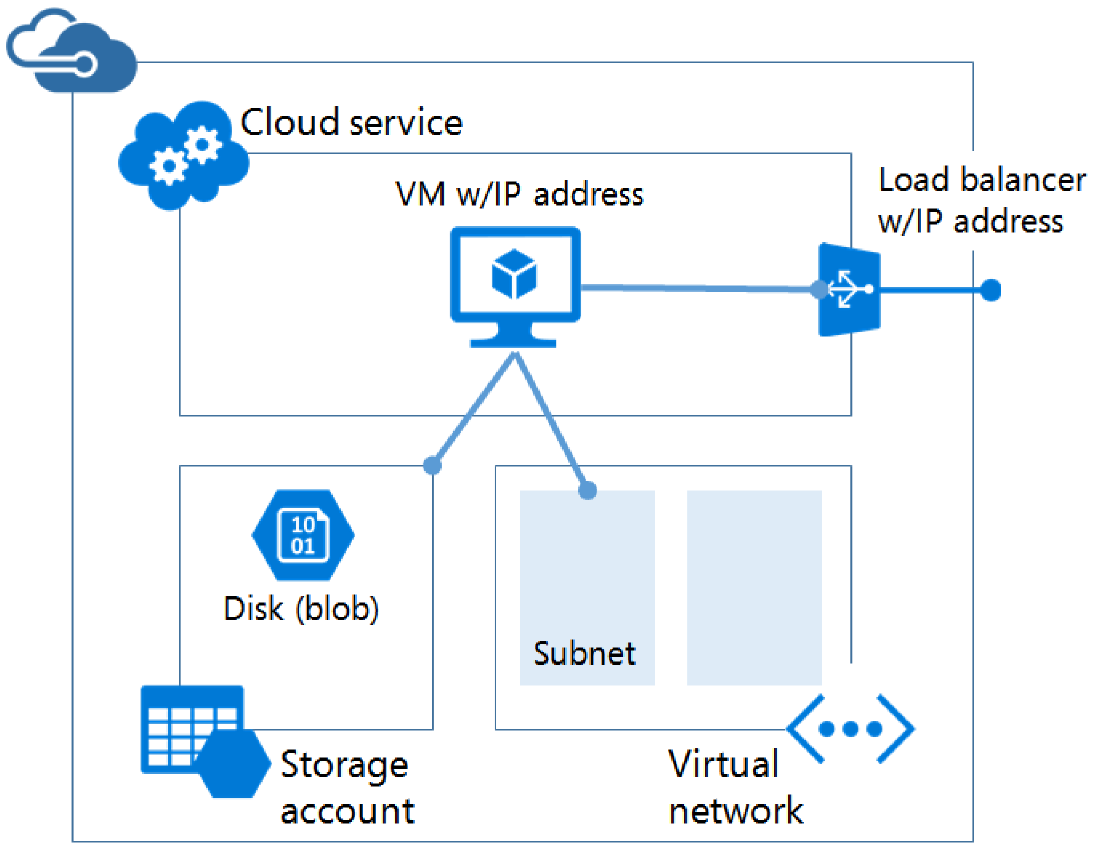
The following diagram displays compute, network, and storage resources deployed through Resource Manager.



Note the following relationships between the resources:

- All the resources exist within a resource group.
- The virtual machine depends on a specific storage account defined in the Storage resource provider to store its disks in blob storage (required).
- The virtual machine references a specific NIC defined in the Network resource provider (required) and an availability set defined in the Compute resource provider (optional).
- The NIC references the virtual machine's assigned IP address (required), the subnet of the virtual network for the virtual machine (required), and to a Network Security Group (optional).
- The subnet within a virtual network references a Network Security Group (optional).
- The load balancer instance references the backend pool of IP addresses that include the NIC of a virtual machine (optional) and references a load balancer public or private IP address (optional).

Here are the components and their relationships for classic deployment:



The classic solution for hosting a virtual machine includes:

- A required cloud service that acts as a container for hosting virtual machines (compute). Virtual machines are automatically provided with a network interface card (NIC) and an IP address assigned by Azure. Additionally, the cloud service contains an external load balancer instance, a public IP address, and default endpoints to allow remote desktop and remote PowerShell traffic for Windows-based virtual machines and Secure Shell (SSH) traffic for Linux-based virtual machines.
- A required storage account that stores the VHDs for a virtual machine, including the operating system, temporary, and additional data disks (storage).
- An optional virtual network that acts as an additional container, in which you can create a subnetted structure and designate the subnet on which the virtual machine is located (network).

The following table describes changes in how Compute, Network, and Storage resource providers interact:

ITEM	CLASSIC	RESOURCE MANAGER
Cloud Service for Virtual Machines	Cloud Service was a container for holding the virtual machines that required Availability from the platform and Load Balancing.	Cloud Service is no longer an object required for creating a Virtual Machine using the new model.
Virtual Networks	A virtual network is optional for the virtual machine. If included, the virtual network cannot be deployed with Resource Manager.	Virtual machine requires a virtual network that has been deployed with Resource Manager.
Storage Accounts	The virtual machine requires a storage account that stores the VHDs for the operating system, temporary, and additional data disks.	The virtual machine requires a storage account to store its disks in blob storage.

ITEM	CLASSIC	RESOURCE MANAGER
Availability Sets	Availability to the platform was indicated by configuring the same “AvailabilitySetName” on the Virtual Machines. The maximum count of fault domains was 2.	Availability Set is a resource exposed by Microsoft.Compute Provider. Virtual Machines that require high availability must be included in the Availability Set. The maximum count of fault domains is now 3.
Affinity Groups	Affinity Groups were required for creating Virtual Networks. However, with the introduction of Regional Virtual Networks, that was not required anymore.	To simplify, the Affinity Groups concept doesn’t exist in the APIs exposed through Azure Resource Manager.
Load Balancing	Creation of a Cloud Service provides an implicit load balancer for the Virtual Machines deployed.	The Load Balancer is a resource exposed by the Microsoft.Network provider. The primary network interface of the Virtual Machines that needs to be load balanced should be referencing the load balancer. Load Balancers can be internal or external. A load balancer instance references the backend pool of IP addresses that include the NIC of a virtual machine (optional) and references a load balancer public or private IP address (optional).
Virtual IP Address	Cloud Services gets a default VIP (Virtual IP Address) when a VM is added to a cloud service. The Virtual IP Address is the address associated with the implicit load balancer.	Public IP address is a resource exposed by the Microsoft.Network provider. Public IP address can be static (reserved) or dynamic. Dynamic public IPs can be assigned to a Load Balancer. Public IPs can be secured using Security Groups.
Reserved IP Address	You can reserve an IP Address in Azure and associate it with a Cloud Service to ensure that the IP Address is sticky.	Public IP Address can be created in static mode and it offers the same capability as a reserved IP address.
Public IP Address (PIP) per VM	Public IP Addresses can also be associated to a VM directly.	Public IP address is a resource exposed by the Microsoft.Network provider. Public IP Address can be static (reserved) or dynamic.
Endpoints	Input Endpoints needed to be configured on a Virtual Machine to be open up connectivity for certain ports. One of the common modes of connecting to virtual machines done by setting up input endpoints.	Inbound NAT Rules can be configured on Load Balancers to achieve the same capability of enabling endpoints on specific ports for connecting to the VMs.
DNS Name	A cloud service would get an implicit globally unique DNS Name. For example: `mycoffeeshop.cloudapp.net` .	DNS Names are optional parameters that can be specified on a Public IP Address resource. The FQDN is in the following format - `<domainlabel>.<region>.cloudapp.azure.com` .

ITEM	CLASSIC	RESOURCE MANAGER
Network Interfaces	Primary and Secondary Network Interface and its properties were defined as network configuration of a Virtual machine.	Network Interface is a resource exposed by Microsoft.Network Provider. The lifecycle of the Network Interface is not tied to a Virtual Machine. It references the virtual machine's assigned IP address (required), the subnet of the virtual network for the virtual machine (required), and to a Network Security Group (optional).

To learn about connecting virtual networks from different deployment models, see Connect virtual networks from different deployment models in the portal.

Migrate from classic to Resource Manager

If you are ready to migrate your resources from classic deployment to Resource Manager deployment, see:

1. Technical deep dive on platform-supported migration from classic to Azure Resource Manager
2. Platform supported migration of IaaS resources from Classic to Azure Resource Manager
3. Migrate IaaS resources from classic to Azure Resource Manager by using Azure PowerShell
4. Migrate IaaS resources from classic to Azure Resource Manager by using Azure CLI

Frequently asked questions

Can I create a virtual machine using Resource Manager to deploy in a virtual network created using classic deployment?

This configuration is not supported. You cannot use Resource Manager to deploy a virtual machine into a virtual network that was created using classic deployment.

Can I create a virtual machine using Resource Manager from a user image that was created using the classic deployment model?

This configuration is not supported. However, you can copy the VHD files from a storage account that was created using the classic deployment model, and add them to a new account created through Resource Manager.

What is the impact on the quota for my subscription?

The quotas for the virtual machines, virtual networks, and storage accounts created through the Azure Resource Manager are separate from other quotas. Each subscription gets quotas to create the resources using the new APIs. You can read more about the additional quotas here.

Can I continue to use my automated scripts for provisioning virtual machines, virtual networks, and storage accounts through the Resource Manager APIs?

All the automation and scripts that you've built continue to work for the existing virtual machines, virtual networks created under the Azure Service Management mode. However, the scripts have to be updated to use the new schema for creating the same resources through the Resource Manager mode.

Where can I find examples of Azure Resource Manager templates?

A comprehensive set of starter templates can be found on Azure Resource Manager Quickstart Templates.

Next steps

- To walk through the creation of template that defines a virtual machine, storage account, and virtual network, see Resource Manager template walkthrough.

- To see the commands for deploying a template, see Deploy an application with Azure Resource Manager template.

Security attributes for Azure Resource Manager

7/24/2019 • 2 minutes to read • Edit Online

This article documents the security attributes built into Azure Resource Manager.

A security attribute is a quality or feature of an Azure service. It contributes to the service's ability to prevent, detect, and respond to security vulnerabilities.

In each category, we show "Yes" or "No" to indicate whether an attribute is used. For some services, we show "N/A" for an attribute that is not applicable. We might also provide a note or a link to more information about an attribute.

Preventative

SECURITY ATTRIBUTE	YES/NO	NOTES
Encryption at rest (such as server-side encryption, server-side encryption with customer-managed keys, and other encryption features)	Yes	
Encryption in transit (such as ExpressRoute encryption, in VNet encryption, and VNet-VNet encryption)	Yes	HTTPS/TLS.
Encryption key handling (CMK, BYOK, etc.)	N/A	Azure Resource Manager stores no customer content, only control data.
Column level encryption (Azure Data Services)	Yes	
API calls encrypted	Yes	

Network segmentation

SECURITY ATTRIBUTE	YES/NO	NOTES
Service endpoint support	No	
VNet injection support	Yes	
Network isolation and firewalling support	No	
Forced tunneling support	No	

Detection

SECURITY ATTRIBUTE	YES/NO	NOTES
Azure monitoring support (Log analytics, App insights, etc.)	No	

Identity and access management

SECURITY ATTRIBUTE	YES/NO	NOTES
Authentication	Yes	Azure Active Directory based.
Authorization	Yes	

Audit trail

SECURITY ATTRIBUTE	YES/NO	NOTES
Control and management plane logging and audit	Yes	Activity logs expose all write operations (PUT, POST, DELETE) performed on your resources; see View activity logs to audit actions on resources.
Data plane logging and audit	N/A	

Configuration management

SECURITY ATTRIBUTE	YES/NO	NOTES
Configuration management support (versioning of configuration, etc.)	Yes	

Create Azure Resource Manager template

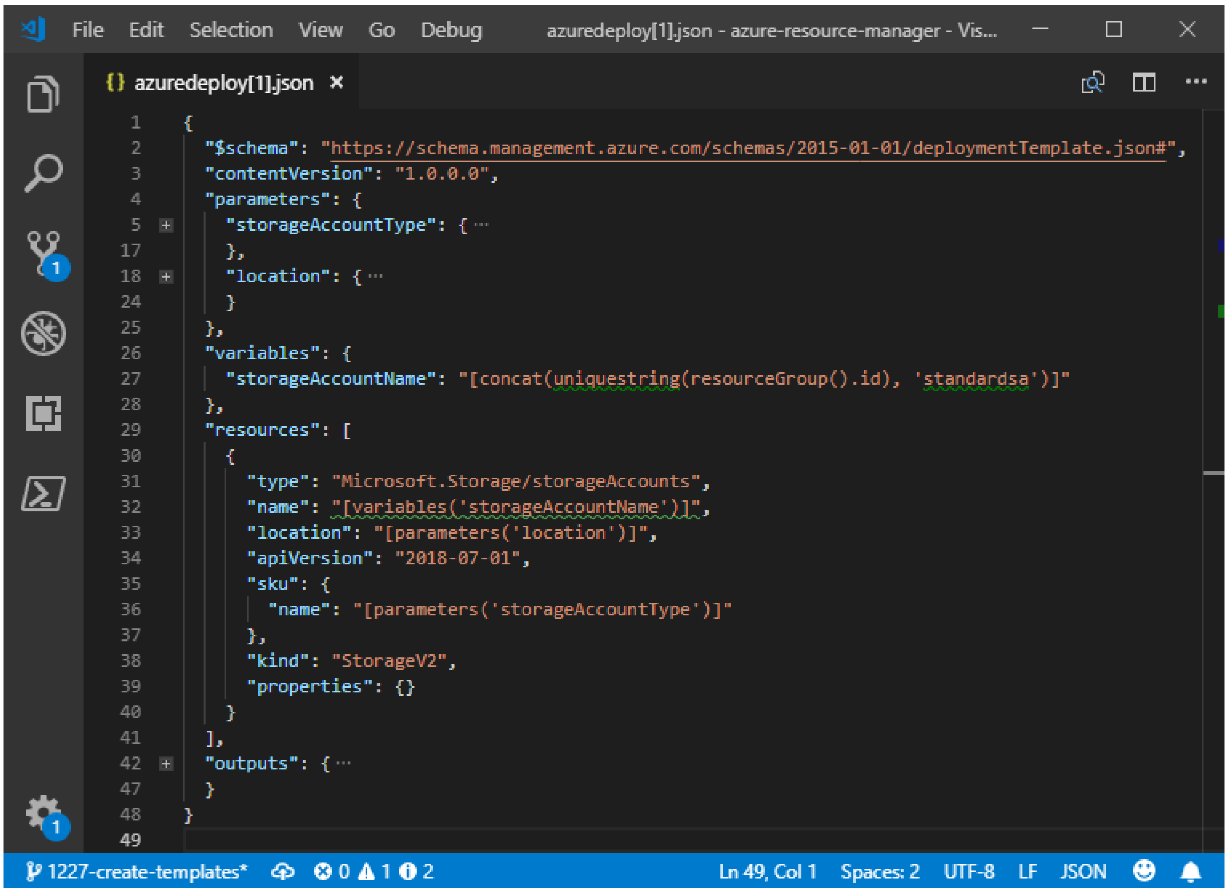
6/18/2019 • 5 minutes to read • Edit Online

This article describes the process and decisions you make when creating an Azure Resource Manager template. It provides an overview of examples and features that may help you when authoring your template. The article assumes you're deploying resources to a resource group. If you need to deploy resources to your Azure subscription, such as creating Azure Policies or role-based access control assignments, see Create resource groups and resources for an Azure subscription.

Select JSON editor

The Resource Manager template is a JSON file. You need a good authoring tool to work on the JSON file. You have many options, but if you don't already have an editor that you prefer, install Visual Studio Code (VS Code).

After installing VS Code, add the Azure Resource Manager Tools extension. This extension adds many features that simplify template authoring.


File Edit Selection View Go Debug
azuredeploy[1].json - azure-resource-manager - Vis...
azuredeploy[1].json
1227-create-templates*
0 1 2
Ln 49, Col 1 Spaces: 2 UTF-8 LF JSON

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "storageAccountType": {⋯
    },
    "location": {⋯
    }
  },
  "variables": {
    "storageAccountName": "[concat(uniquestring(resourceGroup().id), 'standardsa')]"
  },
  "resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "name": "[variables('storageAccountName')]",
      "location": "[parameters('location')]",
      "apiVersion": "2018-07-01",
      "sku": {
        "name": "[parameters('storageAccountType')]"
      },
      "kind": "StorageV2",
      "properties": {}
    }
  ],
  "outputs": {⋯
  }
}
```

The screenshot shows a Resource Manager template opened in Visual Studio Code.

For a tutorial of installing the Resource Manager tools extension and how to use VS Code, see Quickstart: Create Azure Resource Manager templates by using Visual Studio Code.

Understand the template structure

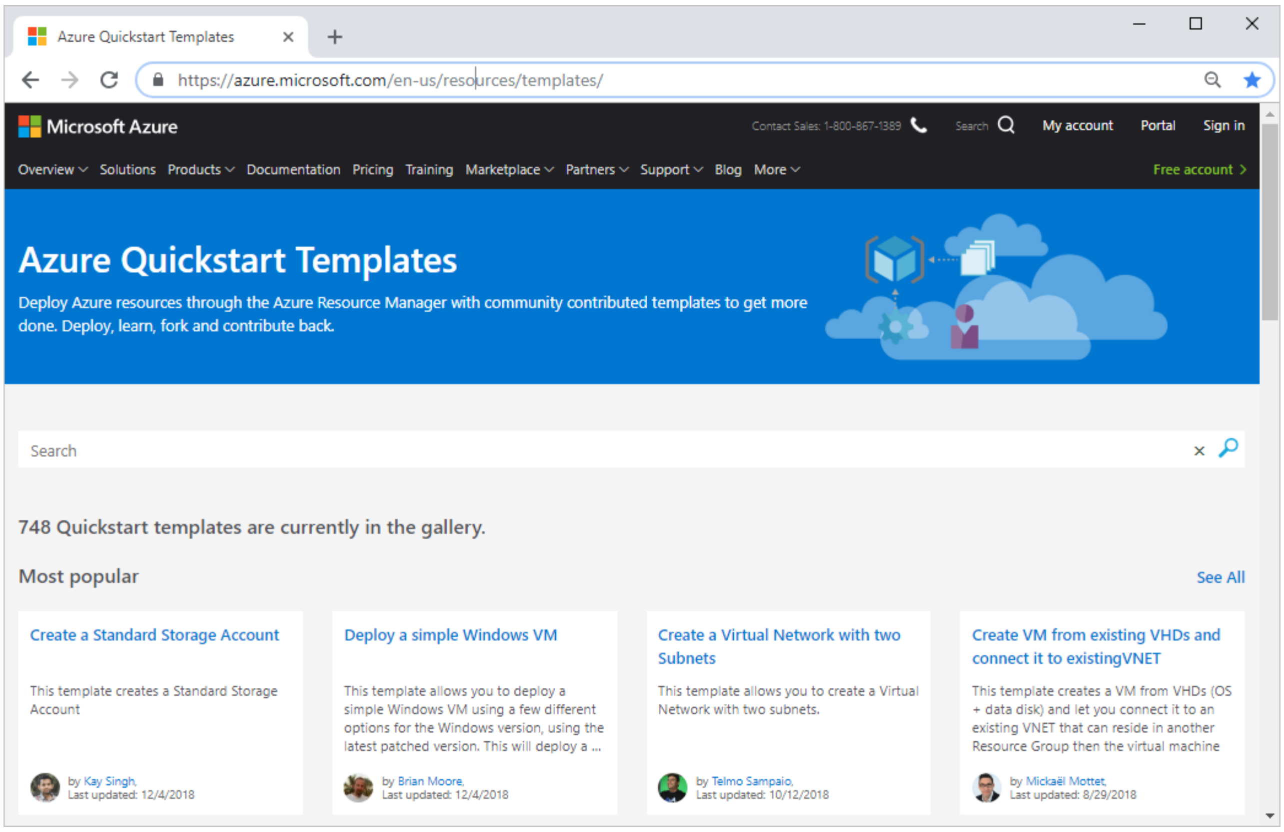
Let's review the parts of the template to understand how the template works. Your template may not have every section. The sections you want to focus on are:

- The parameters section, which shows the values you can specify during deployment to customize the infrastructure that is deployed.

- The variables section, which shows values that are used throughout the template.
- The functions section, which shows customized template expressions that are used in your template.
- The resources section, which shows the Azure resources that are deployed to your subscription.
- The outputs section, which shows the values that are returned after deployment has finished.

Look for similar templates

Often, you can find an existing template that deploys a solution that is similar to what you need. The Azure Quickstart Templates has hundreds of templates from community contributors.



Azure Quickstart Templates

Deploy Azure resources through the Azure Resource Manager with community contributed templates to get more done. Deploy, learn, fork and contribute back.

748 Quickstart templates are currently in the gallery.

Create a Standard Storage Account

This template creates a Standard Storage Account

by Kay Singh,
Last updated: 12/4/2018

Deploy a simple Windows VM

This template allows you to deploy a simple Windows VM using a few different options for the Windows version, using the latest patched version. This will deploy a ...

by Brian Moore,
Last updated: 12/4/2018

Create a Virtual Network with two Subnets

This template allows you to create a Virtual Network with two subnets.

by Telmo Sampaio,
Last updated: 10/12/2018

Create VM from existing VHDs and connect it to existingVNET

This template creates a VM from VHDs (OS + data disk) and let you connect it to an existing VNET that can reside in another Resource Group then the virtual machine

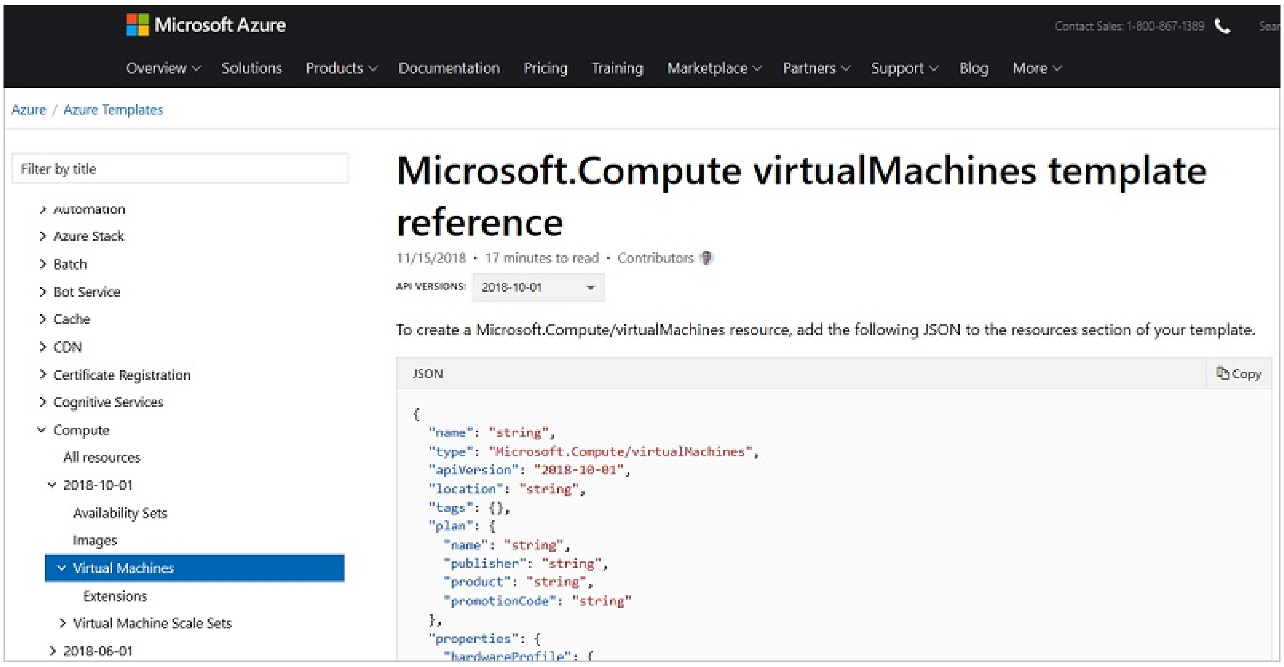
by Mickaël Mottet,
Last updated: 8/29/2018

Search through that repository for a template that is similar to what you need. It's okay if the template doesn't do exactly what you need, you can customize it.

After finding a template, select **Browse on Github**, and then copy the **azuredeploy.json** file from the repository. In VS Code, create a new file named **azuredeploy.json** and add the contents of the template file you copied from the Quickstart repository.

Add resources

You probably want to customize the template to make sure it does exactly what you want. First, review the resources that are deployed. You may need to add, remove, or change resources in the template. For descriptions and syntax of the resources, see Azure Resource Manager template reference.



Microsoft.Compute virtualMachines template reference

11/15/2018 • 17 minutes to read • Contributors

To create a Microsoft.Compute/virtualMachines resource, add the following JSON to the resources section of your template.

```
{
  "name": "string",
  "type": "Microsoft.Compute/virtualMachines",
  "apiVersion": "2018-10-01",
  "location": "string",
  "tags": {},
  "plan": {
    "name": "string",
    "publisher": "string",
    "product": "string",
    "promotionCode": "string"
  },
  "properties": {
    "hardwareProfile": {
```

After reviewing those properties, make any required changes. For recommendations about how to define resources, see resources - recommended practices.

Add or remove parameters

You might also need to adjust the parameters for your template. You can add or remove parameters based on how much customization you want to enable during deployment. For recommendations about how to use parameters, see parameters - recommended practices.

Add tags

You can add tags to your resources to logically organize by categories, and divide billing costs. Adding tags is easy, you apply them in the JSON for the resource. For example, the following storage account has two tags:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "resources": [
      {
        "apiVersion": "2016-01-01",
        "type": "Microsoft.Storage/storageAccounts",
        "name": "[concat('storage', uniqueString(resourceGroup().id))]",
        "location": "[resourceGroup().location]",
        "tags": {
          "Dept": "Finance",
          "Environment": "Production"
        },
        "sku": {
          "name": "Standard_LRS"
        },
        "kind": "Storage",
        "properties": { }
      }
    ]
}
```

You can also apply tags dynamically from parameters. For more information, see tags in template.

Review template functions

You may notice expressions in your template that are surrounded by brackets, such as `"[some-expression]"`. These expressions use template functions to dynamically construct values during deployment.

For example, you often see an expression like:

```
"name": "[parameters('siteName')]"
```

That expression gets the value of a parameter. The value is assigned to the name property.

Or, you may see a more complex expression that uses several functions like:

```
"[reference(resourceId(parameters('storageResourceGroup'), 'Microsoft.Storage/storageAccounts',
parameters('storageAccountName')), '2018-07-01')]"
```

That expression gets an object with the properties of a storage account.

To understand what the functions do, review the template function reference documentation.

Create more than one instance

Sometimes you want to create more than one instance of a resource. For example, you might need several storage accounts. Rather than repeat the resource through your template, you can use the `copy` syntax to specify more than one instance.

The following example creates three storage accounts:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "resources": [
        {
            "apiVersion": "2016-01-01",
            "type": "Microsoft.Storage/storageAccounts",
            "name": "[concat(copyIndex(),'storage', uniqueString(resourceGroup().id))]",
            "location": "[resourceGroup().location]",
            "sku": {
                "name": "Standard_LRS"
            },
            "kind": "Storage",
            "properties": {},
            "copy": {
                "name": "storagecopy",
                "count": 3
            }
        }
    ],
    "outputs": {}
}
```

You can also specify the number of instances dynamically from a parameter. For more information, see Deploy more than one instance of a resource or property in Azure Resource Manager Templates.

Conditionally deploy a resource

Sometimes you need to specify during deployment whether a resource in the template is deployed. For example, you may want the flexibility to either deploy a new resource or use an existing resource. The `condition` element gives you the ability to turn on or off deployment for a resource. When the expression in the condition element is true, the resource is deployed. When false, the resource is skipped during deployment.

The following example conditionally deploys a storage account:

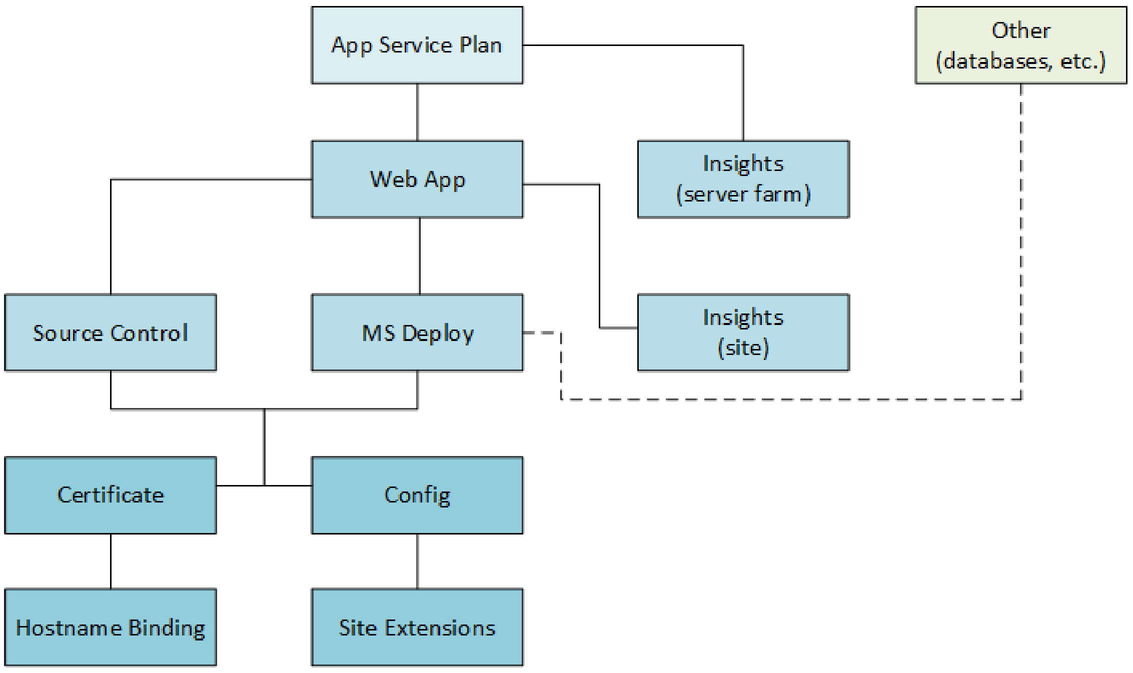
```
{
    "condition": "[equals(parameters('newOrExisting'),'new')]",
    "type": "Microsoft.Storage/storageAccounts",
    "name": "[variables('storageAccountName')]",
    "apiVersion": "2017-06-01",
    "location": "[resourceGroup().location]",
    "sku": {
        "name": "[variables('storageAccountType')]"
    },
    "kind": "Storage",
    "properties": {}
}
```

For more information, see the condition element.

Review dependencies

Some resources in your template need to be deployed before other resources. For example, the SQL server needs to exist before the SQL database is created. Resource Manager implicitly determines the deployment order for resources when the reference function is used. However, in some cases, you need to explicitly define the dependencies by using the `dependsOn` element. Review your template to see if any dependencies need to be added. Be careful to not add unnecessary dependencies as they can slow down deployment or create circular references.

The following image shows the dependency order for various App Service resources:



The following example shows part of a template that defines dependencies.

```
{
    "name": "[parameters('appName')]",
    "type": "Microsoft.Web/Sites",
    ...
    "resources": [
      {
          "name": "MSDeploy",
          "type": "Extensions",
          "dependsOn": [
            "[concat('Microsoft.Web/Sites/', parameters('appName'))]",
            "[concat('Microsoft.Sql/servers/', parameters('dbServerName'), '/databases/',
parameters('dbName'))]",
          ],
          ...
      },
      {
          "name": "connectionstrings",
          "type": "config",
          "dependsOn": [
            "[concat('Microsoft.Web/Sites/', parameters('appName'), '/Extensions/MSDeploy')]"
          ],
          ...
      }
    ]
}
```

For more information, see Define the order for deploying resources in Azure Resource Manager Templates.

Review recommended practices

Before deploying your template, review Azure Resource Manager template best practices to see if there are any recommended approaches you want to implement in your template.

If you need to use your template in different Azure cloud environments, see Develop Azure Resource Manager templates for cloud consistency.

Next steps

- To deploy a template, see Deploy with Azure CLI or Deploy with PowerShell.
- For a step-by-step Quickstart on creating a template, see Create Azure Resource Manager templates by using Visual Studio Code.
- For a list of the available functions in a template, see Template functions.

Create resource groups and resources at the subscription level

6/18/2019 • 8 minutes to read • Edit Online

Typically, you deploy Azure resources to a resource group in your Azure subscription. However, you can also create Azure resource groups, and create Azure resources at the subscription level. To deploy templates at the subscription level, you use Azure CLI and Azure PowerShell. The Azure portal doesn't support deployment in the subscription level.

To create a resource group in an Azure Resource Manager template, define a **Microsoft.Resources/resourceGroups** resource with a name and location for the resource group. You can create a resource group and deploy resources to that resource group in the same template. The resources that you can deploy at the subscription level include: Policies, and Role-based access control.

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

Deployment considerations

Subscription level deployment is different from resource group deployment in the following aspects:

Schema and commands

The schema and commands you use for subscription-level deployments are different than resource group deployments.

For the schema, use `https://schema.management.azure.com/schemas/2018-05-01/subscriptionDeploymentTemplate.json#`.

For the Azure CLI deployment command, use az deployment create. For example, the following CLI command deploys a template to create a resource group:

```
az deployment create \
  --name demoDeployment \
  --location centralus \
  --template-uri https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/emptyRG.json \
  --parameters rgName=demoResourceGroup rgLocation=centralus
```

For the PowerShell deployment command, use New-AzDeployment. For example, the following PowerShell command deploys a template to create a resource group:

```
New-AzDeployment `
  -Name demoDeployment `
  -Location centralus `
  -TemplateUri https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/emptyRG.json `
  -rgName demoResourceGroup `
  -rgLocation centralus
```

Deployment name and location

When deploying to your subscription, you must provide a location for the deployment. You can also provide a name for the deployment. If you don't specify a name for the deployment, the name of the template is used as the deployment name. For example, deploying a template named **azuredeploy.json** creates a default deployment name of **azuredeploy**.

The location of subscription level deployments is immutable. You can't create a deployment in one location when there's an existing deployment with the same name but different location. If you get the error code `InvalidDeploymentLocation`, either use a different name or the same location as the previous deployment for that name.

Use template functions

For subscription-level deployments, there are some important considerations when using template functions:

- The resourceGroup() function is **not** supported.
- The resourceId() function is supported. Use it to get the resource ID for resources that are used at subscription level deployments. For example, get the resource ID for a policy definition with `resourceId('Microsoft.Authorization/roleDefinitions/', parameters('roleDefinition'))`
- The reference() and list() functions are supported.

Create resource groups

The following template creates an empty resource group.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2018-05-01/subscriptionDeploymentTemplate.json#",
    "contentVersion": "1.0.0.1",
    "parameters": {
        "rgName": {
            "type": "string"
        },
        "rgLocation": {
            "type": "string"
        }
    },
    "variables": {},
    "resources": [
        {
            "type": "Microsoft.Resources/resourceGroups",
            "apiVersion": "2018-05-01",
            "location": "[parameters('rgLocation')]",
            "name": "[parameters('rgName')]",
            "properties": {}
        }
    ],
    "outputs": {}
}
```

The template schema can be found at here. Similar templates can be found at GitHub.

Create multiple resource groups

Use the copy element with resource groups to create more than one resource group.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2018-05-01/subscriptionDeploymentTemplate.json#",
    "contentVersion": "1.0.0.1",
    "parameters": {
        "rgNamePrefix": {
            "type": "string"
        },
        "rgLocation": {
            "type": "string"
        },
        "instanceCount": {
            "type": "int"
        }
    },
    "variables": {},
    "resources": [
        {
            "type": "Microsoft.Resources/resourceGroups",
            "apiVersion": "2018-05-01",
            "location": "[parameters('rgLocation')]",
            "name": "[concat(parameters('rgNamePrefix'), copyIndex())]",
            "copy": {
                "name": "rgCopy",
                "count": "[parameters('instanceCount')]"
            },
            "properties": {}
        }
    ],
    "outputs": {}
}
```

For information about resource iteration, see Deploy more than one instance of a resource or property in Azure Resource Manager Templates, and Tutorial: Create multiple resource instances with Resource Manager templates.

Create resource group and deploy resources

To create the resource group and deploy resources to it, use a nested template. The nested template defines the resources to deploy to the resource group. Set the nested template as dependent on the resource group to make sure the resource group exists before deploying the resources.

The following example creates a resource group, and deploys a storage account to the resource group.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2018-05-01/subscriptionDeploymentTemplate.json#",
    "contentVersion": "1.0.0.1",
    "parameters": {
        "rgName": {
            "type": "string"
        },
        "rgLocation": {
            "type": "string"
        },
        "storagePrefix": {
            "type": "string",
            "maxLength": 11
        }
    },
    "variables": {
        "storageName": "[concat(parameters('storagePrefix'), uniqueString(subscription().id,
parameters('rgName')))]"
    },
    "resources": [
        {
            "type": "Microsoft.Resources/resourceGroups",
            "apiVersion": "2018-05-01",
            "location": "[parameters('rgLocation')]",
            "name": "[parameters('rgName')]",
            "properties": {}
        },
        {
            "type": "Microsoft.Resources/deployments",
            "apiVersion": "2018-05-01",
            "name": "storageDeployment",
            "resourceGroup": "[parameters('rgName')]",
            "dependsOn": [
                "[resourceId('Microsoft.Resources/resourceGroups/', parameters('rgName'))]"
            ],
            "properties": {
                "mode": "Incremental",
                "template": {
                    "$schema": "https://schema.management.azure.com/schemas/2015-01-
01/deploymentTemplate.json#",
                    "contentVersion": "1.0.0.0",
                    "parameters": {},
                    "variables": {},
                    "resources": [
                        {
                            "type": "Microsoft.Storage/storageAccounts",
                            "apiVersion": "2017-10-01",
                            "name": "[variables('storageName')]",
                            "location": "[parameters('rgLocation')]",
                            "kind": "StorageV2",
                            "sku": {
                                "name": "Standard_LRS"
                            }
                        }
                    ],
                    "outputs": {}
                }
            }
        }
    ],
    "outputs": {}
}
```

Create policies

Assign policy

The following example assigns an existing policy definition to the subscription. If the policy takes parameters, provide them as an object. If the policy doesn't take parameters, use the default empty object.

```json
{
    "$schema": "https://schema.management.azure.com/schemas/2018-05-01/subscriptionDeploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "policyDefinitionID": {
            "type": "string"
        },
        "policyName": {
            "type": "string"
        },
        "policyParameters": {
            "type": "object",
            "defaultValue": {}
        }
    },
    "variables": {},
    "resources": [
        {
            "type": "Microsoft.Authorization/policyAssignments",
            "name": "[parameters('policyName')]",
            "apiVersion": "2018-03-01",
            "properties": {
                "scope": "[subscription().id]",
                "policyDefinitionId": "[parameters('policyDefinitionID')]",
                "parameters": "[parameters('policyParameters')]"
            }
        }
    ]
}
```

To apply a built-in policy to your Azure subscription, use the following Azure CLI commands:

```
# Built-in policy that does not accept parameters
definition=$(az policy definition list --query "[?displayName=='Audit resource location matches resource
group location'].id" --output tsv)

az deployment create \
  --name demoDeployment \
  --location centralus \
  --template-uri https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/policyassign.json \
  --parameters policyDefinitionID=$definition policyName=auditRGLocation
```

To deploy this template with PowerShell, use:

```
$definition = Get-AzPolicyDefinition | Where-Object { $_.Properties.DisplayName -eq 'Audit resource location
matches resource group location' }

New-AzDeployment `
  -Name policyassign `
  -Location centralus `
  -TemplateUri https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/policyassign.json `
  -policyDefinitionID $definition.PolicyDefinitionId `
  -policyName auditRGLocation
```

To apply a built-in policy to your Azure subscription, use the following Azure CLI commands:

```
# Built-in policy that accepts parameters
definition=$(az policy definition list --query "[?displayName=='Allowed locations'].id" --output tsv)

az deployment create \
  --name demoDeployment \
  --location centralus \
  --template-uri https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-manager/policyassign.json \
  --parameters policyDefinitionID=$definition policyName=setLocation policyParameters="{'listOfAllowedLocations': {'value': ['westus']} }"
```

To deploy this template with PowerShell, use:

```
$definition = Get-AzPolicyDefinition | Where-Object { $_.Properties.DisplayName -eq 'Allowed locations' }

$locations = @("westus", "westus2")
$policyParams =@{listOfAllowedLocations = @{ value = $locations}}

New-AzDeployment `
  -Name policyassign `
  -Location centralus `
  -TemplateUri https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-manager/policyassign.json `
  -policyDefinitionID $definition.PolicyDefinitionId `
  -policyName setLocation `
  -policyParameters $policyParams
```

Define and assign policy

You can define and assign a policy in the same template.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2018-05-01/subscriptionDeploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {},
    "variables": {},
    "resources": [
        {
            "type": "Microsoft.Authorization/policyDefinitions",
            "name": "locationpolicy",
            "apiVersion": "2018-05-01",
            "properties": {
                "policyType": "Custom",
                "parameters": {},
                "policyRule": {
                    "if": {
                        "field": "location",
                        "equals": "northeurope"
                    },
                    "then": {
                        "effect": "deny"
                    }
                }
            }
        },
        {
            "type": "Microsoft.Authorization/policyAssignments",
            "name": "location-lock",
            "apiVersion": "2018-05-01",
            "dependsOn": [
                "locationpolicy"
            ],
            "properties": {
                "scope": "[subscription().id]",
                "policyDefinitionId": "[resourceId('Microsoft.Authorization/policyDefinitions',
'locationpolicy')]"
            }
        }
    ]
}
```

To create the policy definition in your subscription, and apply it to the subscription, use the following CLI command:

```
az deployment create \
  --name demoDeployment \
  --location centralus \
  --template-uri https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/policydefineandassign.json
```

To deploy this template with PowerShell, use:

```
New-AzDeployment `
  -Name definePolicy `
  -Location centralus `
  -TemplateUri https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/policydefineandassign.json
```

Create roles

Assign role at subscription

The following example assigns a role to a user or group for the subscription. In this example, you don't specify a scope for the assignment because the scope is automatically set to the subscription.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2018-05-01/subscriptionDeploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "principalId": {
            "type": "string"
        },
        "roleDefinitionId": {
            "type": "string"
        }
    },
    "variables": {},
    "resources": [
        {
            "type": "Microsoft.Authorization/roleAssignments",
            "name": "[guid(parameters('principalId'), deployment().name)]",
            "apiVersion": "2017-09-01",
            "properties": {
                "roleDefinitionId": "[resourceId('Microsoft.Authorization/roleDefinitions', 
parameters('roleDefinitionId'))]",
                "principalId": "[parameters('principalId')]"
            }
        }
    ]
}
```

To assign an Active Directory group to a role for your subscription, use the following Azure CLI commands:

```
# Get ID of the role you want to assign
role=$(az role definition list --name Contributor --query [].name --output tsv)

# Get ID of the AD group to assign the role to
principalid=$(az ad group show --group demogroup --query objectId --output tsv)

az deployment create \
  --name demoDeployment \
  --location centralus \
  --template-uri https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/roleassign.json \
  --parameters principalId=$principalid roleDefinitionId=$role
```

To deploy this template with PowerShell, use:

```
$role = Get-AzRoleDefinition -Name Contributor

$adgroup = Get-AzADGroup -DisplayName demogroup

New-AzDeployment `
  -Name demoRole `
  -Location centralus `
  -TemplateUri https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/roleassign.json `
  -roleDefinitionId $role.Id `
  -principalId $adgroup.Id
```

Assign role at scope

The following subscription-level template assigns a role to a user or group that is scoped to a resource group within the subscription. The scope must be at or below the level of deployment. You can deploy to a subscription and specify a role assignment scoped to a resource group within that subscription. However, you can't deploy to a resource group and specify a role assignment scope to the subscription.

To assign the role at a scope, use a nested deployment. Notice that the resource group name is specified both in the properties for the deployment resource and in the scope property of the role assignment.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2018-05-01/subscriptionDeploymentTemplate.json#",
    "contentVersion": "1.0.0.1",
    "parameters": {
        "principalId": {
            "type": "string"
        },
        "roleDefinitionId": {
            "type": "string"
        },
        "rgName": {
            "type": "string"
        }
    },
    "variables": {},
    "resources": [
        {
            "type": "Microsoft.Resources/deployments",
            "apiVersion": "2018-05-01",
            "name": "assignRole",
            "resourceGroup": "[parameters('rgName')]",
            "properties": {
                "mode": "Incremental",
                "template": {
                    "$schema": "https://schema.management.azure.com/schemas/2015-01-
01/deploymentTemplate.json#",
                    "contentVersion": "1.0.0.0",
                    "parameters": {},
                    "variables": {},
                    "resources": [
                        {
                            "type": "Microsoft.Authorization/roleAssignments",
                            "name": "[guid(parameters('principalId'), deployment().name)]",
                            "apiVersion": "2017-09-01",
                            "properties": {
                                "roleDefinitionId": "[resourceId('Microsoft.Authorization/roleDefinitions',
parameters('roleDefinitionId'))]",
                                "principalId": "[parameters('principalId')]",
                                "scope": "[concat(subscription().id, '/resourceGroups/',
parameters('rgName'))]"
                            }
                        }
                    ],
                    "outputs": {}
                }
            }
        }
    ],
    "outputs": {}
}
```

To assign an Active Directory group to a role for your subscription, use the following Azure CLI commands:

```
# Get ID of the role you want to assign
role=$(az role definition list --name Contributor --query [].name --output tsv)

# Get ID of the AD group to assign the role to
principalid=$(az ad group show --group demogroup --query objectId --output tsv)

az deployment create \
  --name demoDeployment \
  --location centralus \
  --template-uri https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/scopedRoleAssign.json \
  --parameters principalId=$principalid roleDefinitionId=$role rgName demoRg
```

To deploy this template with PowerShell, use:

```
$role = Get-AzRoleDefinition -Name Contributor

$adgroup = Get-AzADGroup -DisplayName demogroup

New-AzDeployment `
  -Name demoRole `
  -Location centralus `
  -TemplateUri https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/scopedRoleAssign.json `
  -roleDefinitionId $role.Id `
  -principalId $adgroup.Id `
  -rgName demoRg
```

Next steps

- For an example of deploying workspace settings for Azure Security Center, see deployASCwithWorkspaceSettings.json.
- To learn about creating Azure Resource Manager templates, see Authoring templates.
- For a list of the available functions in a template, see Template functions.

Define the order for deploying resources in Azure Resource Manager Templates

6/18/2019 • 4 minutes to read • Edit Online

For a given resource, there can be other resources that must exist before the resource is deployed. For example, a SQL server must exist before attempting to deploy a SQL database. You define this relationship by marking one resource as dependent on the other resource. You define a dependency with the **dependsOn** element, or by using the **reference** function.

Resource Manager evaluates the dependencies between resources, and deploys them in their dependent order. When resources aren't dependent on each other, Resource Manager deploys them in parallel. You only need to define dependencies for resources that are deployed in the same template.

For a tutorial, see Tutorial: create Azure Resource Manager templates with dependent resources.

dependsOn

Within your template, the dependsOn element enables you to define one resource as a dependent on one or more resources. Its value can be a comma-separated list of resource names.

The following example shows a virtual machine scale set that depends on a load balancer, virtual network, and a loop that creates multiple storage accounts. These other resources aren't shown in the following example, but they would need to exist elsewhere in the template.

```
{
  "type": "Microsoft.Compute/virtualMachineScaleSets",
  "name": "[variables('namingInfix')]",
  "location": "[variables('location')]",
  "apiVersion": "2016-03-30",
  "tags": {
    "displayName": "VMScaleSet"
  },
  "dependsOn": [
    "[variables('loadBalancerName')]",
    "[variables('virtualNetworkName')]",
    "storageLoop",
  ],
  ...
}
```

In the preceding example, a dependency is included on the resources that are created through a copy loop named **storageLoop**. For an example, see Create multiple instances of resources in Azure Resource Manager.

When defining dependencies, you can include the resource provider namespace and resource type to avoid ambiguity. For example, to clarify a load balancer and virtual network that may have the same names as other resources, use the following format:

```
"dependsOn": [
  "[resourceId('Microsoft.Network/loadBalancers', variables('loadBalancerName'))]",
  "[resourceId('Microsoft.Network/virtualNetworks', variables('virtualNetworkName'))]"
]
```

While you may be inclined to use dependsOn to map relationships between your resources, it's important to understand why you're doing it. For example, to document how resources are interconnected, dependsOn isn't the right approach. You can't query which resources were defined in the dependsOn element after deployment. By using dependsOn, you potentially impact deployment time because Resource Manager doesn't deploy in parallel two resources that have a dependency.

Child resources

The resources property allows you to specify child resources that are related to the resource being defined. Child resources can only be defined five levels deep. It's important to note that an implicit deployment dependency isn't created between a child resource and the parent resource. If you need the child resource to be deployed after the parent resource, you must explicitly state that dependency with the dependsOn property.

Each parent resource accepts only certain resource types as child resources. The accepted resource types are specified in the template schema of the parent resource. The name of child resource type includes the name of the parent resource type, such as **Microsoft.Web/sites/config** and **Microsoft.Web/sites/extensions** are both child resources of the **Microsoft.Web/sites**.

The following example shows a SQL server and SQL database. Notice that an explicit dependency is defined between the SQL database and SQL server, even though the database is a child of the server.

```
"resources": [
  {
    "name": "[variables('sqlserverName')]",
    "type": "Microsoft.Sql/servers",
    "location": "[resourceGroup().location]",
    "tags": {
      "displayName": "SqlServer"
    },
    "apiVersion": "2014-04-01-preview",
    "properties": {
      "administratorLogin": "[parameters('administratorLogin')]",
      "administratorLoginPassword": "[parameters('administratorLoginPassword')]"
    },
    "resources": [
      {
        "name": "[parameters('databaseName')]",
        "type": "databases",
        "location": "[resourceGroup().location]",
        "tags": {
          "displayName": "Database"
        },
        "apiVersion": "2014-04-01-preview",
        "dependsOn": [
          "[variables('sqlserverName')]"
        ],
        "properties": {
          "edition": "[parameters('edition')]",
          "collation": "[parameters('collation')]",
          "maxSizeBytes": "[parameters('maxSizeBytes')]",
          "requestedServiceObjectiveName": "[parameters('requestedServiceObjectiveName')]"
        }
      }
    ]
  }
]
```

reference and list functions

The reference function enables an expression to derive its value from other JSON name and value pairs or runtime resources. The list* functions return values for a resource from a list operation. Reference and list expressions implicitly declare that one resource depends on another, when the referenced resource is deployed in the same template and referred to by its name (not resource ID). If you pass the resource ID into the reference or list functions, an implicit reference isn't created.

The general format of the reference function is:

```
reference('resourceName').propertyPath
```

The general format of the listKeys function is:

```
listKeys('resourceName', 'yyyy-mm-dd')
```

In the following example, a CDN endpoint explicitly depends on the CDN profile, and implicitly depends on a web app.

```
{
    "name": "[variables('endpointName')]",
    "type": "endpoints",
    "location": "[resourceGroup().location]",
    "apiVersion": "2016-04-02",
    "dependsOn": [
            "[variables('profileName')]"
    ],
    "properties": {
        "originHostHeader": "[reference(variables('webAppName')).hostNames[0]]",
        ...
    }
```

You can use either this element or the dependsOn element to specify dependencies, but you don't need to use both for the same dependent resource. Whenever possible, use an implicit reference to avoid adding an unnecessary dependency.

To learn more, see reference function.

Circular dependencies

Resource Manager identifies circular dependencies during template validation. If you receive an error stating that a circular dependency exists, evaluate your template to see if any dependencies aren't needed and can be removed. If removing dependencies doesn't work, you can avoid circular dependencies by moving some deployment operations into child resources that are deployed after the resources that have the circular dependency. For example, suppose you're deploying two virtual machines but you must set properties on each one that refer to the other. You can deploy them in the following order:

1. vm1
2. vm2
3. Extension on vm1 depends on vm1 and vm2. The extension sets values on vm1 that it gets from vm2.
4. Extension on vm2 depends on vm1 and vm2. The extension sets values on vm2 that it gets from vm1.

For information about assessing the deployment order and resolving dependency errors, see Troubleshoot common Azure deployment errors with Azure Resource Manager.

Next steps

- To go through a tutorial, see Tutorial: create Azure Resource Manager templates with dependent resources.
- For recommendations when setting dependencies, see Azure Resource Manager template best practices.
- To learn about troubleshooting dependencies during deployment, see Troubleshoot common Azure

deployment errors with Azure Resource Manager.

- To learn about creating Azure Resource Manager templates, see Authoring templates.
- For a list of the available functions in a template, see Template functions.

Using linked and nested templates when deploying Azure resources

7/18/2019 • 10 minutes to read • Edit Online

To deploy your solution, you can use either a single template or a main template with many related templates. The related template can be either a separate file that is linked to from the main template, or a template that is nested within the main template.

For small to medium solutions, a single template is easier to understand and maintain. You can see all the resources and values in a single file. For advanced scenarios, linked templates enable you to break down the solution into targeted components, and reuse templates.

When using linked templates, you create a main template that receives the parameter values during deployment. The main template contains all the linked templates and passes values to those templates as needed.

For a tutorial, see Tutorial: create linked Azure Resource Manager templates.

NOTE

For linked or nested templates, you can only use Incremental deployment mode.

Link or nest a template

To link to another template, add a **deployments** resource to your main template.

```
"resources": [
  {
    "type": "Microsoft.Resources/deployments",
    "apiVersion": "2018-05-01",
    "name": "linkedTemplate",
    "properties": {
        "mode": "Incremental",
        <nested-template-or-external-template>
    }
  }
]
```

The properties you provide for the deployment resource vary based on whether you're linking to an external template or nesting an inline template in the main template.

Nested template

To nest the template within the main template, use the **template** property and specify the template syntax.

```
"resources": [
  {
    "type": "Microsoft.Resources/deployments",
    "apiVersion": "2018-05-01",
    "name": "nestedTemplate",
    "properties": {
      "mode": "Incremental",
      "template": {
        "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
        "contentVersion": "1.0.0.0",
        "resources": [
          {
            "type": "Microsoft.Storage/storageAccounts",
            "apiVersion": "2019-04-01",
            "name": "[variables('storageName')]",
            "location": "West US",
            "kind": "StorageV2",
            "sku": {
                "name": "Standard_LRS"
            }
          }
        ]
      }
    }
  }
]
```

NOTE

For nested templates, you cannot use parameters or variables that are defined within the nested template. You can use parameters and variables from the main template. In the preceding example, `[variables('storageName')]` retrieves a value from the main template, not the nested template. This restriction does not apply to external templates.

For two resources defined inside a nested template and one resource depends on the other, the value of the dependency is simply the name of the dependent resource:

```
"dependsOn": [
  "[variables('storageAccountName')]"
],
```

You can't use the `reference` function in the outputs section of a nested template for a resource you have deployed in the nested template. To return the values for a deployed resource in a nested template, convert your nested template to a linked template.

The nested template requires the same properties as a standard template.

External template and external parameters

To link to an external template and parameter file, use **templateLink** and **parametersLink**. When linking to a template, the Resource Manager service must be able to access it. You can't specify a local file or a file that is only available on your local network. You can only provide a URI value that includes either **http** or **https**. One option is to place your linked template in a storage account, and use the URI for that item.

```
"resources": [
  {
    "type": "Microsoft.Resources/deployments",
    "apiVersion": "2018-05-01",
    "name": "linkedTemplate",
    "properties": {
    "mode": "Incremental",
    "templateLink": {
        "uri":"https://mystorageaccount.blob.core.windows.net/AzureTemplates/newStorageAccount.json",
        "contentVersion":"1.0.0.0"
    },
    "parametersLink": {

"uri":"https://mystorageaccount.blob.core.windows.net/AzureTemplates/newStorageAccount.parameters.json",
        "contentVersion":"1.0.0.0"
    }
    }
  }
]
```

You don't have to provide the `contentVersion` property for the template or parameters. If you don't provide a content version value, the current version of the template is deployed. If you provide a value for content version, it must match the version in the linked template; otherwise, the deployment fails with an error.

External template and inline parameters

Or, you can provide the parameter inline. You can't use both inline parameters and a link to a parameter file. The deployment fails with an error when both `parametersLink` and `parameters` are specified.

To pass a value from the main template to the linked template, use **parameters**.

```
"resources": [
  {
     "type": "Microsoft.Resources/deployments",
     "apiVersion": "2018-05-01",
     "name": "linkedTemplate",
     "properties": {
       "mode": "Incremental",
       "templateLink": {
          "uri":"https://mystorageaccount.blob.core.windows.net/AzureTemplates/newStorageAccount.json",
          "contentVersion":"1.0.0.0"
       },
       "parameters": {
          "StorageAccountName":{"value": "[parameters('StorageAccountName')]"}
        }
     }
  }
]
```

Using copy

To create multiple instances of a resource with a nested template, add the copy element at the level of the **Microsoft.Resources/deployments** resource.

The following example template shows how to use copy with a nested template.

```
"resources": [
  {
    "type": "Microsoft.Resources/deployments",
    "apiVersion": "2018-05-01",
    "name": "[concat('nestedTemplate', copyIndex())]",
    // yes, copy works here
    "copy":{
      "name": "storagecopy",
      "count": 2
    },
    "properties": {
      "mode": "Incremental",
      "template": {
        "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
        "contentVersion": "1.0.0.0",
        "resources": [
          {
            "type": "Microsoft.Storage/storageAccounts",
            "apiVersion": "2019-04-01",
            "name": "[concat(variables('storageName'), copyIndex())]",
            "location": "West US",
            "kind": "StorageV2",
            "sku": {
              "name": "Standard_LRS"
            }
            // no, copy doesn't work here
            //"copy":{
            //  "name": "storagecopy",
            //  "count": 2
            //}
          }
        ]
      }
    }
  }
]
```

Using variables to link templates

The previous examples showed hard-coded URL values for the template links. This approach might work for a simple template but it doesn't work well when working with a large set of modular templates. Instead, you can create a static variable that stores a base URL for the main template and then dynamically create URLs for the linked templates from that base URL. The benefit of this approach is you can easily move or fork the template because you only need to change the static variable in the main template. The main template passes the correct URIs throughout the decomposed template.

The following example shows how to use a base URL to create two URLs for linked templates (**sharedTemplateUrl** and **vmTemplate**).

```
"variables": {
    "templateBaseUrl": "https://raw.githubusercontent.com/Azure/azure-quickstart-
templates/master/postgresql-on-ubuntu/",
    "sharedTemplateUrl": "[concat(variables('templateBaseUrl'), 'shared-resources.json')]",
    "vmTemplateUrl": "[concat(variables('templateBaseUrl'), 'database-2disk-resources.json')]"
}
```

You can also use deployment() to get the base URL for the current template, and use that to get the URL for other templates in the same location. This approach is useful if your template location changes or you want to avoid hard coding URLs in the template file. The templateLink property is only returned when linking to a remote template with a URL. If you're using a local template, that property isn't available.

```
"variables": {
    "sharedTemplateUrl": "[uri(deployment().properties.templateLink.uri, 'shared-resources.json')]"
}
```

Get values from linked template

To get an output value from a linked template, retrieve the property value with syntax like: `"[reference('deploymentName').outputs.propertyName.value]"`.

When getting an output property from a linked template, the property name can't include a dash.

The following examples demonstrate how to reference a linked template and retrieve an output value. The linked template returns a simple message.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {},
    "variables": {},
    "resources": [],
    "outputs": {
        "greetingMessage": {
            "value": "Hello World",
            "type" : "string"
        }
    }
}
```

The main template deploys the linked template and gets the returned value. Notice that it references the deployment resource by name, and it uses the name of the property returned by the linked template.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {},
    "variables": {},
    "resources": [
        {
            "type": "Microsoft.Resources/deployments",
            "apiVersion": "2018-05-01",
            "name": "linkedTemplate",
            "properties": {
                "mode": "Incremental",
                "templateLink": {
                    "uri": "[uri(deployment().properties.templateLink.uri, 'helloworld.json')]",
                    "contentVersion": "1.0.0.0"
                }
            }
        }
    ],
    "outputs": {
        "messageFromLinkedTemplate": {
            "type": "string",
            "value": "[reference('linkedTemplate').outputs.greetingMessage.value]"
        }
    }
}
```

Like other resource types, you can set dependencies between the linked template and other resources. Therefore, when other resources require an output value from the linked template, make sure the linked template is deployed before them. Or, when the linked template relies on other resources, make sure other resources are deployed before the linked template.

The following example shows a template that deploys a public IP address and returns the resource ID:

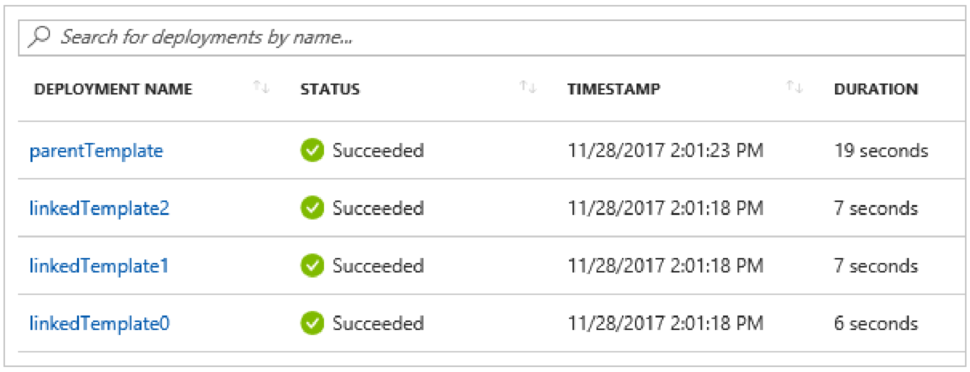
```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "publicIPAddresses_name": {
            "type": "string"
        }
    },
    "variables": {},
    "resources": [
        {
            "type": "Microsoft.Network/publicIPAddresses",
            "apiVersion": "2018-11-01",
            "name": "[parameters('publicIPAddresses_name')]",
            "location": "eastus",
            "properties": {
                "publicIPAddressVersion": "IPv4",
                "publicIPAllocationMethod": "Dynamic",
                "idleTimeoutInMinutes": 4
            },
            "dependsOn": []
        }
    ],
    "outputs": {
        "resourceID": {
            "type": "string",
            "value": "[resourceId('Microsoft.Network/publicIPAddresses',
parameters('publicIPAddresses_name'))]"
        }
    }
}
```

To use the public IP address from the preceding template when deploying a load balancer, link to the template and add a dependency on the deployment resource. The public IP address on the load balancer is set to the output value from the linked template.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "loadBalancers_name": {
            "defaultValue": "mylb",
            "type": "string"
        },
        "publicIPAddresses_name": {
            "defaultValue": "myip",
            "type": "string"
        }
    },
    "variables": {},
    "resources": [
        {
            "type": "Microsoft.Network/loadBalancers",
            "apiVersion": "2018-11-01",
            "name": "[parameters('loadBalancers_name')]",
            "location": "eastus",
            "properties": {
                "frontendIPConfigurations": [
                    {
                        "name": "LoadBalancerFrontEnd",
                        "properties": {
                            "privateIPAllocationMethod": "Dynamic",
                            "publicIPAddress": {
                                "id": "[reference('linkedTemplate').outputs.resourceID.value]"
                            }
                        }
                    }
                ],
                "backendAddressPools": [],
                "loadBalancingRules": [],
                "probes": [],
                "inboundNatRules": [],
                "outboundNatRules": [],
                "inboundNatPools": []
            },
            "dependsOn": [
                "linkedTemplate"
            ]
        },
        {
            "type": "Microsoft.Resources/deployments",
            "apiVersion": "2018-05-01",
            "name": "linkedTemplate",
            "properties": {
                "mode": "Incremental",
                "templateLink": {
                    "uri": "[uri(deployment().properties.templateLink.uri, 'publicip.json')]",
                    "contentVersion": "1.0.0.0"
                },
                "parameters":{
                    "publicIPAddresses_name":{"value": "[parameters('publicIPAddresses_name')]"}
                }
            }
        }
    ]
}
```

Linked and nested templates in deployment history

Resource Manager processes each template as a separate deployment in the deployment history. Therefore, a main template with three linked or nested templates appears in the deployment history as:



Search for deployments by name...

DEPLOYMENT NAME	STATUS	TIMESTAMP	DURATION
parentTemplate	✓ Succeeded	11/28/2017 2:01:23 PM	19 seconds
linkedTemplate2	✓ Succeeded	11/28/2017 2:01:18 PM	7 seconds
linkedTemplate1	✓ Succeeded	11/28/2017 2:01:18 PM	7 seconds
linkedTemplate0	✓ Succeeded	11/28/2017 2:01:18 PM	6 seconds

You can use these separate entries in the history to retrieve output values after the deployment. The following template creates a public IP address and outputs the IP address:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "publicIPAddresses_name": {
            "type": "string"
        }
    },
    "variables": {},
    "resources": [
        {
            "type": "Microsoft.Network/publicIPAddresses",
            "apiVersion": "2018-11-01",
            "name": "[parameters('publicIPAddresses_name')]",
            "location": "southcentralus",
            "properties": {
                "publicIPAddressVersion": "IPv4",
                "publicIPAllocationMethod": "Static",
                "idleTimeoutInMinutes": 4,
                "dnsSettings": {
                    "domainNameLabel": "[concat(parameters('publicIPAddresses_name'),
uniqueString(resourceGroup().id))]"
                }
            },
            "dependsOn": []
        }
    ],
    "outputs": {
        "returnedIPAddress": {
            "type": "string",
            "value": "[reference(parameters('publicIPAddresses_name')).ipAddress]"
        }
    }
}
```

The following template links to the preceding template. It creates three public IP addresses.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
    },
    "variables": {},
    "resources": [
        {
            "type": "Microsoft.Resources/deployments",
            "apiVersion": "2018-05-01",
            "name": "[concat('linkedTemplate', copyIndex())]",
            "copy": {
                "count": 3,
                "name": "ip-loop"
            },
            "properties": {
              "mode": "Incremental",
              "templateLink": {
                "uri": "[uri(deployment().properties.templateLink.uri, 'static-public-ip.json')]",
                "contentVersion": "1.0.0.0"
              },
              "parameters":{
                  "publicIPAddresses_name":{"value": "[concat('myip-', copyIndex())]"}
              }
            }
        }
    ]
}
```

After the deployment, you can retrieve the output values with the following PowerShell script:

```
$loopCount = 3
for ($i = 0; $i -lt $loopCount; $i++)
{
    $name = 'linkedTemplate' + $i;
    $deployment = Get-AzResourceGroupDeployment -ResourceGroupName examplegroup -Name $name
    Write-Output "deployment $($deployment.DeploymentName) returned
$($deployment.Outputs.returnedIPAddress.value)"
}
```

Or, Azure CLI script in a Bash shell:

```
#!/bin/bash

for i in 0 1 2;
do
    name="linkedTemplate$i";
    deployment=$(az group deployment show -g examplegroup -n $name);
    ip=$(echo $deployment | jq .properties.outputs.returnedIPAddress.value);
    echo "deployment $name returned $ip";
done
```

Securing an external template

Although the linked template must be externally available, it doesn't need to be generally available to the public. You can add your template to a private storage account that is accessible to only the storage account owner. Then, you create a shared access signature (SAS) token to enable access during deployment. You add that SAS token to the URI for the linked template. Even though the token is passed in as a secure string, the URI of the linked template, including the SAS token, is logged in the deployment operations. To limit exposure, set an expiration for the token.

The parameter file can also be limited to access through a SAS token.

The following example shows how to pass a SAS token when linking to a template:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "containerSasToken": { "type": "string" }
  },
  "resources": [
    {
      "type": "Microsoft.Resources/deployments",
      "apiVersion": "2018-05-01",
      "name": "linkedTemplate",
      "properties": {
        "mode": "Incremental",
        "templateLink": {
          "uri": "[concat(uri(deployment().properties.templateLink.uri, 'helloworld.json'),
parameters('containerSasToken'))]",
          "contentVersion": "1.0.0.0"
        }
      }
    }
  ],
  "outputs": {
  }
}
```

In PowerShell, you get a token for the container and deploy the templates with the following commands. Notice that the **containerSasToken** parameter is defined in the template. It isn't a parameter in the **New-AzResourceGroupDeployment** command.

```
Set-AzCurrentStorageAccount -ResourceGroupName ManageGroup -Name storagecontosotemplates
$token = New-AzStorageContainerSASToken -Name templates -Permission r -ExpiryTime (Get-
Date).AddMinutes(30.0)
$url = (Get-AzStorageBlob -Container templates -Blob parent.json).ICloudBlob.uri.AbsoluteUri
New-AzResourceGroupDeployment -ResourceGroupName ExampleGroup -TemplateUri ($url + $token) -
containerSasToken $token
```

For Azure CLI in a Bash shell, you get a token for the container and deploy the templates with the following code:

```
#!/bin/bash

expiretime=$(date -u -d '30 minutes' +%Y-%m-%dT%H:%MZ)
connection=$(az storage account show-connection-string \
    --resource-group ManageGroup \
    --name storagecontosotemplates \
    --query connectionString)
token=$(az storage container generate-sas \
    --name templates \
    --expiry $expiretime \
    --permissions r \
    --output tsv \
    --connection-string $connection)
url=$(az storage blob url \
    --container-name templates \
    --name parent.json \
    --output tsv \
    --connection-string $connection)
parameter='{"containerSasToken":{"value":"?'$token'"}}'
az group deployment create --resource-group ExampleGroup --template-uri $url?$token --parameters $parameter
```

Example templates

The following examples show common uses of linked templates.

MAIN TEMPLATE	LINKED TEMPLATE	DESCRIPTION
Hello World	linked template	Returns string from linked template.
Load Balancer with public IP address	linked template	Returns public IP address from linked template and sets that value in load balancer.
Multiple IP addresses	linked template	Creates several public IP addresses in linked template.

Next steps

- To go through a tutorial, see Tutorial: create linked Azure Resource Manager templates.
- To learn about the defining the deployment order for your resources, see Defining dependencies in Azure Resource Manager templates.
- To learn how to define one resource but create many instances of it, see Create multiple instances of resources in Azure Resource Manager.
- For steps on setting up a template in a storage account and generating a SAS token, see Deploy resources with Resource Manager templates and Azure PowerShell or Deploy resources with Resource Manager templates and Azure CLI.

Use Azure Key Vault to pass secure parameter value during deployment

7/9/2019 • 6 minutes to read • Edit Online

Instead of putting a secure value (like a password) directly in your template or parameter file, you can retrieve the value from an Azure Key Vault during a deployment. You retrieve the value by referencing the key vault and secret in your parameter file. The value is never exposed because you only reference its key vault ID. The key vault can exist in a different subscription than the resource group you're deploying to.

Deploy key vaults and secrets

To access a key vault during template deployment, set `enabledForTemplateDeployment` on the key vault to `true`.

The following Azure CLI and Azure PowerShell samples show how to create the key vault, and add a secret.

```
az group create --name $resourceGroupName --location $location
az keyvault create \
  --name $keyVaultName \
  --resource-group $resourceGroupName \
  --location $location \
  --enabled-for-template-deployment true
az keyvault secret set --vault-name $keyVaultName --name "ExamplePassword" --value "hVFkk965BuUv"
```

```
New-AzResourceGroup -Name $resourceGroupName -Location $location
New-AzKeyVault `
  -VaultName $keyVaultName `
  -resourceGroupName $resourceGroupName `
  -Location $location `
  -EnabledForTemplateDeployment
$secretvalue = ConvertTo-SecureString 'hVFkk965BuUv' -AsPlainText -Force
$secret = Set-AzKeyVaultSecret -VaultName $keyVaultName -Name 'ExamplePassword' -SecretValue $secretvalue
```

As the owner of the key vault, you automatically have access to creating secrets. If the user working with secrets isn't the owner of the key vault, grant access with:

```
az keyvault set-policy \
  --upn $userPrincipalName \
  --name $keyVaultName \
  --secret-permissions set delete get list
```

```
$userPrincipalName = "<Email Address of the deployment operator>"

Set-AzKeyVaultAccessPolicy `
  -VaultName $keyVaultName `
  -UserPrincipalName $userPrincipalName `
  -PermissionsToSecrets set,delete,get,list
```

For more information about creating key vaults and adding secrets, see:

- Set and retrieve a secret by using CLI
- Set and retrieve a secret by using Powershell

- Set and retrieve a secret by using the portal
- Set and retrieve a secret by using .NET
- Set and retrieve a secret by using Node.js

Grant access to the secrets

The user who deploys the template must have the `Microsoft.KeyVault/vaults/deploy/action` permission for the scope of the resource group and key vault. The Owner and Contributor roles both grant this access. If you created the key vault, you're the owner so you have the permission.

The following procedure shows how to create a role with the minimum permission, and how to assign the user

1. Create a custom role definition JSON file:

```
{
  "Name": "Key Vault resource manager template deployment operator",
  "IsCustom": true,
  "Description": "Lets you deploy a resource manager template with the access to the secrets in the Key
Vault.",
  "Actions": [
    "Microsoft.KeyVault/vaults/deploy/action"
  ],
  "NotActions": [],
  "DataActions": [],
  "NotDataActions": [],
  "AssignableScopes": [
    "/subscriptions/00000000-0000-0000-0000-000000000000"
  ]
}
```

   Replace "00000000-0000-0000-0000-000000000000" with the subscription ID.

2. Create the new role using the JSON file:

```
az role definition create --role-definition "<PathToRoleFile>"
az role assignment create \
  --role "Key Vault resource manager template deployment operator" \
  --assignee $userPrincipalName \
  --resource-group $resourceGroupName
```

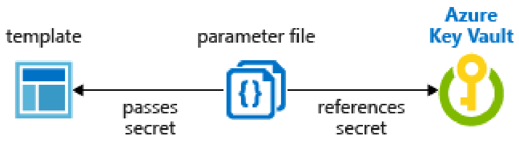
```
New-AzRoleDefinition -InputFile "<PathToRoleFile>"
New-AzRoleAssignment `
  -ResourceGroupName $resourceGroupName `
  -RoleDefinitionName "Key Vault resource manager template deployment operator" `
  -SignInName $userPrincipalName
```

   The samples assign the custom role to the user on the resource group level.

When using a Key Vault with the template for a Managed Application, you must grant access to the **Appliance Resource Provider** service principal. For more information, see Access Key Vault secret when deploying Azure Managed Applications.

Reference secrets with static ID

With this approach, you reference the key vault in the parameter file, not the template. The following image shows how the parameter file references the secret and passes that value to the template.



Tutorial: Integrate Azure Key Vault in Resource Manager Template deployment uses this method.

The following template deploys a SQL server that includes an administrator password. The password parameter is set to a secure string. But, the template doesn't specify where that value comes from.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "adminLogin": {
      "type": "string"
    },
    "adminPassword": {
      "type": "securestring"
    },
    "sqlServerName": {
      "type": "string"
    }
  },
  "resources": [
    {
      "name": "[parameters('sqlServerName')]",
      "type": "Microsoft.Sql/servers",
      "apiVersion": "2015-05-01-preview",
      "location": "[resourceGroup().location]",
      "tags": {},
      "properties": {
        "administratorLogin": "[parameters('adminLogin')]",
        "administratorLoginPassword": "[parameters('adminPassword')]",
        "version": "12.0"
      }
    }
  ],
  "outputs": {
  }
}
```

Now, create a parameter file for the preceding template. In the parameter file, specify a parameter that matches the name of the parameter in the template. For the parameter value, reference the secret from the key vault. You reference the secret by passing the resource identifier of the key vault and the name of the secret:

In the following parameter file, the key vault secret must already exist, and you provide a static value for its resource ID.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "adminLogin": {
            "value": "exampleadmin"
        },
        "adminPassword": {
            "reference": {
              "keyVault": {
                "id": "/subscriptions/<subscription-id>/resourceGroups/<rg-
name>/providers/Microsoft.KeyVault/vaults/<vault-name>"
              },
              "secretName": "ExamplePassword"
            }
        },
        "sqlServerName": {
            "value": "<your-server-name>"
        }
    }
}
```

If you need to use a version of the secret other than the current version, use the `secretVersion` property.

```
"secretName": "ExamplePassword",
"secretVersion": "cd91b2b7e10e492ebb870a6ee0591b68"
```

Deploy the template and pass in the parameter file:

For Azure CLI, use:

```
az group create --name $resourceGroupName --location $location
az group deployment create \
    --resource-group $resourceGroupName \
    --template-uri <The Template File URI> \
    --parameters <The Parameter File>
```

For PowerShell, use:

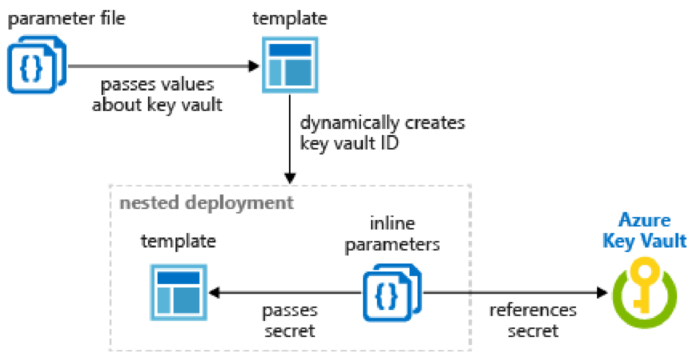
```
New-AzResourceGroup -Name $resourceGroupName -Location $location
New-AzResourceGroupDeployment `
  -ResourceGroupName $resourceGroupName `
  -TemplateUri <The Template File URI> `
  -TemplateParameterFile <The Parameter File>
```

Reference secrets with dynamic ID

The previous section showed how to pass a static resource ID for the key vault secret from the parameter. However, in some scenarios, you need to reference a key vault secret that varies based on the current deployment. Or, you may want to pass parameter values to the template rather than create a reference parameter in the parameter file. In either case, you can dynamically generate the resource ID for a key vault secret by using a linked template.

You can't dynamically generate the resource ID in the parameters file because template expressions aren't allowed in the parameters file.

In your parent template, you add the linked template and pass in a parameter that contains the dynamically generated resource ID. The following image shows how a parameter in the linked template references the secret.



The following template dynamically creates the key vault ID and passes it as a parameter.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "location": {
            "type": "string",
            "defaultValue": "[resourceGroup().location]",
            "metadata": {
                "description": "The location where the resources will be deployed."
            }
        },
        "vaultName": {
            "type": "string",
            "metadata": {
                "description": "The name of the keyvault that contains the secret."
            }
        },
        "secretName": {
            "type": "string",
            "metadata": {
                "description": "The name of the secret."
            }
        },
        "vaultResourceGroupName": {
            "type": "string",
            "metadata": {
                "description": "The name of the resource group that contains the keyvault."
            }
        },
        "vaultSubscription": {
            "type": "string",
            "defaultValue": "[subscription().subscriptionId]",
            "metadata": {
                "description": "The name of the subscription that contains the keyvault."
            }
        },
        "_artifactsLocation": {
            "type": "string",
            "metadata": {
                "description": "The base URI where artifacts required by this template are located. When the
template is deployed using the accompanying scripts, a private location in the subscription will be used and
this value will be automatically generated."
            },
            "defaultValue": "https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/201-
key-vault-use-dynamic-id/"
        },
        "_artifactsLocationSasToken": {
            "type": "securestring",
            "metadata": {
                "description": "The sasToken required to access _artifactsLocation.  When the template is
deployed using the accompanying scripts, a sasToken will be automatically generated."
            },
            "defaultValue": ""
        }
    },
    "resources": [
        {
            "apiVersion": "2018-05-01",
            "name": "dynamicSecret",
            "type": "Microsoft.Resources/deployments",
            "properties": {
                "mode": "Incremental",
                "templateLink": {
                    "contentVersion": "1.0.0.0",
                    "uri": "[uri(parameters('_artifactsLocation'), concat('./nested/sqlserver.json',
parameters('_artifactsLocationSasToken')))]"
                },
                "parameters": {
                    "location": {
                        "value": "[parameters('location')]"
                    },
                    "adminLogin": {
                        "value": "ghuser"
                    },
                    "adminPassword": {
                        "reference": {
                            "keyVault": {
                                "id": "[resourceId(parameters('vaultSubscription'),
parameters('vaultResourceGroupName'), 'Microsoft.KeyVault/vaults', parameters('vaultName'))]"
                            },
                            "secretName": "[parameters('secretName')]"
                        }
                    }
                }
            }
        }
    ],
    "outputs": {
        "sqlFQDN": {
            "type": "string",
            "value": "[reference('dynamicSecret').outputs.sqlFQDN.value]"
        }
    }
}
```

Deploy the preceding template, and provide values for the parameters. You can use the example template from GitHub, but you must provide parameter values for your environment.

For Azure CLI, use:

```
az group create --name $resourceGroupName --location $location
az group deployment create \
    --resource-group $resourceGroupName \
    --template-uri https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/201-key-vault-
use-dynamic-id/azuredeploy.json \
    --parameters vaultName=$keyVaultName vaultResourceGroupName=examplegroup secretName=examplesecret
```

For PowerShell, use:

```
New-AzResourceGroup -Name $resourceGroupName -Location $location
New-AzResourceGroupDeployment `
  -ResourceGroupName $resourceGroupName `
  -TemplateUri https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/201-key-vault-use-
dynamic-id/azuredeploy.json `
  -vaultName $keyVaultName -vaultResourceGroupName $keyVaultResourceGroupName -secretName $secretName
```

Next steps

- For general information about key vaults, see What is Azure Key Vault?.
- For complete examples of referencing key secrets, see Key Vault examples.

Resource, property, or variable iteration in Azure Resource Manager templates

7/25/2019 • 9 minutes to read • Edit Online

This article shows you how to create more than one instance of a resource, variable, or property in your Azure Resource Manager template. To create multiple instances, add the `copy` object to your template.

When used with a resource, the copy object has the following format:

```
"copy": {
    "name": "<name-of-loop>",
    "count": <number-of-iterations>,
    "mode": "serial" <or> "parallel",
    "batchSize": <number-to-deploy-serially>
}
```

When used with a variable or property, the copy object has the following format:

```
"copy": [
  {
      "name": "<name-of-loop>",
      "count": <number-of-iterations>,
      "input": <values-for-the-property-or-variable>
  }
]
```

Both uses are described in greater detail in this article. For a tutorial, see Tutorial: create multiple resource instances using Resource Manager templates.

If you need to specify whether a resource is deployed at all, see condition element.

Copy limits

To specify the number of iterations, you provide a value for the count property. The count can't exceed 800.

The count can't be a negative number. If you deploy a template with REST API version **2019-05-10** or later, you can set count to zero. Earlier versions of the REST API don't support zero for count. Currently, Azure CLI or PowerShell don't support zero for count, but that support will be added in a future release.

Be careful using complete mode deployment with copy. If you redeploy with complete mode to a resource group, any resources that aren't specified in the template after resolving the copy loop are deleted.

The limits for the count are the same whether used with a resource, variable, or property.

Resource iteration

When you must decide during deployment to create one or more instances of a resource, add a `copy` element to the resource type. In the copy element, specify the number of iterations and a name for this loop.

The resource to create several times takes the following format:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "resources": [
    {
      "apiVersion": "2016-01-01",
      "type": "Microsoft.Storage/storageAccounts",
      "name": "[concat(copyIndex(),'storage', uniqueString(resourceGroup().id))]",
      "location": "[resourceGroup().location]",
      "sku": {
        "name": "Standard_LRS"
      },
      "kind": "Storage",
      "properties": {},
      "copy": {
        "name": "storagecopy",
        "count": 3
      }
    }
  ],
  "outputs": {}
}
```

Notice that the name of each resource includes the `copyIndex()` function, which returns the current iteration in the loop. `copyIndex()` is zero-based. So, the following example:

```
"name": "[concat('storage', copyIndex())]",
```

Creates these names:

- storage0
- storage1
- storage2.

To offset the index value, you can pass a value in the copyIndex() function. The number of iterations is still specified in the copy element, but the value of copyIndex is offset by the specified value. So, the following example:

```
"name": "[concat('storage', copyIndex(1))]",
```

Creates these names:

- storage1
- storage2
- storage3

The copy operation is helpful when working with arrays because you can iterate through each element in the array. Use the `length` function on the array to specify the count for iterations, and `copyIndex` to retrieve the current index in the array. So, the following example:

```
"parameters": {
  "org": {
    "type": "array",
    "defaultValue": [
      "contoso",
      "fabrikam",
      "coho"
    ]
  }
},
"resources": [
  {
    "name": "[concat('storage', parameters('org')[copyIndex()])]",
    "copy": {
      "name": "storagecopy",
      "count": "[length(parameters('org'))]"
    },
    ...
  }
]
```

Creates these names:

- storagecontoso
- storagefabrikam
- storagecoho

By default, Resource Manager creates the resources in parallel. It applies no limit to the number of resources deployed in parallel, other than the total limit of 800 resources in the template. The order in which they're created isn't guaranteed.

However, you may want to specify that the resources are deployed in sequence. For example, when updating a production environment, you may want to stagger the updates so only a certain number are updated at any one time. To serially deploy more than one instance of a resource, set `mode` to **serial** and `batchSize` to the number of instances to deploy at a time. With serial mode, Resource Manager creates a dependency on earlier instances in the loop, so it doesn't start one batch until the previous batch completes.

For example, to serially deploy storage accounts two at a time, use:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "resources": [
    {
      "apiVersion": "2016-01-01",
      "type": "Microsoft.Storage/storageAccounts",
      "name": "[concat(copyIndex(),'storage', uniqueString(resourceGroup().id))]",
      "location": "[resourceGroup().location]",
      "sku": {
        "name": "Standard_LRS"
      },
      "kind": "Storage",
      "properties": {},
      "copy": {
        "name": "storagecopy",
        "count": 4,
        "mode": "serial",
        "batchSize": 2
      }
    }
  ],
  "outputs": {}
}
```

The mode property also accepts **parallel**, which is the default value.

For information about using copy with nested templates, see Using copy.

Property iteration

To create more than one value for a property on a resource, add a `copy` array in the properties element. This array contains objects, and each object has the following properties:

- name - the name of the property to create several values for
- count - the number of values to create.
- input - an object that contains the values to assign to the property

The following example shows how to apply `copy` to the dataDisks property on a virtual machine:

```
{
  "name": "examplevm",
  "type": "Microsoft.Compute/virtualMachines",
  "apiVersion": "2017-03-30",
  "properties": {
    "storageProfile": {
      "copy": [{
        "name": "dataDisks",
        "count": 3,
        "input": {
          "lun": "[copyIndex('dataDisks')]",
          "createOption": "Empty",
          "diskSizeGB": "1023"
        }
      }],
      ...
```

Notice that when using `copyIndex` inside a property iteration, you must provide the name of the iteration. You don't have to provide the name when used with resource iteration.

Resource Manager expands the `copy` array during deployment. The name of the array becomes the name of the property. The input values become the object properties. The deployed template becomes:

```
{
  "name": "examplevm",
  "type": "Microsoft.Compute/virtualMachines",
  "apiVersion": "2017-03-30",
  "properties": {
    "storageProfile": {
      "dataDisks": [
        {
          "lun": 0,
          "createOption": "Empty",
          "diskSizeGB": "1023"
        },
        {
          "lun": 1,
          "createOption": "Empty",
          "diskSizeGB": "1023"
        },
        {
          "lun": 2,
          "createOption": "Empty",
          "diskSizeGB": "1023"
        }
      ],
      ...
```

The copy element is an array so you can specify more than one property for the resource. Add an object for each property to create.

```
{
  "name": "string",
  "type": "Microsoft.Network/loadBalancers",
  "apiVersion": "2017-10-01",
  "properties": {
    "copy": [
      {
        "name": "loadBalancingRules",
        "count": "[length(parameters('loadBalancingRules'))]",
        "input": {
          ...
        }
      },
      {
        "name": "probes",
        "count": "[length(parameters('loadBalancingRules'))]",
        "input": {
          ...
        }
      }
    ]
  }
}
```

You can use resource and property iteration together. Reference the property iteration by name.

```
{
  "type": "Microsoft.Network/virtualNetworks",
  "name": "[concat(parameters('vnetname'), copyIndex())]",
  "apiVersion": "2018-04-01",
  "copy":{
    "count": 2,
    "name": "vnetloop"
  },
  "location": "[resourceGroup().location]",
  "properties": {
    "addressSpace": {
      "addressPrefixes": [
        "[parameters('addressPrefix')]"
      ]
    },
    "copy": [
      {
        "name": "subnets",
        "count": 2,
        "input": {
          "name": "[concat('subnet-', copyIndex('subnets'))]",
          "properties": {
            "addressPrefix": "[variables('subnetAddressPrefix')[copyIndex('subnets')]]"
          }
        }
      }
    ]
  }
}
```

Variable iteration

To create multiple instances of a variable, use the `copy` property in the variables section. You create an array of elements constructed from the value in the `input` property. You can use the `copy` property within a variable, or at the top level of the variables section. When using `copyIndex` inside a variable iteration, you must provide the name of the iteration.

For a simple example of creating an array of string values, see copy array template.

The following example shows several different ways to create array variables with dynamically constructed elements. It shows how to use copy inside a variable to create arrays of objects and strings. It also shows how to use copy at the top level to create arrays of objects, strings, and integers.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {},
  "variables": {
    "disk-array-on-object": {
      "copy": [
        {
          "name": "disks",
          "count": 5,
          "input": {
            "name": "[concat('myDataDisk', copyIndex('disks', 1))]",
            "diskSizeGB": "1",
            "diskIndex": "[copyIndex('disks')]"
          }
        },
        {
          "name": "diskNames",
          "count": 5,
          "input": "[concat('myDataDisk', copyIndex('diskNames', 1))]"
        }
      ]
    },
    "copy": [
      {
        "name": "top-level-object-array",
        "count": 5,
        "input": {
          "name": "[concat('myDataDisk', copyIndex('top-level-object-array', 1))]",
          "diskSizeGB": "1",
          "diskIndex": "[copyIndex('top-level-object-array')]"
        }
      },
      {
        "name": "top-level-string-array",
        "count": 5,
        "input": "[concat('myDataDisk', copyIndex('top-level-string-array', 1))]"
      },
      {
        "name": "top-level-integer-array",
        "count": 5,
        "input": "[copyIndex('top-level-integer-array')]"
      }
    ]
  },
  "resources": [],
  "outputs": {
    "exampleObject": {
      "value": "[variables('disk-array-on-object')]",
      "type": "object"
    },
    "exampleArrayOnObject": {
      "value": "[variables('disk-array-on-object').disks]",
      "type" : "array"
    },
    "exampleObjectArray": {
      "value": "[variables('top-level-object-array')]",
      "type" : "array"
    },
    "exampleStringArray": {
      "value": "[variables('top-level-string-array')]",
      "type" : "array"
    },
    "exampleIntegerArray": {
      "value": "[variables('top-level-integer-array')]",
      "type" : "array"
    }
  }
}
```

The type of variable that gets created depends on the input object. For example, the variable named **top-level-object-array** in the preceding example returns:

```
[
  {
    "name": "myDataDisk1",
    "diskSizeGB": "1",
    "diskIndex": 0
  },
  {
    "name": "myDataDisk2",
    "diskSizeGB": "1",
    "diskIndex": 1
  },
  {
    "name": "myDataDisk3",
    "diskSizeGB": "1",
    "diskIndex": 2
  },
  {
    "name": "myDataDisk4",
    "diskSizeGB": "1",
    "diskIndex": 3
  },
  {
    "name": "myDataDisk5",
    "diskSizeGB": "1",
    "diskIndex": 4
  }
]
```

And, the variable named **top-level-string-array** returns:

```
[
  "myDataDisk1",
  "myDataDisk2",
  "myDataDisk3",
  "myDataDisk4",
  "myDataDisk5"
]
```

Depend on resources in a loop

You specify that a resource is deployed after another resource by using the `dependsOn` element. To deploy a resource that depends on the collection of resources in a loop, provide the name of the copy loop in the dependsOn element. The following example shows how to deploy three storage accounts before deploying the Virtual Machine. The full Virtual Machine definition isn't shown. Notice that the copy element has name set to `storagecopy` and the dependsOn element for the Virtual Machines is also set to `storagecopy`.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {},
  "resources": [
    {
      "apiVersion": "2016-01-01",
      "type": "Microsoft.Storage/storageAccounts",
      "name": "[concat(copyIndex(),'storage', uniqueString(resourceGroup().id))]",
      "location": "[resourceGroup().location]",
      "sku": {
        "name": "Standard_LRS"
      },
      "kind": "Storage",
      "properties": {},
      "copy": {
        "name": "storagecopy",
        "count": 3
      }
    },
    {
      "apiVersion": "2015-06-15",
      "type": "Microsoft.Compute/virtualMachines",
      "name": "[concat('VM', uniqueString(resourceGroup().id))]",
      "dependsOn": ["storagecopy"],
      ...
    }
  ],
  "outputs": {}
}
```

Iteration for a child resource

You can't use a copy loop for a child resource. To create more than one instance of a resource that you typically define as nested within another resource, you must instead create that resource as a top-level resource. You define the relationship with the parent resource through the type and name properties.

For example, suppose you typically define a dataset as a child resource within a data factory.

```
"resources": [
{
  "type": "Microsoft.DataFactory/datafactories",
  "name": "exampleDataFactory",
  ...
  "resources": [
    {
      "type": "datasets",
      "name": "exampleDataSet",
      "dependsOn": [
        "exampleDataFactory"
      ],
      ...
    }
  ]
```

To create more than one data set, move it outside of the data factory. The dataset must be at the same level as the data factory, but it's still a child resource of the data factory. You preserve the relationship between data set and data factory through the type and name properties. Since type can no longer be inferred from its position in the template, you must provide the fully qualified type in the format: `{resource-provider-namespace}/{parent-resource-type}/{child-resource-type}`.

To establish a parent/child relationship with an instance of the data factory, provide a name for the data set that includes the parent resource name. Use the format: `{parent-resource-name}/{child-resource-name}`.

The following example shows the implementation:

```
"resources": [
{
  "type": "Microsoft.DataFactory/datafactories",
  "name": "exampleDataFactory",
  ...
},
{
  "type": "Microsoft.DataFactory/datafactories/datasets",
  "name": "[concat('exampleDataFactory', '/', 'exampleDataSet', copyIndex())]",
  "dependsOn": [
    "exampleDataFactory"
  ],
  "copy": {
    "name": "datasetcopy",
    "count": "3"
  },
  ...
}]
```

Example templates

The following examples show common scenarios for creating more than one instance of a resource or property.

TEMPLATE	DESCRIPTION
Copy storage	Deploys more than one storage account with an index number in the name.
Serial copy storage	Deploys several storage accounts one at time. The name includes the index number.
Copy storage with array	Deploys several storage accounts. The name includes a value from an array.
VM deployment with a variable number of data disks	Deploys several data disks with a virtual machine.
Copy variables	Demonstrates the different ways of iterating on variables.
Multiple security rules	Deploys several security rules to a network security group. It constructs the security rules from a parameter. For the parameter, see multiple NSG parameter file.

Next steps

- To go through a tutorial, see Tutorial: create multiple resource instances using Resource Manager templates.
- If you want to learn about the sections of a template, see Authoring Azure Resource Manager Templates.
- To learn how to deploy your template, see Deploy an application with Azure Resource Manager Template.

Provide post-deployment configurations by using extensions

12/14/2018 • 2 minutes to read • Edit Online

Template extensions are small applications that provide post-deployment configuration and automation tasks on Azure resources. The most popular one is virtual machine extensions. See Virtual machine extensions and features for Windows, and Virtual machine extensions and features for Linux.

Extensions

The existing extensions are:

- Microsoft.Compute/virtualMachines/extensions
- Microsoft.Compute virtualMachineScaleSets/extensions
- Microsoft.HDInsight clusters/extensions
- Microsoft.Sql servers/databases/extensions
- Microsoft.Web/sites/siteextensions

To find out the available extensions, browse to the template reference. In **Filter by title**, enter **extension**.

To learn how to use these extensions, see:

- Tutorial: Deploy virtual machine extensions with Azure Resource Manager templates.
- Tutorial: Import SQL BACPAC files with Azure Resource Manager templates

Next steps

Tutorial: Deploy virtual machine extensions with Azure Resource Manager templates

Deploy resources with Resource Manager templates and Azure portal

6/28/2019 • 3 minutes to read • Edit Online

Learn how to use the Azure portal with Azure Resource Manager to deploy your Azure resources. To learn about managing your resources, see Manage Azure resources by using the Azure portal.

Deploying Azure resources by using the Azure portal usually involves two steps:

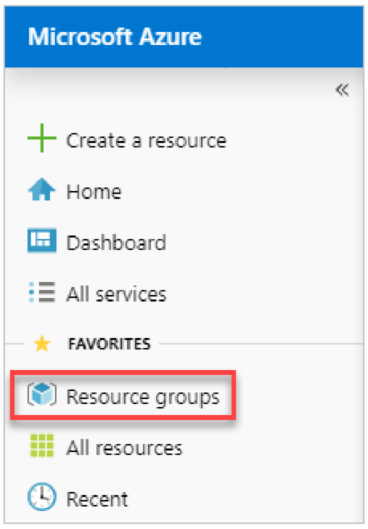
- Create a resource group.
- Deploy resources to the resource group.

In addition, you can also deploy an Azure Resource Manager template to create Azure resources.

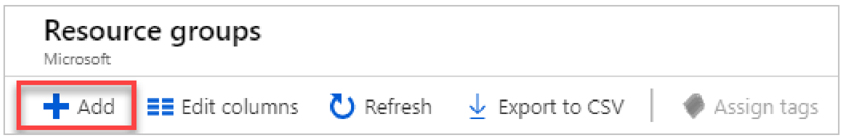
This article shows both methods.

Create a resource group

1. To create a new resource group, select **Resource groups** from the Azure portal.

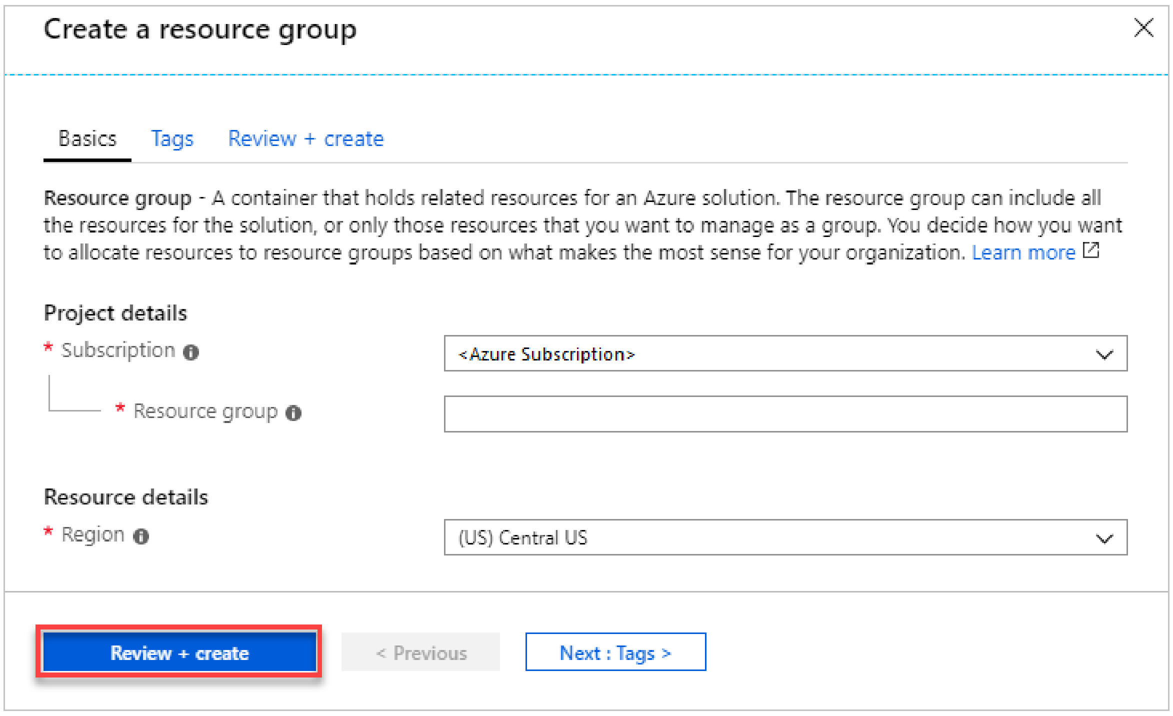


2. Under Resource groups, select **Add**.



3. Select or enter the following property values:

   - **Subscription**: Select an Azure subscription.
   - **Resource group**: Give the resource group a name.
   - **Region**: Specify an Azure location. This is where the resource group stores metadata about the resources. For compliance reasons, you may want to specify where that metadata is stored. In general, we recommend that you specify a location where most of your resources will reside. Using the same location can simplify your template.


Create a resource group

Basics | Tags | Review + create

Resource group - A container that holds related resources for an Azure solution. The resource group can include all the resources for the solution, or only those resources that you want to manage as a group. You decide how you want to allocate resources to resource groups based on what makes the most sense for your organization. Learn more

Project details

* Subscription: <Azure Subscription>

* Resource group:

Resource details

* Region: (US) Central US

4. Select **Review + create**.

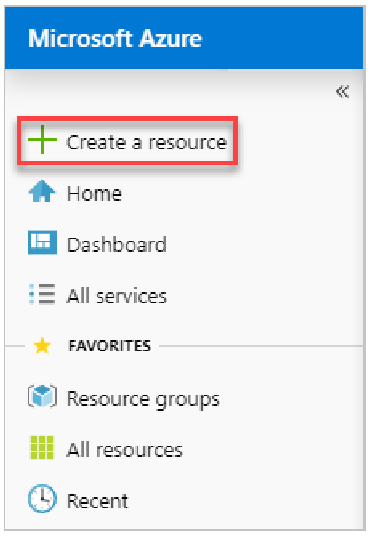
5. review the values, and then select **Create**.

6. Select **Refresh** before you can see the new resource group in the list.

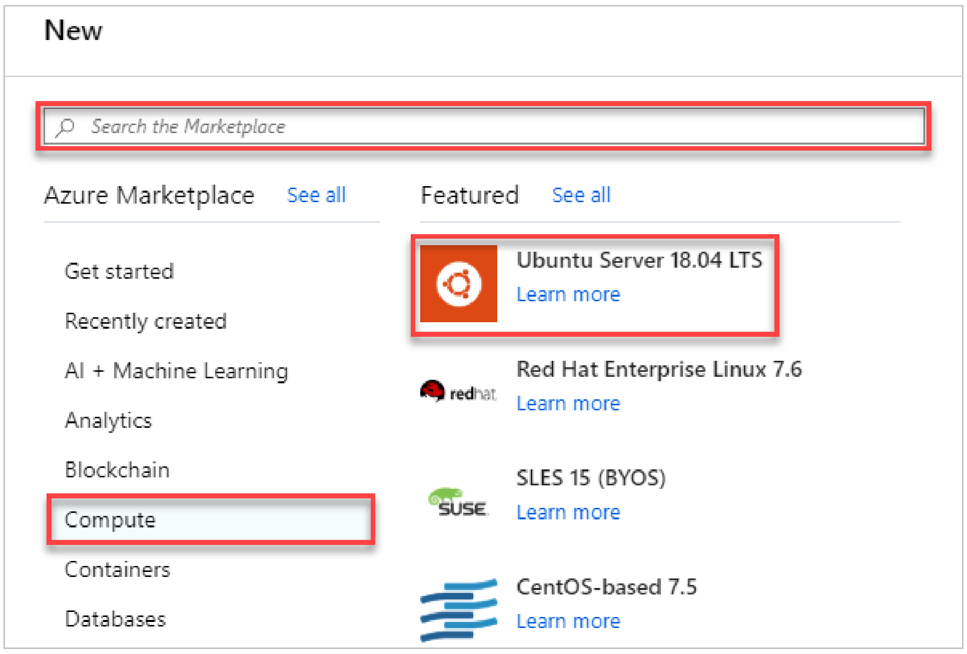
Deploy resources to a resource group

After you create a resource group, you can deploy resources to the group from the Marketplace. The Marketplace provides pre-defined solutions for common scenarios.

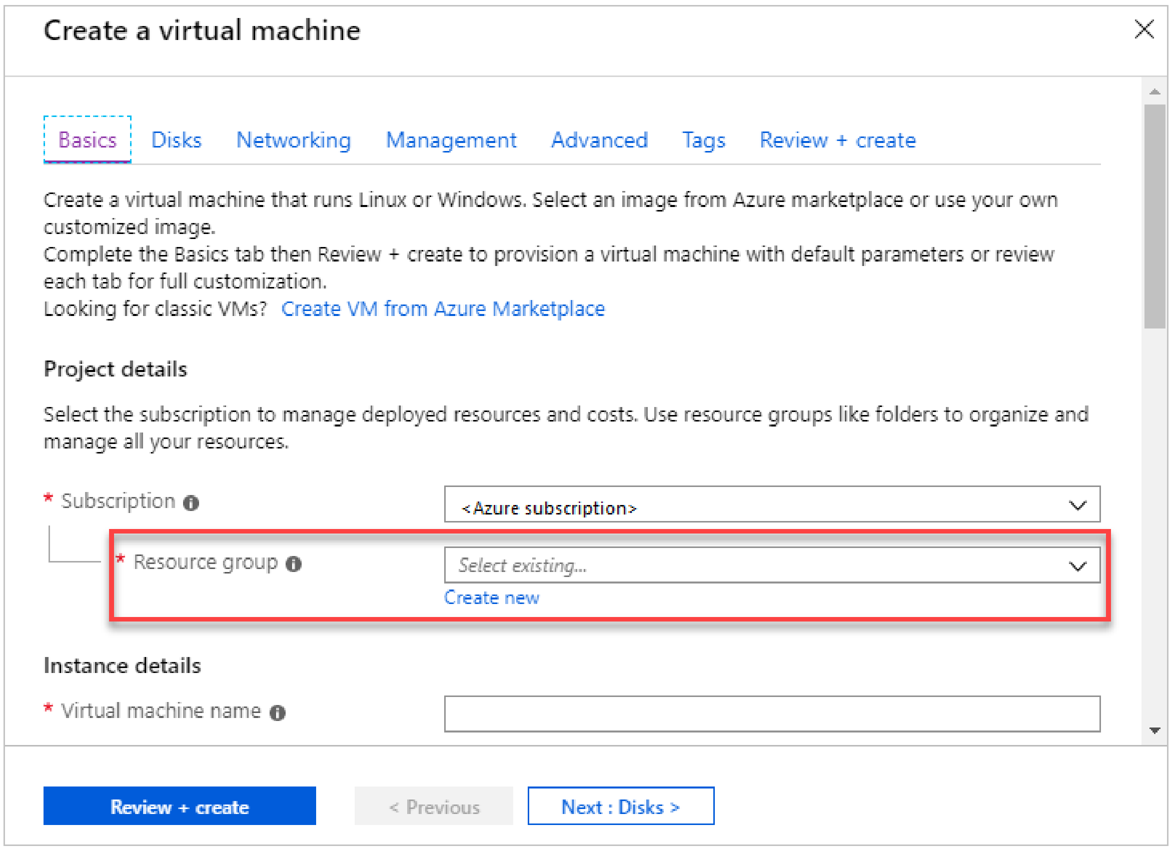
1. To start a deployment, select **Create a resource** from the Azure portal.



2. Find the type of resource you would like to deploy. The resources are organized in categories. If you don't see the particular solution you would like to deploy, you can search the Marketplace for it. The following screenshot shows that Ubuntu Server is selected.



3. Depending on the type of selected resource, you have a collection of relevant properties to set before deployment. For all types, you must select a destination resource group. The following image shows how to create a Linux virtual machine and deploy it to the resource group you created.



Create a virtual machine that runs Linux or Windows. Select an image from Azure marketplace or use your own customized image.
Complete the Basics tab then Review + create to provision a virtual machine with default parameters or review each tab for full customization.
Looking for classic VMs? Create VM from Azure Marketplace

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

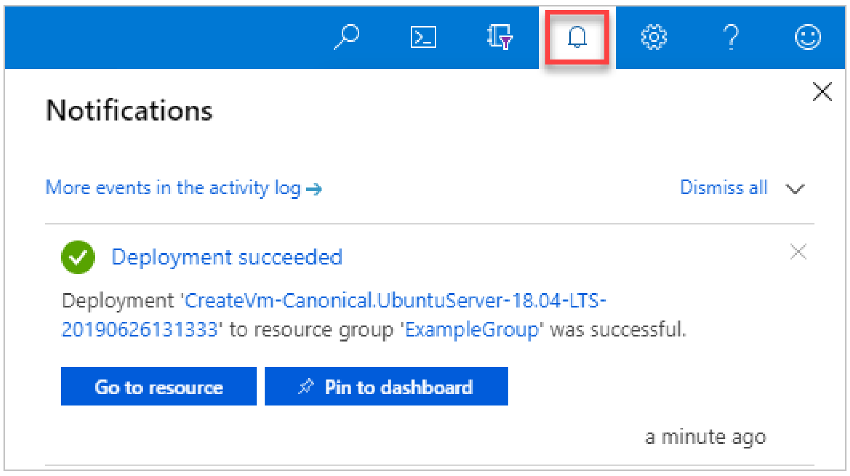
* Subscription <Azure subscription>
* Resource group Select existing...
Create new

Instance details

* Virtual machine name

Alternatively, you can decide to create a resource group when deploying your resources. Select **Create new** and give the resource group a name.

4. Your deployment begins. The deployment could take several minutes. Some resources take longer time than other resources. When the deployment has finished, you see a notification. Select **Go to resource** to open


Notifications

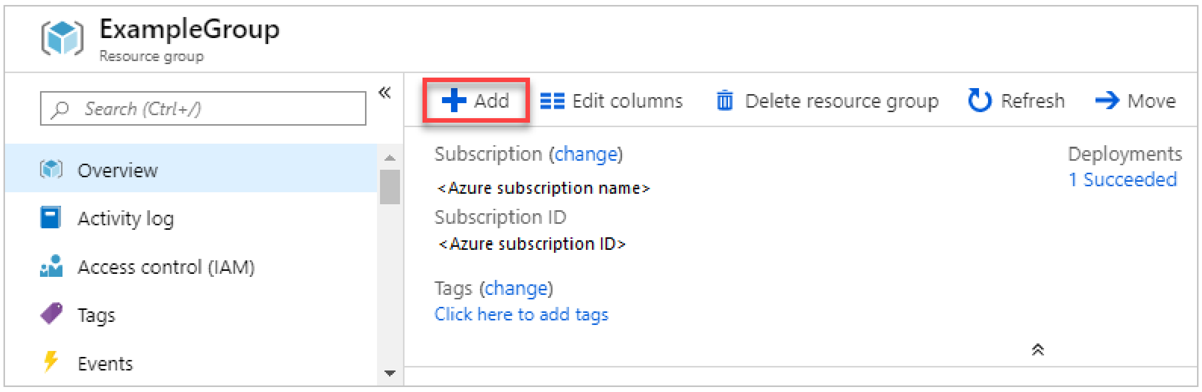
More events in the activity log →

Dismiss all

Deployment succeeded

Deployment 'CreateVm-Canonical.UbuntuServer-18.04-LTS-20190626131333' to resource group 'ExampleGroup' was successful.

5. After deploying your resources, you can add more resources to the resource group by selecting **Add**.


ExampleGroup
Resource group

Subscription (change)
<Azure subscription name>
Subscription ID
<Azure subscription ID>
Tags (change)
Click here to add tags

Deployments
1 Succeeded

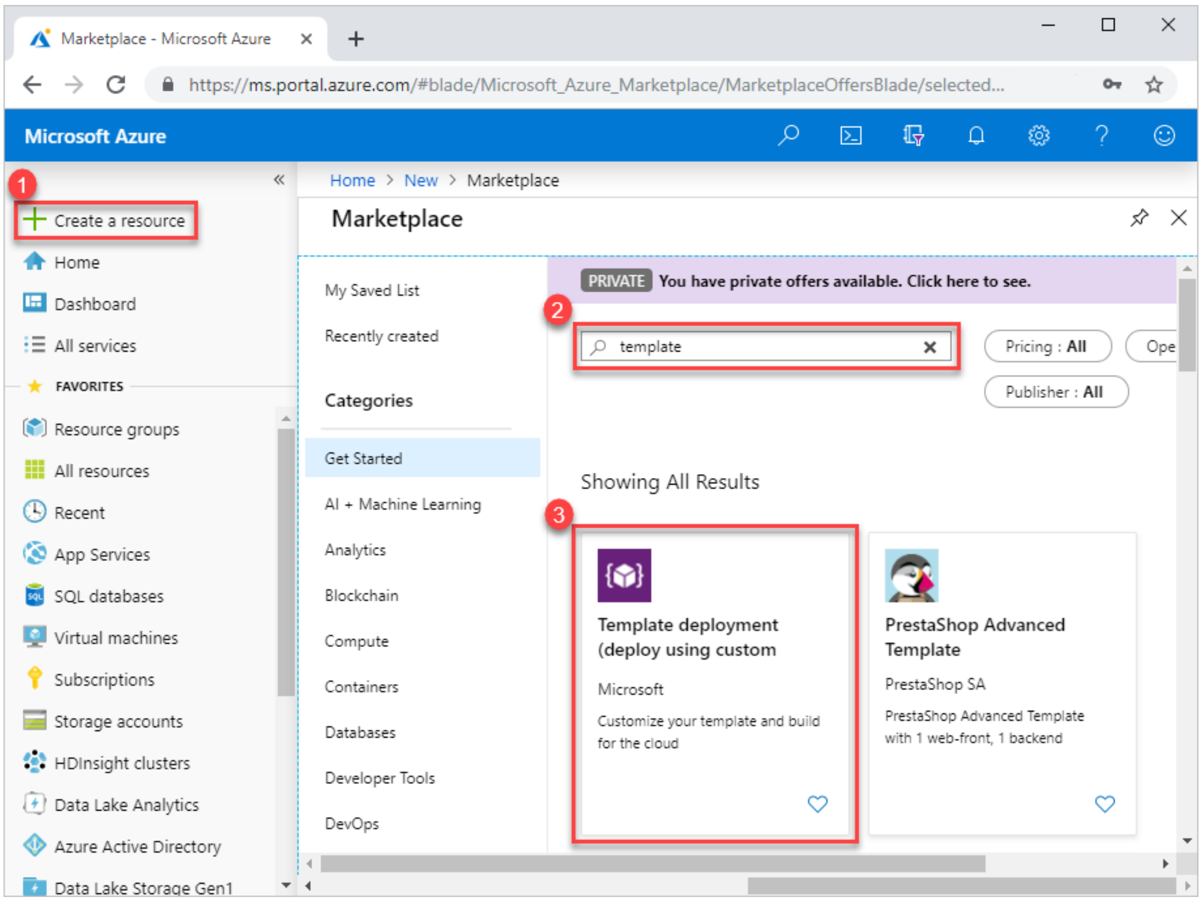
Deploy resources from custom template

If you want to execute a deployment but not use any of the templates in the Marketplace, you can create a customized template that defines the infrastructure for your solution. To learn about creating templates, see Understand the structure and syntax of Azure Resource Manager templates.

NOTE

The portal interface doesn't support referencing a secret from a Key Vault. Instead, use PowerShell or Azure CLI to deploy your template locally or from an external URI.

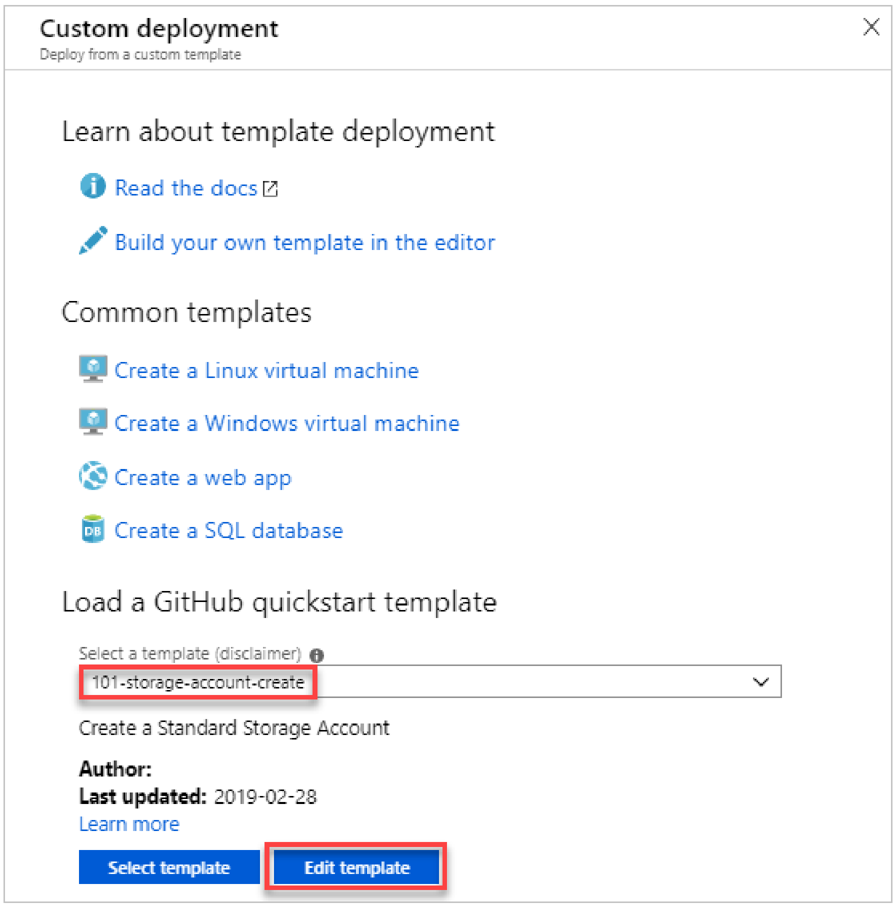
1. To deploy a customized template through the portal, select **Create a resource**, search for **template**. and then select **Template deployment**.



2. Select **Create**.

3. You see several options for creating a template:

- **Build your own template in editor**: create a template using the portal template editor. The editor is capable to add a resource template schema.
- **Common templates**: There are four common templatess for creating a Linux virtual machine, Windows virtual machine, a web application, and an Azure SQL database.
- **Load a GitHub quickstart template**: use an existing quickstart templates.



Custom deployment
Deploy from a custom template

Learn about template deployment

Read the docs

Build your own template in the editor

Common templates

Create a Linux virtual machine

Create a Windows virtual machine

Create a web app

Create a SQL database

Load a GitHub quickstart template

Select a template (disclaimer)
101-storage-account-create

Create a Standard Storage Account

Author:
Last updated: 2019-02-28
Learn more

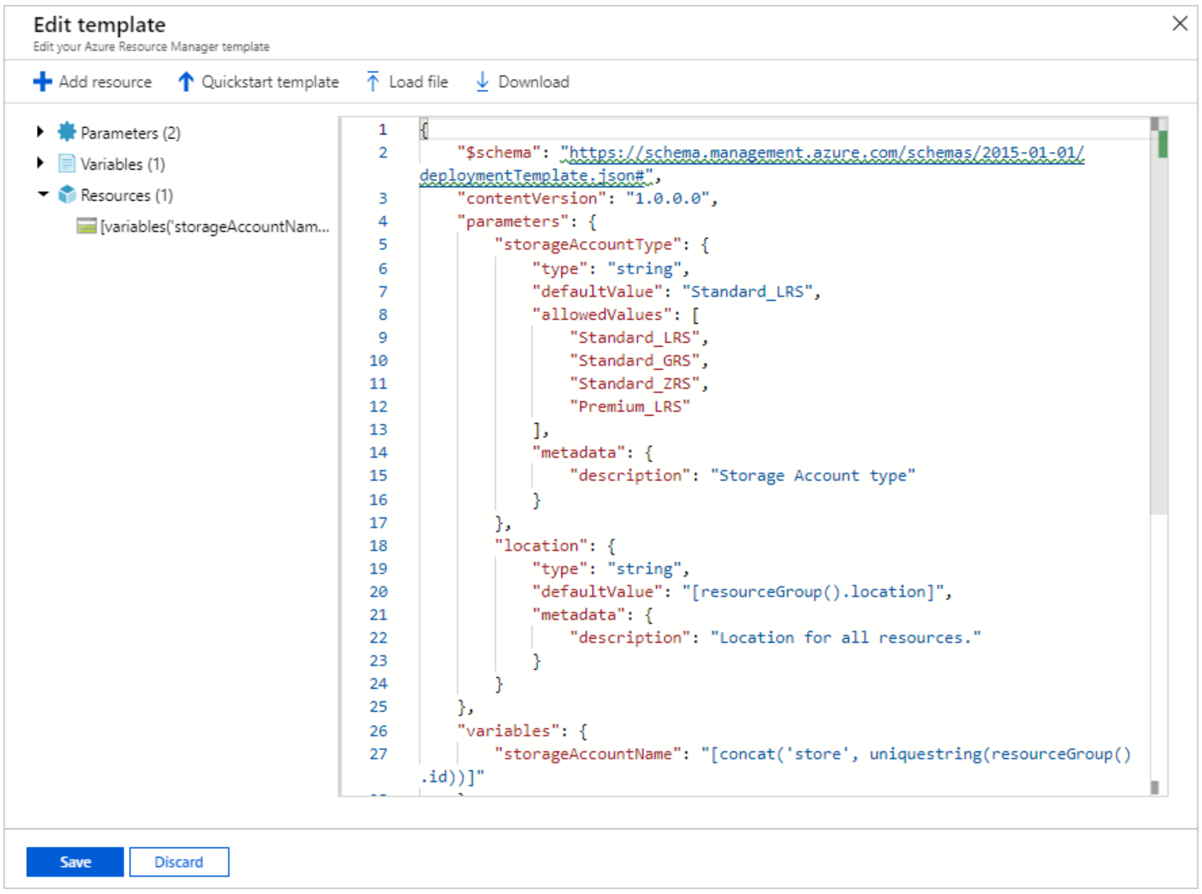
This tutorial provides the instruction for loading a quickstart template.

4. Under **Load a GitHub quickstart template**, type or select **101-storage-account-create**.

   You have two options:

   - **Select template**: deploy the template.
   - **Edit template**: edit the quickstart template before you deploy it.

5. Select **Edit template** to explore the portal template editor. The template is loaded in the editor. Notice there are two parameters: **storageAccountType** and **location**.



```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/
deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "storageAccountType": {
            "type": "string",
            "defaultValue": "Standard_LRS",
            "allowedValues": [
                "Standard_LRS",
                "Standard_GRS",
                "Standard_ZRS",
                "Premium_LRS"
            ],
            "metadata": {
                "description": "Storage Account type"
            }
        },
        "location": {
            "type": "string",
            "defaultValue": "[resourceGroup().location]",
            "metadata": {
                "description": "Location for all resources."
            }
        }
    },
    "variables": {
        "storageAccountName": "[concat('store', uniquestring(resourceGroup()
.id))]"
```

6. Make a minor change to the template. For example, update the **storageAccountName** variable to:

```
"storageAccountName": "[concat('azstore', uniquestring(resourceGroup().id))]"
```

7. Select **Save**. Now you see the portal template deployment interface. Notice the two parameters that you defined in the template.

8. Enter or select the property values:

   - **Subscription**: Select an Azure subscription.
   - **Resource group**: Select **Create new** and give a name.
   - **Location**: Select an Azure location.
   - **Storage Account Type**: Use the default value.
   - **Location**: Use the default value.
   - **I agree to the terms and conditions stated above**: (select)

9. Select **Purchase**.

Next steps

- To view audit logs, see Audit operations with Resource Manager.
- To troubleshoot deployment errors, see View deployment operations.
- To export a template from a deployment or resource group, see Export Azure Resource Manager templates.
- To safely roll out your service across multiple regions, see Azure Deployment Manager.

Deploy resources with Resource Manager templates and Azure CLI

7/12/2019 • 9 minutes to read • Edit Online

This article explains how to use Azure CLI with Resource Manager templates to deploy your resources to Azure. If you aren't familiar with the concepts of deploying and managing your Azure solutions, see Azure Resource Manager overview.

To run this sample, install the latest version of the Azure CLI. To start, run `az login` to create a connection with Azure.

Samples for the Azure CLI are written for the `bash` shell. To run this sample in Windows PowerShell or Command Prompt, you may need to change elements of the script.

If you don't have Azure CLI installed, you can use the Cloud Shell.

Deployment scope

You can target your deployment to either an Azure subscription or a resource group within a subscription. In most cases, you'll target deployment to a resource group. Use subscription deployments to apply policies and role assignments across the subscription. You also use subscription deployments to create a resource group and deploy resources to it. Depending on the scope of the deployment, you use different commands.

To deploy to a **resource group**, use az group deployment create:

```
az group deployment create --resource-group <resource-group-name> --template-file <path-to-template>
```

To deploy to a **subscription**, use az deployment create:

```
az deployment create --location <location> --template-file <path-to-template>
```

Currently, management group deployments are only supported through the REST API. See Deploy resources with Resource Manager templates and Resource Manager REST API.

The examples in this article use resource group deployments. For more information about subscription deployments, see Create resource groups and resources at the subscription level.

Deploy local template

When deploying resources to Azure, you:

1. Sign in to your Azure account
2. Create a resource group that serves as the container for the deployed resources. The name of the resource group can only include alphanumeric characters, periods, underscores, hyphens, and parenthesis. It can be up to 90 characters. It can't end in a period.
3. Deploy to the resource group the template that defines the resources to create

A template can include parameters that enable you to customize the deployment. For example, you can provide values that are tailored for a particular environment (such as dev, test, and production). The sample template defines a parameter for the storage account SKU.

The following example creates a resource group, and deploys a template from your local machine:

```
az group create --name ExampleGroup --location "Central US"
az group deployment create \
  --name ExampleDeployment \
  --resource-group ExampleGroup \
  --template-file storage.json \
  --parameters storageAccountType=Standard_GRS
```

The deployment can take a few minutes to complete. When it finishes, you see a message that includes the result:

```
"provisioningState": "Succeeded",
```

Deploy remote template

Instead of storing Resource Manager templates on your local machine, you may prefer to store them in an external location. You can store templates in a source control repository (such as GitHub). Or, you can store them in an Azure storage account for shared access in your organization.

To deploy an external template, use the **template-uri** parameter. Use the URI in the example to deploy the sample template from GitHub.

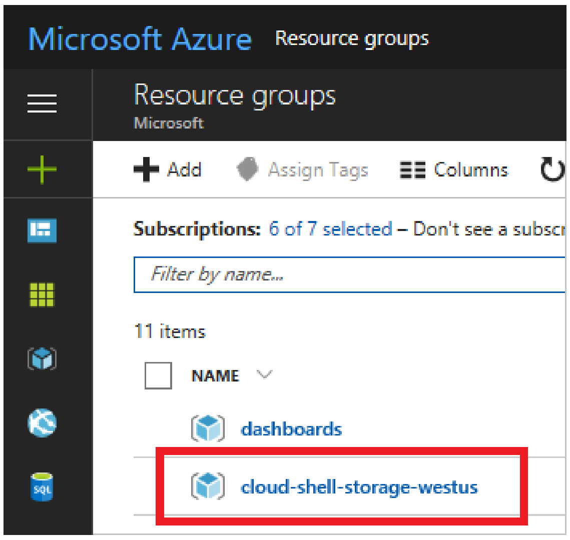
```
az group create --name ExampleGroup --location "Central US"
az group deployment create \
  --name ExampleDeployment \
  --resource-group ExampleGroup \
  --template-uri "https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-storage-account-create/azuredeploy.json" \
  --parameters storageAccountType=Standard_GRS
```

The preceding example requires a publicly accessible URI for the template, which works for most scenarios because your template shouldn't include sensitive data. If you need to specify sensitive data (like an admin password), pass that value as a secure parameter. However, if you don't want your template to be publicly accessible, you can protect it by storing it in a private storage container. For information about deploying a template that requires a shared access signature (SAS) token, see Deploy private template with SAS token.

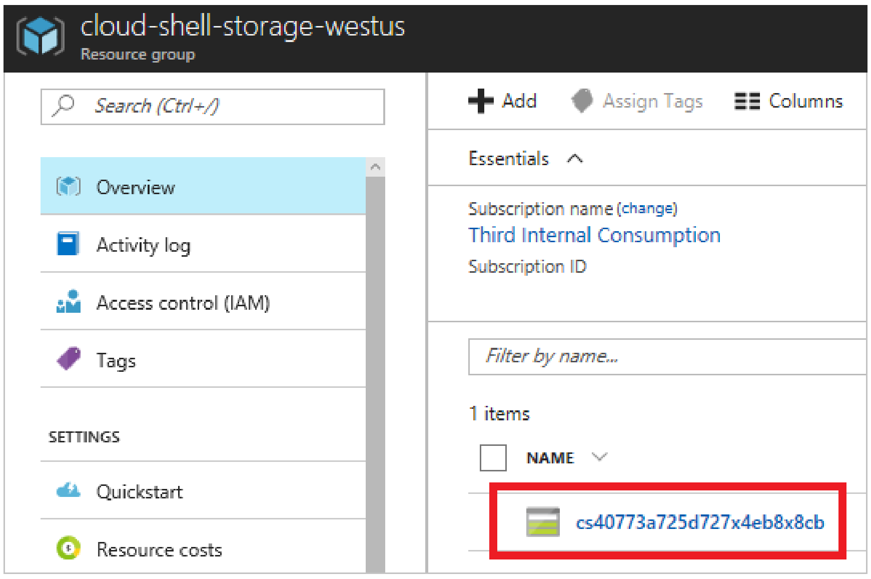
Deploy template from Cloud Shell

You can use Cloud Shell to deploy your template. To deploy an external template, provide the URI of the template exactly as you would for any external deployment. To deploy a local template, you must first load your template into the storage account for your Cloud Shell. This section describes how to load the template to your cloud shell account, and deploy it as a local file. If you haven't used Cloud Shell, see Overview of Azure Cloud Shell for information about setting it up.

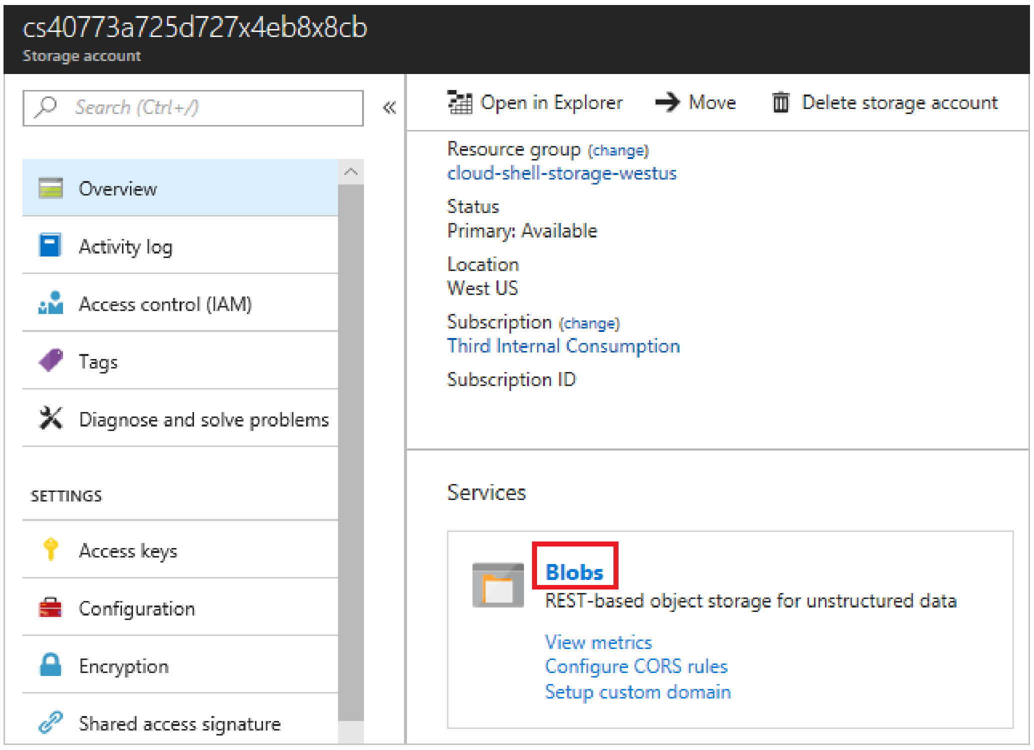
1. Sign in to the Azure portal.
2. Select your Cloud Shell resource group. The name pattern is `cloud-shell-storage-<region>`.



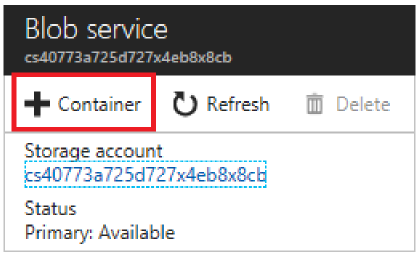
3. Select the storage account for your Cloud Shell.



4. Select **Blobs**.

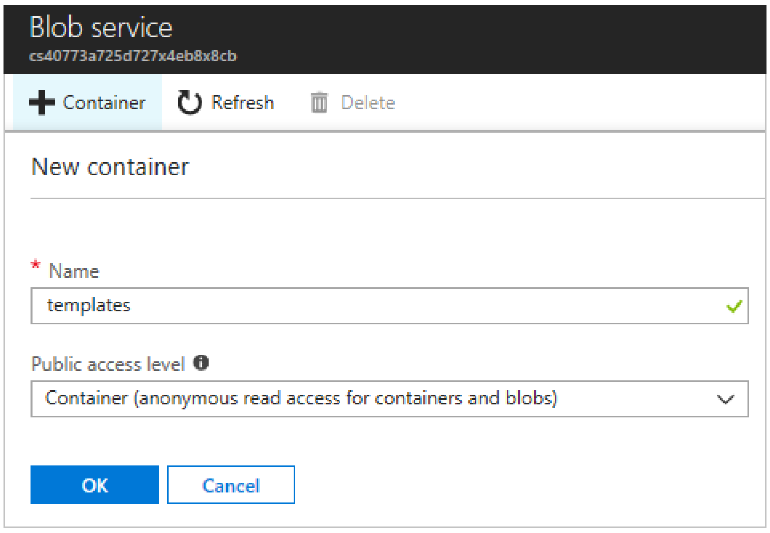


5. Select **+ Container**.



Storage account
cs40773a725d727x4eb8x8cb
Status
Primary: Available

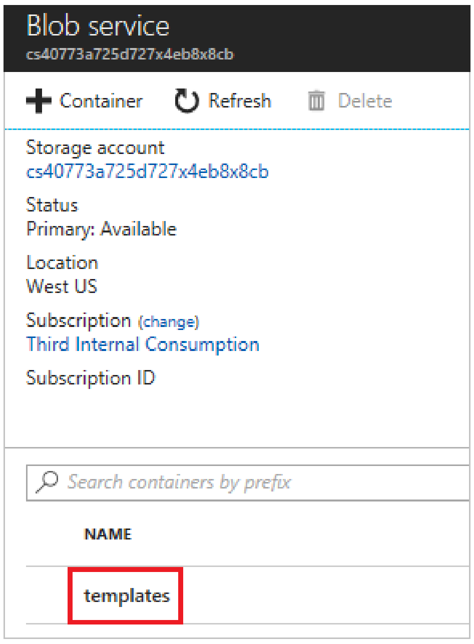
6. Give your container a name and an access level. The sample template in this article contains no sensitive information, so allow anonymous read access. Select **OK**.



* Name
templates

Public access level
Container (anonymous read access for containers and blobs)

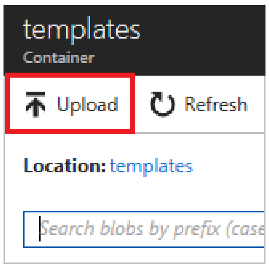
7. Select the container you created.



Storage account
cs40773a725d727x4eb8x8cb
Status
Primary: Available
Location
West US
Subscription (change)
Third Internal Consumption
Subscription ID

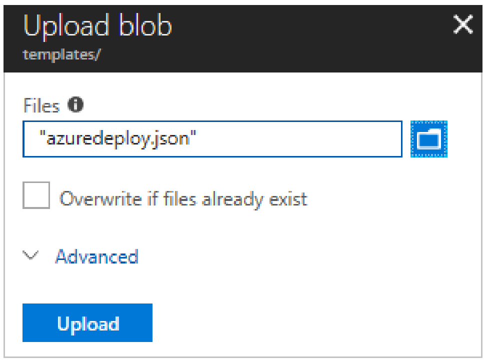
NAME
templates

8. Select **Upload**.

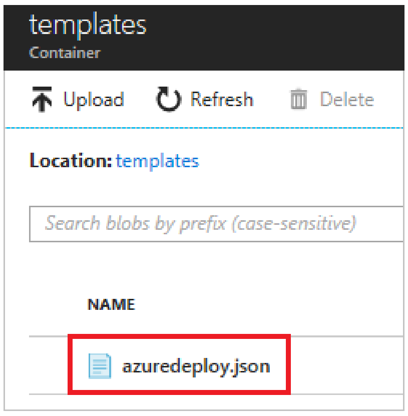


Location: templates

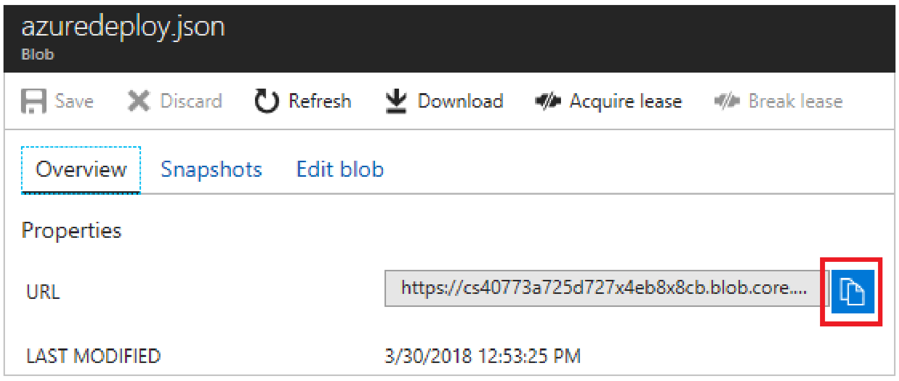
9. Find and upload your template.



10. After it has uploaded, select the template.



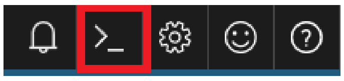
11. Copy the URL.



URL https://cs40773a725d727x4eb8x8cb.blob.core....

LAST MODIFIED 3/30/2018 12:53:25 PM

12. Open the prompt.



In the Cloud Shell, use the following commands:

```
az group create --name examplegroup --location "South Central US"
az group deployment create --resource-group examplegroup \
  --template-uri <copied URL> \
  --parameters storageAccountType=Standard_GRS
```

Redeploy when deployment fails

This feature is also known as *Rollback on error*. When a deployment fails, you can automatically redeploy an earlier, successful deployment from your deployment history. To specify redeployment, use the `--rollback-on-error` parameter in the deployment command. This functionality is useful if you've got a known good state for your infrastructure deployment and want to revert to this state. There are a number of caveats and restrictions:

- The redeployment is run exactly as it was run previously with the same parameters. You can't change the parameters.
- The previous deployment is run using the complete mode. Any resources not included in the previous deployment are deleted, and any resource configurations are set to their previous state. Make sure you fully understand the deployment modes.
- The redeployment only affects the resources, any data changes aren't affected.
- This feature is only supported on Resource Group deployments, not subscription level deployments. For more information about subscription level deployment, see Create resource groups and resources at the subscription level.

To use this option, your deployments must have unique names so they can be identified in the history. If you don't have unique names, the current failed deployment might overwrite the previously successful deployment in the history. You can only use this option with root level deployments. Deployments from a nested template aren't available for redeployment.

To redeploy the last successful deployment, add the `--rollback-on-error` parameter as a flag.

```
az group deployment create \
  --name ExampleDeployment \
  --resource-group ExampleGroup \
  --template-file storage.json \
  --parameters storageAccountType=Standard_GRS \
  --rollback-on-error
```

To redeploy a specific deployment, use the `--rollback-on-error` parameter and provide the name of the deployment.

```
az group deployment create \
  --name ExampleDeployment02 \
  --resource-group ExampleGroup \
  --template-file storage.json \
  --parameters storageAccountType=Standard_GRS \
  --rollback-on-error ExampleDeployment01
```

The specified deployment must have succeeded.

Parameters

To pass parameter values, you can use either inline parameters or a parameter file. The preceding examples in this article show inline parameters.

Inline parameters

To pass inline parameters, provide the values in `parameters`. For example, to pass a string and array to a template is a Bash shell, use:

```
az group deployment create \
  --resource-group testgroup \
  --template-file demotemplate.json \
  --parameters exampleString='inline string' exampleArray='("value1", "value2")'
```

If you are using Azure CLI with Windows Command Prompt (CMD) or PowerShell, pass the array in the format: `exampleArray="['value1','value2']"`.

You can also get the contents of file and provide that content as an inline parameter.

```
az group deployment create \
  --resource-group testgroup \
  --template-file demotemplate.json \
  --parameters exampleString=@stringContent.txt exampleArray=@arrayContent.json
```

Getting a parameter value from a file is helpful when you need to provide configuration values. For example, you can provide cloud-init values for a Linux virtual machine.

The arrayContent.json format is:

```
[
    "value1",
    "value2"
]
```

Parameter files

Rather than passing parameters as inline values in your script, you may find it easier to use a JSON file that contains the parameter values. The parameter file must be a local file. External parameter files aren't supported with Azure CLI.

The parameter file must be in the following format:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
     "storageAccountType": {
         "value": "Standard_GRS"
     }
  }
}
```

Notice that the parameters section includes a parameter name that matches the parameter defined in your template (storageAccountType). The parameter file contains a value for the parameter. This value is automatically passed to the template during deployment. You can create more than one parameter file, and then pass in the appropriate parameter file for the scenario.

Copy the preceding example and save it as a file named `storage.parameters.json`.

To pass a local parameter file, use `@` to specify a local file named storage.parameters.json.

```
az group deployment create \
  --name ExampleDeployment \
  --resource-group ExampleGroup \
  --template-file storage.json \
  --parameters @storage.parameters.json
```

Parameter precedence

You can use inline parameters and a local parameter file in the same deployment operation. For example, you can specify some values in the local parameter file and add other values inline during deployment. If you provide values for a parameter in both the local parameter file and inline, the inline value takes precedence.

```
az group deployment create \
  --resource-group testgroup \
  --template-file demotemplate.json \
  --parameters @demotemplate.parameters.json \
  --parameters exampleArray=@arrtest.json
```

Test a template deployment

To test your template and parameter values without actually deploying any resources, use az group deployment validate.

```
az group deployment validate \
  --resource-group ExampleGroup \
  --template-file storage.json \
  --parameters @storage.parameters.json
```

If no errors are detected, the command returns information about the test deployment. In particular, notice that the **error** value is null.

```
{
  "error": null,
  "properties": {
      ...
```

If an error is detected, the command returns an error message. For example, passing an incorrect value for the storage account SKU, returns the following error:

```
{
  "error": {
    "code": "InvalidTemplate",
    "details": null,
    "message": "Deployment template validation failed: 'The provided value 'badSKU' for the template parameter
      'storageAccountType' at line '13' and column '20' is not valid. The parameter value is not part of the
allowed
      value(s): 'Standard_LRS,Standard_ZRS,Standard_GRS,Standard_RAGRS,Premium_LRS'.'.",
    "target": null
  },
  "properties": null
}
```

If your template has a syntax error, the command returns an error indicating it couldn't parse the template. The message indicates the line number and position of the parsing error.

```
{
  "error": {
    "code": "InvalidTemplate",
    "details": null,
    "message": "Deployment template parse failed: 'After parsing a value an unexpected character was
encountered:
      \". Path 'variables', line 31, position 3.'.",
    "target": null
  },
  "properties": null
}
```

Next steps

- The examples in this article deploy resources to a resource group in your default subscription. To use a different subscription, see Manage multiple Azure subscriptions.
- To specify how to handle resources that exist in the resource group but aren't defined in the template, see Azure Resource Manager deployment modes.
- To understand how to define parameters in your template, see Understand the structure and syntax of Azure Resource Manager templates.
- For tips on resolving common deployment errors, see Troubleshoot common Azure deployment errors with Azure Resource Manager.
- For information about deploying a template that requires a SAS token, see Deploy private template with SAS token.
- To safely roll out your service to more than one region, see Azure Deployment Manager.

Deploy resources with Resource Manager templates and Azure PowerShell

5/31/2019 • 9 minutes to read • Edit Online

Learn how to use Azure PowerShell with Resource Manager templates to deploy your resources to Azure. For more information about the concepts of deploying and managing your Azure solutions, see Azure Resource Manager overview.

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

Deployment scope

You can target your deployment to either an Azure subscription or a resource group within a subscription. In most cases, you'll target deployment to a resource group. Use subscription deployments to apply policies and role assignments across the subscription. You also use subscription deployments to create a resource group and deploy resources to it. Depending on the scope of the deployment, you use different commands.

To deploy to a **resource group**, use New-AzResourceGroupDeployment:

```
New-AzResourceGroupDeployment -ResourceGroupName <resource-group-name> -TemplateFile <path-to-template>
```

To deploy to a **subscription**, use New-AzDeployment:

```
New-AzDeployment -Location <location> -TemplateFile <path-to-template>
```

Currently, management group deployments are only supported through the REST API. See Deploy resources with Resource Manager templates and Resource Manager REST API.

The examples in this article use resource group deployments. For more information about subscription deployments, see Create resource groups and resources at the subscription level.

Prerequisites

You need a template to deploy. If you don't already have one, download and save an example template from the Azure Quickstart templates repo. The local file name used in this article is **c:\MyTemplates\azuredeploy.json**.

Unless you use the Azure Cloud shell to deploy templates, you need to install Azure PowerShell and connect to Azure:

- **Install Azure PowerShell cmdlets on your local computer.** For more information, see Get started with Azure PowerShell.
- **Connect to Azure by using Connect-AZAccount**. If you have multiple Azure subscriptions, you might also need to run Set-AzContext. For more information, see Use multiple Azure subscriptions.

Deploy local template

The following example creates a resource group, and deploys a template from your local machine. The name of the resource group can only include alphanumeric characters, periods, underscores, hyphens, and parenthesis. It can be up to 90 characters. It can't end in a period.

```
$resourceGroupName = Read-Host -Prompt "Enter the Resource Group name"
$location = Read-Host -Prompt "Enter the location (i.e. centralus)"

New-AzResourceGroup -Name $resourceGroupName -Location $location
New-AzResourceGroupDeployment -ResourceGroupName $resourceGroupName `
  -TemplateFile c:\MyTemplates\azuredeploy.json
```

The deployment can take a few minutes to complete.

Deploy remote template

Instead of storing Resource Manager templates on your local machine, you may prefer to store them in an external location. You can store templates in a source control repository (such as GitHub). Or, you can store them in an Azure storage account for shared access in your organization.

To deploy an external template, use the **TemplateUri** parameter. Use the URI in the example to deploy the sample template from GitHub.

```
$resourceGroupName = Read-Host -Prompt "Enter the Resource Group name"
$location = Read-Host -Prompt "Enter the location (i.e. centralus)"

New-AzResourceGroup -Name $resourceGroupName -Location $location
New-AzResourceGroupDeployment -ResourceGroupName $resourceGroupName `
  -TemplateUri https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-storage-account-create/azuredeploy.json
```

The preceding example requires a publicly accessible URI for the template, which works for most scenarios because your template shouldn't include sensitive data. If you need to specify sensitive data (like an admin password), pass that value as a secure parameter. However, if you don't want your template to be publicly accessible, you can protect it by storing it in a private storage container. For information about deploying a template that requires a shared access signature (SAS) token, see Deploy private template with SAS token. To go through a tutorial, see Tutorial: Integrate Azure Key Vault in Resource Manager Template deployment.

Deploy from Azure Cloud shell

You can use the Azure Cloud Shell to deploy your template. To deploy an external template, provide the URI of the template. To deploy a local template, you must first load your template into the storage account for your Cloud Shell. To upload files to the shell, select the **Upload/Download files** menu icon from the shell window.

To open the Cloud shell, browse to https://shell.azure.com, or select **Try-It** from the following code section:

```
$resourceGroupName = Read-Host -Prompt "Enter the Resource Group name"
$location = Read-Host -Prompt "Enter the location (i.e. centralus)"

New-AzResourceGroup -Name $resourceGroupName -Location $location
New-AzResourceGroupDeployment -ResourceGroupName $resourceGroupName `
  -TemplateUri https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-storage-account-create/azuredeploy.json
```

To paste the code into the shell, right-click inside the shell and then select **Paste**.

Redeploy when deployment fails

This feature is also known as *Rollback on error*. When a deployment fails, you can automatically redeploy an earlier, successful deployment from your deployment history. To specify redeployment, use either the `-RollbackToLastDeployment` or `-RollBackDeploymentName` parameter in the deployment command. This functionality is useful if you've got a known good state for your infrastructure deployment and want to revert to this state. There are a number of caveats and restrictions:

- The redeployment is run exactly as it was run previously with the same parameters. You can't change the parameters.
- The previous deployment is run using the complete mode. Any resources not included in the previous deployment are deleted, and any resource configurations are set to their previous state. Make sure you fully understand the deployment modes.
- The redeployment only affects the resources, any data changes aren't affected.
- This feature is only supported on Resource Group deployments, not subscription level deployments. For more information about subscription level deployment, see Create resource groups and resources at the subscription level.

To use this option, your deployments must have unique names so they can be identified in the history. If you don't have unique names, the current failed deployment might overwrite the previously successful deployment in the history. You can only use this option with root level deployments. Deployments from a nested template aren't available for redeployment.

To redeploy the last successful deployment, add the `-RollbackToLastDeployment` parameter as a flag.

```
New-AzResourceGroupDeployment -Name ExampleDeployment02 `
  -ResourceGroupName $resourceGroupName `
  -TemplateFile c:\MyTemplates\azuredeploy.json `
  -RollbackToLastDeployment
```

To redeploy a specific deployment, use the `-RollBackDeploymentName` parameter and provide the name of the deployment.

```
New-AzResourceGroupDeployment -Name ExampleDeployment02 `
  -ResourceGroupName $resourceGroupName `
  -TemplateFile c:\MyTemplates\azuredeploy.json `
  -RollBackDeploymentName ExampleDeployment01
```

The specified deployment must have succeeded.

Pass parameter values

To pass parameter values, you can use either inline parameters or a parameter file. The preceding examples in this article show inline parameters.

Inline parameters

To pass inline parameters, provide the names of the parameter with the `New-AzResourceGroupDeployment` command. For example, to pass a string and array to a template, use:

```
$arrayParam = "value1", "value2"
New-AzResourceGroupDeployment -ResourceGroupName testgroup `
  -TemplateFile c:\MyTemplates\demotemplate.json `
  -exampleString "inline string" `
  -exampleArray $arrayParam
```

You can also get the contents of file and provide that content as an inline parameter.

```
$arrayParam = "value1", "value2"
New-AzResourceGroupDeployment -ResourceGroupName testgroup `
  -TemplateFile c:\MyTemplates\demotemplate.json `
  -exampleString $(Get-Content -Path c:\MyTemplates\stringcontent.txt -Raw) `
  -exampleArray $arrayParam
```

Getting a parameter value from a file is helpful when you need to provide configuration values. For example, you can provide cloud-init values for a Linux virtual machine.

If you need to pass in an array of objects, create hash tables in PowerShell and add them to an array. Pass that array as a parameter during deployment.

```
$hash1 = @{ Name = "firstSubnet"; AddressPrefix = "10.0.0.0/24"}
$hash2 = @{ Name = "secondSubnet"; AddressPrefix = "10.0.1.0/24"}
$subnetArray = $hash1, $hash2
New-AzResourceGroupDeployment -ResourceGroupName testgroup `
  -TemplateFile c:\MyTemplates\demotemplate.json `
  -exampleArray $subnetArray
```

Parameter files

Rather than passing parameters as inline values in your script, you may find it easier to use a JSON file that contains the parameter values. The parameter file can be a local file or an external file with an accessible URI.

The parameter file must be in the following format:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
     "storageAccountType": {
         "value": "Standard_GRS"
     }
  }
}
```

Notice that the parameters section includes a parameter name that matches the parameter defined in your template (storageAccountType). The parameter file contains a value for the parameter. This value is automatically passed to the template during deployment. You can create more than one parameter file, and then pass in the appropriate parameter file for the scenario.

Copy the preceding example and save it as a file named `storage.parameters.json`.

To pass a local parameter file, use the **TemplateParameterFile** parameter:

```
New-AzResourceGroupDeployment -Name ExampleDeployment -ResourceGroupName ExampleResourceGroup `
  -TemplateFile c:\MyTemplates\azuredeploy.json `
  -TemplateParameterFile c:\MyTemplates\storage.parameters.json
```

To pass an external parameter file, use the **TemplateParameterUri** parameter:

```
New-AzResourceGroupDeployment -Name ExampleDeployment -ResourceGroupName ExampleResourceGroup `
  -TemplateUri https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-storage-
account-create/azuredeploy.json `
  -TemplateParameterUri https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-
storage-account-create/azuredeploy.parameters.json
```

Parameter precedence

You can use inline parameters and a local parameter file in the same deployment operation. For example, you can specify some values in the local parameter file and add other values inline during deployment. If you provide values for a parameter in both the local parameter file and inline, the inline value takes precedence.

However, when you use an external parameter file, you can't pass other values either inline or from a local file. When you specify a parameter file in the **TemplateParameterUri** parameter, all inline parameters are ignored. Provide all parameter values in the external file. If your template includes a sensitive value that you can't include in the parameter file, either add that value to a key vault, or dynamically provide all parameter values inline.

Parameter name conflicts

If your template includes a parameter with the same name as one of the parameters in the PowerShell command, PowerShell presents the parameter from your template with the postfix **FromTemplate**. For example, a parameter named **ResourceGroupName** in your template conflicts with the **ResourceGroupName** parameter in the New-AzResourceGroupDeployment cmdlet. You're prompted to provide a value for **ResourceGroupNameFromTemplate**. In general, you should avoid this confusion by not naming parameters with the same name as parameters used for deployment operations.

Test template deployments

To test your template and parameter values without actually deploying any resources, use Test-AzureRm ResourceGroupDeployment.

```
Test-AzResourceGroupDeployment -ResourceGroupName ExampleResourceGroup `
  -TemplateFile c:\MyTemplates\azuredeploy.json -storageAccountType Standard_GRS
```

If no errors are detected, the command finishes without a response. If an error is detected, the command returns an error message. For example, passing an incorrect value for the storage account SKU, returns the following error:

```
Test-AzResourceGroupDeployment -ResourceGroupName testgroup `
  -TemplateFile c:\MyTemplates\azuredeploy.json -storageAccountType badSku

Code    : InvalidTemplate
Message : Deployment template validation failed: 'The provided value 'badSku' for the template parameter
'storageAccountType'
          at line '15' and column '24' is not valid. The parameter value is not part of the allowed
value(s):
          'Standard_LRS,Standard_ZRS,Standard_GRS,Standard_RAGRS,Premium_LRS'.'.
Details :
```

If your template has a syntax error, the command returns an error indicating it couldn't parse the template. The message indicates the line number and position of the parsing error.

```
Test-AzResourceGroupDeployment : After parsing a value an unexpected character was encountered:
  ". Path 'variables', line 31, position 3.
```

Next steps

- To safely roll out your service to more than one region, see Azure Deployment Manager.
- To specify how to handle resources that exist in the resource group but aren't defined in the template, see Azure Resource Manager deployment modes.
- To understand how to define parameters in your template, see Understand the structure and syntax of Azure Resource Manager templates.
- For information about deploying a template that requires a SAS token, see Deploy private template with SAS token.

Deploy resources with Resource Manager templates and Resource Manager REST API

6/18/2019 • 5 minutes to read • Edit Online

This article explains how to use the Resource Manager REST API with Resource Manager templates to deploy your resources to Azure.

You can either include your template in the request body or link to a file. When using a file, it can be a local file or an external file that is available through a URI. When your template is in a storage account, you can restrict access to the template and provide a shared access signature (SAS) token during deployment.

Deployment scope

You can target your deployment to a management group, an Azure subscription, or a resource group. In most cases, you'll target deployments to a resource group. Use management group or subscription deployments to apply policies and role assignments across the specified scope. You also use subscription deployments to create a resource group and deploy resources to it. Depending on the scope of the deployment, you use different commands.

To deploy to a **resource group**, use Deployments - Create. The request is sent to:

```
PUT
https://management.azure.com/subscriptions/{subscriptionId}/resourcegroups/{resourceGroupName}/providers/Micro
soft.Resources/deployments/{deploymentName}?api-version=2019-05-01
```

To deploy to a **subscription**, use Deployments - Create At Subscription Scope. The request is sent to:

```
PUT
https://management.azure.com/subscriptions/{subscriptionId}/providers/Microsoft.Resources/deployments/{deploym
entName}?api-version=2019-05-01
```

To deploy to a **management group**, use Deployments - Create At Management Group Scope. The request is sent to:

```
PUT
https://management.azure.com/providers/Microsoft.Management/managementGroups/{groupId}/providers/Microsoft.Res
ources/deployments/{deploymentName}?api-version=2019-05-01
```

The examples in this article use resource group deployments. For more information about subscription deployments, see Create resource groups and resources at the subscription level.

Deploy with the REST API

1. Set common parameters and headers, including authentication tokens.

2. If you don't have an existing resource group, create a resource group. Provide your subscription ID, the name of the new resource group, and location that you need for your solution. For more information, see Create a resource group.

```
PUT
https://management.azure.com/subscriptions/<YourSubscriptionId>/resourcegroups/<YourResourceGroupName>?
api-version=2019-05-01
```

With a request body like:

```
{
 "location": "West US",
 "tags": {
   "tagname1": "tagvalue1"
 }
}
```

3. Validate your deployment before executing it by running the Validate a template deployment operation. When testing the deployment, provide parameters exactly as you would when executing the deployment (shown in the next step).

4. To deploy a template, provide your subscription ID, the name of the resource group, the name of the deployment in the request URI.

```
PUT
https://management.azure.com/subscriptions/<YourSubscriptionId>/resourcegroups/<YourResourceGroupName>/
providers/Microsoft.Resources/deployments/<YourDeploymentName>?api-version=2019-05-01
```

In the request body, provide a link to your template and parameter file. Notice the **mode** is set to **Incremental**. To run a complete deployment, set **mode** to **Complete**. Be careful when using the complete mode as you can inadvertently delete resources that aren't in your template.

```
{
 "properties": {
   "templateLink": {
     "uri": "http://mystorageaccount.blob.core.windows.net/templates/template.json",
     "contentVersion": "1.0.0.0"
   },
   "parametersLink": {
     "uri": "http://mystorageaccount.blob.core.windows.net/templates/parameters.json",
     "contentVersion": "1.0.0.0"
   },
   "mode": "Incremental"
 }
}
```

If you want to log response content, request content, or both, include **debugSetting** in the request.

```
{
 "properties": {
   "templateLink": {
     "uri": "http://mystorageaccount.blob.core.windows.net/templates/template.json",
     "contentVersion": "1.0.0.0"
   },
   "parametersLink": {
     "uri": "http://mystorageaccount.blob.core.windows.net/templates/parameters.json",
     "contentVersion": "1.0.0.0"
   },
   "mode": "Incremental",
   "debugSetting": {
     "detailLevel": "requestContent, responseContent"
   }
 }
}
```

You can set up your storage account to use a shared access signature (SAS) token. For more information, see Delegating Access with a Shared Access Signature.

5. Instead of linking to files for the template and parameters, you can include them in the request body. The following example shows the request body with the template and parameter inline:

```
{
   "properties": {
   "mode": "Incremental",
   "template": {
     "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
     "contentVersion": "1.0.0.0",
     "parameters": {
       "storageAccountType": {
         "type": "string",
         "defaultValue": "Standard_LRS",
         "allowedValues": [
           "Standard_LRS",
           "Standard_GRS",
           "Standard_ZRS",
           "Premium_LRS"
         ],
         "metadata": {
           "description": "Storage Account type"
         }
       },
       "location": {
         "type": "string",
         "defaultValue": "[resourceGroup().location]",
         "metadata": {
           "description": "Location for all resources."
         }
       }
     },
     "variables": {
       "storageAccountName": "[concat(uniquestring(resourceGroup().id), 'standardsa')]"
     },
     "resources": [
       {
         "type": "Microsoft.Storage/storageAccounts",
         "name": "[variables('storageAccountName')]",
         "apiVersion": "2018-02-01",
         "location": "[parameters('location')]",
         "sku": {
           "name": "[parameters('storageAccountType')]"
         },
         "kind": "StorageV2",
         "properties": {}
       }
     ],
     "outputs": {
       "storageAccountName": {
         "type": "string",
         "value": "[variables('storageAccountName')]"
       }
     }
   },
   "parameters": {
     "location": {
       "value": "eastus2"
     }
   }
 }
}
```

6. To get the status of the template deployment, use Deployments - Get.

```
GET
https://management.azure.com/subscriptions/<YourSubscriptionId>/resourcegroups/<YourResourceGroupName>/
providers/Microsoft.Resources/deployments/<YourDeploymentName>?api-version=2018-05-01
```

Redeploy when deployment fails

This feature is also known as *Rollback on error*. When a deployment fails, you can automatically redeploy an earlier, successful deployment from your deployment history. To specify redeployment, use the `onErrorDeployment` property in the request body. This functionality is useful if you've got a known good state for your infrastructure deployment and want to revert to this state. There are a number of caveats and restrictions:

- The redeployment is run exactly as it was run previously with the same parameters. You cannot change the parameters.
- The previous deployment is run using the complete mode. Any resources not included in the previous deployment are deleted, and any resource configurations are set to their previous state. Make sure you fully understand the deployment modes.
- The redeployment only affects the resources, any data changes aren't affected.
- This feature is only supported on Resource Group deployments, not subscription level deployments. For more information about subscription level deployment, see Create resource groups and resources at the subscription level.

To use this option, your deployments must have unique names so they can be identified in the history. If you don't have unique names, the current failed deployment might overwrite the previously successful deployment in the history. You can only use this option with root level deployments. Deployments from a nested template aren't available for redeployment.

To redeploy the last successful deployment if the current deployment fails, use:

```
{
  "properties": {
    "templateLink": {
      "uri": "http://mystorageaccount.blob.core.windows.net/templates/template.json",
      "contentVersion": "1.0.0.0"
    },
    "mode": "Incremental",
    "parametersLink": {
      "uri": "http://mystorageaccount.blob.core.windows.net/templates/parameters.json",
      "contentVersion": "1.0.0.0"
    },
    "onErrorDeployment": {
      "type": "LastSuccessful",
    }
  }
}
```

To redeploy a specific deployment if the current deployment fails, use:

```
{
  "properties": {
    "templateLink": {
      "uri": "http://mystorageaccount.blob.core.windows.net/templates/template.json",
      "contentVersion": "1.0.0.0"
    },
    "mode": "Incremental",
    "parametersLink": {
      "uri": "http://mystorageaccount.blob.core.windows.net/templates/parameters.json",
      "contentVersion": "1.0.0.0"
    },
    "onErrorDeployment": {
      "type": "SpecificDeployment",
      "deploymentName": "<deploymentname>"
    }
  }
}
```

The specified deployment must have succeeded.

Parameter file

If you use a parameter file to pass parameter values during deployment, you need to create a JSON file with a format similar to the following example:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "webSiteName": {
            "value": "ExampleSite"
        },
        "webSiteHostingPlanName": {
            "value": "DefaultPlan"
        },
        "webSiteLocation": {
            "value": "West US"
        },
        "adminPassword": {
            "reference": {
               "keyVault": {
                  "id": "/subscriptions/{guid}/resourceGroups/{group-
name}/providers/Microsoft.KeyVault/vaults/{vault-name}"
               },
               "secretName": "sqlAdminPassword"
            }
        }
   }
}
```

The size of the parameter file can't be more than 64 KB.

If you need to provide a sensitive value for a parameter (such as a password), add that value to a key vault. Retrieve the key vault during deployment as shown in the previous example. For more information, see Pass secure values during deployment.

Next steps

- To specify how to handle resources that exist in the resource group but aren't defined in the template, see Azure Resource Manager deployment modes.
- To learn about handling asynchronous REST operations, see Track asynchronous Azure operations.

- To learn more about templates, see Understand the structure and syntax of Azure Resource Manager templates.

Deploy private Resource Manager template with SAS token and Azure CLI

6/18/2019 • 2 minutes to read • Edit Online

When your template resides in a storage account, you can restrict access to the template and provide a shared access signature (SAS) token during deployment. This topic explains how to use Azure PowerShell with Resource Manager templates to provide a SAS token during deployment.

Add private template to storage account

You can add your templates to a storage account and link to them during deployment with a SAS token.

IMPORTANT

By following the steps below, the blob containing the template is accessible to only the account owner. However, when you create a SAS token for the blob, the blob is accessible to anyone with that URI. If another user intercepts the URI, that user is able to access the template. Using a SAS token is a good way of limiting access to your templates, but you should not include sensitive data like passwords directly in the template.

The following example sets up a private storage account container and uploads a template:

```
az group create --name "ManageGroup" --location "South Central US"
az storage account create \
    --resource-group ManageGroup \
    --location "South Central US" \
    --sku Standard_LRS \
    --kind Storage \
    --name {your-unique-name}
connection=$(az storage account show-connection-string \
    --resource-group ManageGroup \
    --name {your-unique-name} \
    --query connectionString)
az storage container create \
    --name templates \
    --public-access Off \
    --connection-string $connection
az storage blob upload \
    --container-name templates \
    --file vmlinux.json \
    --name vmlinux.json \
    --connection-string $connection
```

Provide SAS token during deployment

To deploy a private template in a storage account, generate a SAS token and include it in the URI for the template. Set the expiry time to allow enough time to complete the deployment.

```
expiretime=$(date -u -d '30 minutes' +%Y-%m-%dT%H:%MZ)
connection=$(az storage account show-connection-string \
    --resource-group ManageGroup \
    --name {your-unique-name} \
    --query connectionString)
token=$(az storage blob generate-sas \
    --container-name templates \
    --name vmlinux.json \
    --expiry $expiretime \
    --permissions r \
    --output tsv \
    --connection-string $connection)
url=$(az storage blob url \
    --container-name templates \
    --name vmlinux.json \
    --output tsv \
    --connection-string $connection)
az group deployment create --resource-group ExampleGroup --template-uri $url?$token
```

For an example of using a SAS token with linked templates, see Using linked templates with Azure Resource Manager.

Next steps

- For an introduction to deploying templates, see Deploy resources with Resource Manager templates and Azure PowerShell.
- For a complete sample script that deploys a template, see Deploy Resource Manager template script
- To define parameters in template, see Authoring templates.

Deploy private Resource Manager template with SAS token and Azure PowerShell

2/14/2019 • 2 minutes to read • Edit Online

When your template resides in a storage account, you can restrict access to the template and provide a shared access signature (SAS) token during deployment. This topic explains how to use Azure PowerShell with Resource Manager templates to provide a SAS token during deployment.

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

Add private template to storage account

You can add your templates to a storage account and link to them during deployment with a SAS token.

IMPORTANT

By following the steps below, the blob containing the template is accessible to only the account owner. However, when you create a SAS token for the blob, the blob is accessible to anyone with that URI. If another user intercepts the URI, that user is able to access the template. Using a SAS token is a good way of limiting access to your templates, but you should not include sensitive data like passwords directly in the template.

The following example sets up a private storage account container and uploads a template:

```
# create a storage account for templates
New-AzResourceGroup -Name ManageGroup -Location "South Central US"
New-AzStorageAccount -ResourceGroupName ManageGroup -Name {your-unique-name} -Type Standard_LRS -Location
"West US"
Set-AzCurrentStorageAccount -ResourceGroupName ManageGroup -Name {your-unique-name}

# create a container and upload template
New-AzStorageContainer -Name templates -Permission Off
Set-AzStorageBlobContent -Container templates -File c:\MyTemplates\storage.json
```

Provide SAS token during deployment

To deploy a private template in a storage account, generate a SAS token and include it in the URI for the template. Set the expiry time to allow enough time to complete the deployment.

```
Set-AzCurrentStorageAccount -ResourceGroupName ManageGroup -Name {your-unique-name}

# get the URI with the SAS token
$templateuri = New-AzStorageBlobSASToken -Container templates -Blob storage.json -Permission r `
  -ExpiryTime (Get-Date).AddHours(2.0) -FullUri

# provide URI with SAS token during deployment
New-AzResourceGroup -Name ExampleGroup -Location "South Central US"
New-AzResourceGroupDeployment -ResourceGroupName ExampleGroup -TemplateUri $templateuri
```

For an example of using a SAS token with linked templates, see Using linked templates with Azure Resource Manager.

Next steps

- For an introduction to deploying templates, see Deploy resources with Resource Manager templates and Azure PowerShell.
- For a complete sample script that deploys a template, see Deploy Resource Manager template script
- To define parameters in template, see Authoring templates.

Deploy Azure resources to more than one subscription or resource group

6/18/2019 • 6 minutes to read • Edit Online

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

Typically, you deploy all the resources in your template to a single resource group. However, there are scenarios where you want to deploy a set of resources together but place them in different resource groups or subscriptions. For example, you may want to deploy the backup virtual machine for Azure Site Recovery to a separate resource group and location. Resource Manager enables you to use nested templates to target different subscriptions and resource groups than the subscription and resource group used for the parent template.

NOTE

You can deploy to only five resource groups in a single deployment. Typically, this limitation means you can deploy to one resource group specified for the parent template, and up to four resource groups in nested or linked deployments. However, if your parent template contains only nested or linked templates and does not itself deploy any resources, then you can include up to five resource groups in nested or linked deployments.

Specify a subscription and resource group

To target a different resource, use a nested or linked template. The `Microsoft.Resources/deployments` resource type provides parameters for `subscriptionId` and `resourceGroup`. These properties enable you to specify a different subscription and resource group for the nested deployment. All the resource groups must exist before running the deployment. If you do not specify either the subscription ID or resource group, the subscription and resource group from the parent template is used.

The account you use to deploy the template must have permissions to deploy to the specified subscription ID. If the specified subscription exists in a different Azure Active Directory tenant, you must add guest users from another directory.

To specify a different resource group and subscription, use:

```
"resources": [
    {
        "apiVersion": "2017-05-10",
        "name": "nestedTemplate",
        "type": "Microsoft.Resources/deployments",
        "resourceGroup": "[parameters('secondResourceGroup')]",
        "subscriptionId": "[parameters('secondSubscriptionID')]",
        ...
    }
]
```

If your resource groups are in the same subscription, you can remove the **subscriptionId** value.

The following example deploys two storage accounts - one in the resource group specified during deployment, and one in a resource group specified in the `secondResourceGroup` parameter:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "storagePrefix": {
            "type": "string",
            "maxLength": 11
        },
        "secondResourceGroup": {
            "type": "string"
        },
        "secondSubscriptionID": {
            "type": "string",
            "defaultValue": ""
        },
        "secondStorageLocation": {
            "type": "string",
            "defaultValue": "[resourceGroup().location]"
        }
    },
    "variables": {
        "firstStorageName": "[concat(parameters('storagePrefix'), uniqueString(resourceGroup().id))]",
        "secondStorageName": "[concat(parameters('storagePrefix'),
uniqueString(parameters('secondSubscriptionID'), parameters('secondResourceGroup')))]"
    },
    "resources": [
        {
            "apiVersion": "2017-05-10",
            "name": "nestedTemplate",
            "type": "Microsoft.Resources/deployments",
            "resourceGroup": "[parameters('secondResourceGroup')]",
            "subscriptionId": "[parameters('secondSubscriptionID')]",
            "properties": {
                "mode": "Incremental",
                "template": {
                    "$schema": "https://schema.management.azure.com/schemas/2015-01-
01/deploymentTemplate.json#",
                    "contentVersion": "1.0.0.0",
                    "parameters": {},
                    "variables": {},
                    "resources": [
                        {
                            "type": "Microsoft.Storage/storageAccounts",
                            "name": "[variables('secondStorageName')]",
                            "apiVersion": "2017-06-01",
                            "location": "[parameters('secondStorageLocation')]",
                            "sku":{
                                "name": "Standard_LRS"
                            },
                            "kind": "Storage",
                            "properties": {
                            }
                        }
                    ]
                },
                "parameters": {}
            }
        },
        {
            "type": "Microsoft.Storage/storageAccounts",
            "name": "[variables('firstStorageName')]",
            "apiVersion": "2017-06-01",
            "location": "[resourceGroup().location]",
            "sku":{
                "name": "Standard_LRS"
                "name": "Standard_LRS"
            },
            "kind": "Storage",
            "properties": {
            }
        }
    ]
}
```

If you set `resourceGroup` to the name of a resource group that does not exist, the deployment fails.

Use the resourceGroup() and subscription() functions

For cross resource group deployments, the resourceGroup() and subscription() functions resolve differently based on how you specify the nested template.

If you embed one template within another template, the functions in the nested template resolve to the parent resource group and subscription. An embedded template uses the following format:

```
"apiVersion": "2017-05-10",
"name": "embeddedTemplate",
"type": "Microsoft.Resources/deployments",
"resourceGroup": "crossResourceGroupDeployment",
"properties": {
    "mode": "Incremental",
    "template": {
        ...
        resourceGroup() and subscription() refer to parent resource group/subscription
    }
}
```

If you link to a separate template, the functions in the linked template resolve to the nested resource group and subscription. A linked template uses the following format:

```
"apiVersion": "2017-05-10",
"name": "linkedTemplate",
"type": "Microsoft.Resources/deployments",
"resourceGroup": "crossResourceGroupDeployment",
"properties": {
    "mode": "Incremental",
    "templateLink": {
        ...
        resourceGroup() and subscription() in linked template refer to linked resource group/subscription
    }
}
```

Example templates

The following templates demonstrate multiple resource group deployments. Scripts to deploy the templates are shown after the table.

TEMPLATE	DESCRIPTION
Cross subscription template	Deploys one storage account to one resource group and one storage account to a second resource group. Include a value for the subscription ID when the second resource group is in a different subscription.

TEMPLATE	DESCRIPTION
Cross resource group properties template	Demonstrates how the `resourceGroup()` function resolves. It does not deploy any resources.

PowerShell

For PowerShell, to deploy two storage accounts to two resource groups in the **same subscription**, use:

```
$firstRG = "primarygroup"
$secondRG = "secondarygroup"

New-AzResourceGroup -Name $firstRG -Location southcentralus
New-AzResourceGroup -Name $secondRG -Location eastus

New-AzResourceGroupDeployment `
  -ResourceGroupName $firstRG `
  -TemplateUri https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/crosssubscription.json `
  -storagePrefix storage `
  -secondResourceGroup $secondRG `
  -secondStorageLocation eastus
```

For PowerShell, to deploy two storage accounts to **two subscriptions**, use:

```
$firstRG = "primarygroup"
$secondRG = "secondarygroup"

$firstSub = "<first-subscription-id>"
$secondSub = "<second-subscription-id>"

Select-AzSubscription -Subscription $secondSub
New-AzResourceGroup -Name $secondRG -Location eastus

Select-AzSubscription -Subscription $firstSub
New-AzResourceGroup -Name $firstRG -Location southcentralus

New-AzResourceGroupDeployment `
  -ResourceGroupName $firstRG `
  -TemplateUri https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/crosssubscription.json `
  -storagePrefix storage `
  -secondResourceGroup $secondRG `
  -secondStorageLocation eastus `
  -secondSubscriptionID $secondSub
```

For PowerShell, to test how the **resource group object** resolves for the parent template, inline template, and linked template, use:

```
New-AzResourceGroup -Name parentGroup -Location southcentralus
New-AzResourceGroup -Name inlineGroup -Location southcentralus
New-AzResourceGroup -Name linkedGroup -Location southcentralus

New-AzResourceGroupDeployment `
  -ResourceGroupName parentGroup `
  -TemplateUri https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/crossresourcegroupproperties.json
```

In the preceding example, both **parentRG** and **inlineRG** resolve to **parentGroup**. **linkedRG** resolves to **linkedGroup**. The output from the preceding example is:

```
 Name             Type                       Value
 ===============  =========================  ==========
 parentRG         Object                     {
                                               "id": "/subscriptions/<subscription-
id>/resourceGroups/parentGroup",
                                               "name": "parentGroup",
                                               "location": "southcentralus",
                                               "properties": {
                                                 "provisioningState": "Succeeded"
                                               }
                                             }
 inlineRG         Object                     {
                                               "id": "/subscriptions/<subscription-
id>/resourceGroups/parentGroup",
                                               "name": "parentGroup",
                                               "location": "southcentralus",
                                               "properties": {
                                                 "provisioningState": "Succeeded"
                                               }
                                             }
 linkedRG         Object                     {
                                               "id": "/subscriptions/<subscription-
id>/resourceGroups/linkedGroup",
                                               "name": "linkedGroup",
                                               "location": "southcentralus",
                                               "properties": {
                                                 "provisioningState": "Succeeded"
                                               }
                                             }
```

Azure CLI

For Azure CLI, to deploy two storage accounts to two resource groups in the **same subscription**, use:

```
firstRG="primarygroup"
secondRG="secondarygroup"

az group create --name $firstRG --location southcentralus
az group create --name $secondRG --location eastus
az group deployment create \
  --name ExampleDeployment \
  --resource-group $firstRG \
  --template-uri https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-manager/crosssubscription.json \
  --parameters storagePrefix=tfstorage secondResourceGroup=$secondRG secondStorageLocation=eastus
```

For Azure CLI, to deploy two storage accounts to **two subscriptions**, use:

```
firstRG="primarygroup"
secondRG="secondarygroup"

firstSub="<first-subscription-id>"
secondSub="<second-subscription-id>"

az account set --subscription $secondSub
az group create --name $secondRG --location eastus

az account set --subscription $firstSub
az group create --name $firstRG --location southcentralus

az group deployment create \
  --name ExampleDeployment \
  --resource-group $firstRG \
  --template-uri https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-manager/crosssubscription.json \
  --parameters storagePrefix=storage secondResourceGroup=$secondRG secondStorageLocation=eastus secondSubscriptionID=$secondSub
```

For Azure CLI, to test how the **resource group object** resolves for the parent template, inline template, and linked template, use:

```
az group create --name parentGroup --location southcentralus
az group create --name inlineGroup --location southcentralus
az group create --name linkedGroup --location southcentralus

az group deployment create \
  --name ExampleDeployment \
  --resource-group parentGroup \
  --template-uri https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-manager/crossresourcegroupproperties.json
```

In the preceding example, both **parentRG** and **inlineRG** resolve to **parentGroup**. **linkedRG** resolves to **linkedGroup**. The output from the preceding example is:

```
...
"outputs": {
  "inlineRG": {
    "type": "Object",
    "value": {
      "id": "/subscriptions/<subscription-id>/resourceGroups/parentGroup",
      "location": "southcentralus",
      "name": "parentGroup",
      "properties": {
        "provisioningState": "Succeeded"
      }
    }
  },
  "linkedRG": {
    "type": "Object",
    "value": {
      "id": "/subscriptions/<subscription-id>/resourceGroups/linkedGroup",
      "location": "southcentralus",
      "name": "linkedGroup",
      "properties": {
        "provisioningState": "Succeeded"
      }
    }
  },
  "parentRG": {
    "type": "Object",
    "value": {
      "id": "/subscriptions/<subscription-id>/resourceGroups/parentGroup",
      "location": "southcentralus",
      "name": "parentGroup",
      "properties": {
        "provisioningState": "Succeeded"
      }
    }
  }
},
...
```

Next steps

- To understand how to define parameters in your template, see Understand the structure and syntax of Azure Resource Manager templates.
- For tips on resolving common deployment errors, see Troubleshoot common Azure deployment errors with Azure Resource Manager.
- For information about deploying a template that requires a SAS token, see Deploy private template with SAS token.

Integrate Resource Manager templates with Azure Pipelines

6/18/2019 • 5 minutes to read • Edit Online

Visual Studio provides the Azure Resource Group project for creating templates and deploying them to your Azure subscription. You can integrate this project with Azure Pipelines for continuous integration and continuous deployment (CI/CD).

There are two ways to deploy templates with Azure Pipelines:

- **Add task that runs an Azure PowerShell script**. This option has the advantage of providing consistency throughout the development life cycle because you use the same script that is included in the Visual Studio project (Deploy-AzureResourceGroup.ps1). The script stages artifacts from your project to a storage account that Resource Manager can access. Artifacts are items in your project such as linked templates, scripts, and application binaries. Then, the script deploys the template.
- **Add tasks to copy and deploy tasks**. This option offers a convenient alternative to the project script. You configure two tasks in the pipeline. One task stages the artifacts and the other task deploys the template.

This article shows both approaches.

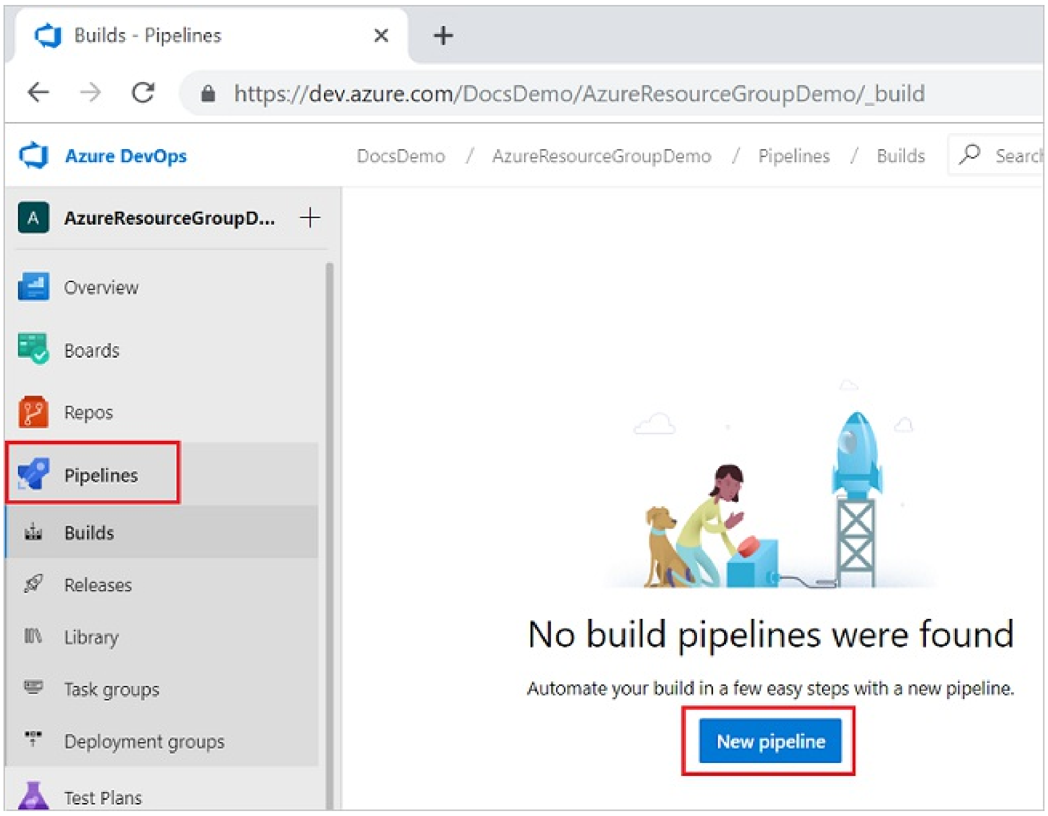
Prepare your project

This article assumes your Visual Studio project and Azure DevOps organization are ready for creating the pipeline. The following steps show how to make sure you're ready:

- You have an Azure DevOps organization. If you don't have one, create one for free. If your team already has an Azure DevOps organization, make sure you're an administrator of the Azure DevOps project that you want to use.
- You've configured a service connection to your Azure subscription. The tasks in the pipeline execute under the identity of the service principal. For steps to create the connection, see Create a DevOps project.
- You have a Visual Studio project that was created from the **Azure Resource Group** starter template. For information about creating that type of project, see Creating and deploying Azure resource groups through Visual Studio.
- Your Visual Studio project is connected to an Azure DevOps project.

Create pipeline

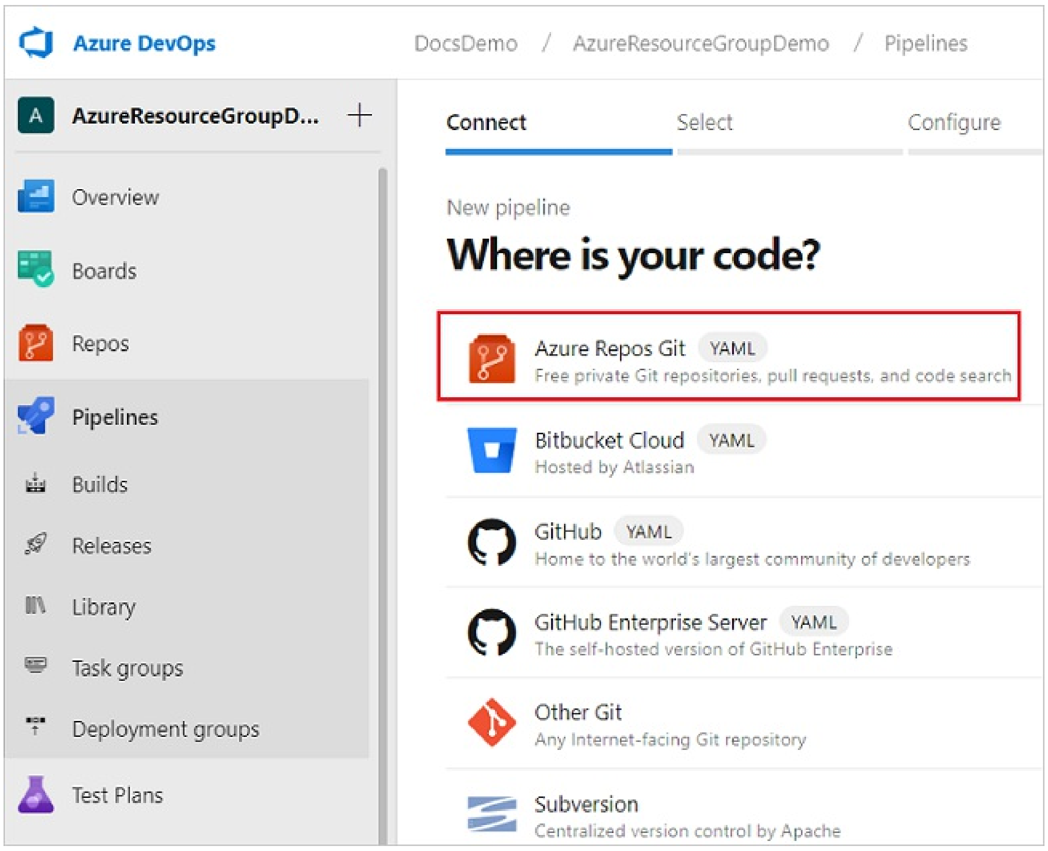
1. If you haven't added a pipeline previously, you need to create a new pipeline. From your Azure DevOps organization, select **Pipelines** and **New pipeline**.



No build pipelines were found

Automate your build in a few easy steps with a new pipeline.

2. Specify where your code is stored. The following image shows selecting **Azure Repos Git**.



Where is your code?

Azure Repos Git YAML
Free private Git repositories, pull requests, and code search

Bitbucket Cloud YAML
Hosted by Atlassian

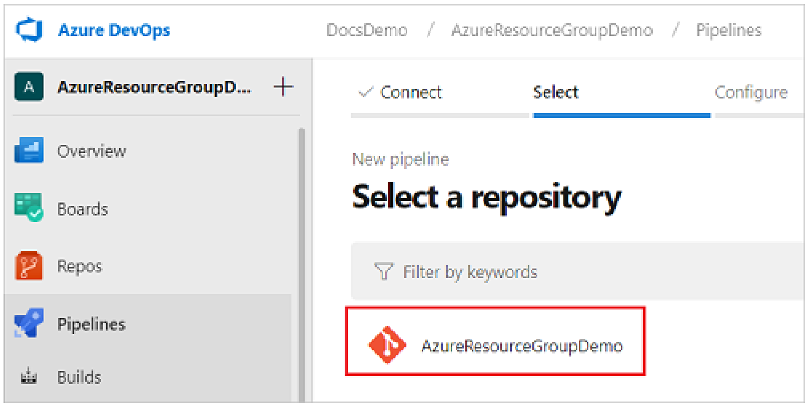
GitHub YAML
Home to the world's largest community of developers

GitHub Enterprise Server YAML
The self-hosted version of GitHub Enterprise

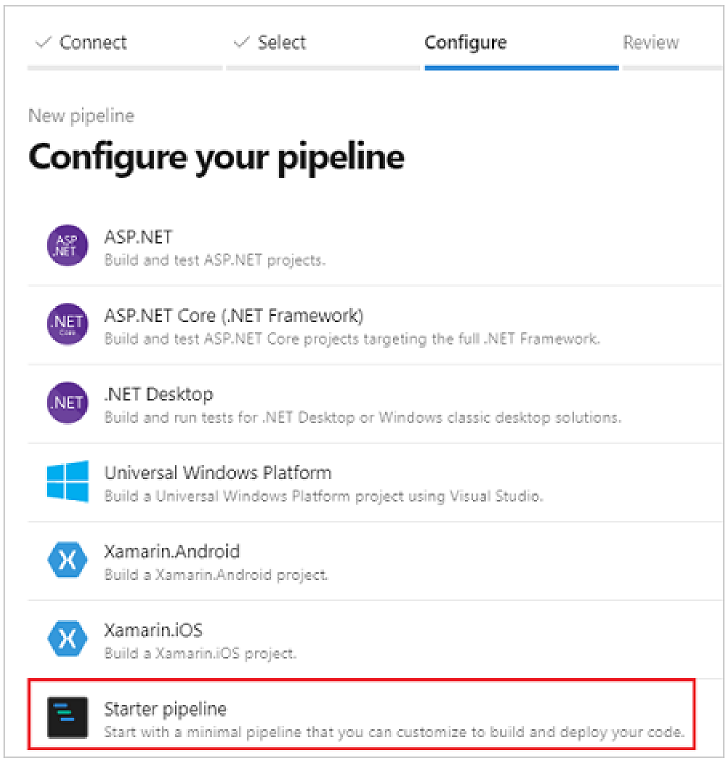
Other Git
Any Internet-facing Git repository

Subversion
Centralized version control by Apache

3. From that source, select the repository that has the code for your project.



4. Select the type of pipeline to create. You can select **Starter pipeline**.


Connect
Select
Configure
Review
New pipeline
Configure your pipeline
ASP.NET
Build and test ASP.NET projects.
ASP.NET Core (.NET Framework)
Build and test ASP.NET Core projects targeting the full .NET Framework.
.NET Desktop
Build and run tests for .NET Desktop or Windows classic desktop solutions.
Universal Windows Platform
Build a Universal Windows Platform project using Visual Studio.
Xamarin.Android
Build a Xamarin.Android project.
Xamarin.iOS
Build a Xamarin.iOS project.
Starter pipeline
Start with a minimal pipeline that you can customize to build and deploy your code.

You're ready to either add an Azure PowerShell task or the copy file and deploy tasks.

Azure PowerShell task

This section shows how to configure continuous deployment by using a single task that runs the PowerShell script in your project. The following YAML file creates an Azure PowerShell task:

```
pool:
  name: Hosted Windows 2019 with VS2019
  demands: azureps

steps:
- task: AzurePowerShell@3
  inputs:
    azureSubscription: 'demo-deploy-sp'
    ScriptPath: 'AzureResourceGroupDemo/Deploy-AzureResourceGroup.ps1'
    ScriptArguments: -ResourceGroupName 'demogroup' -ResourceGroupLocation 'centralus'
    azurePowerShellVersion: LatestVersion
```

When you set the task to `AzurePowerShell@3`, the pipeline uses commands from the AzureRM module to authenticate the connection. By default, the PowerShell script in the Visual Studio project uses the AzureRM module. If you've updated your script to use the Az module, set the task to `AzurePowerShell@4`.

```
steps:
- task: AzurePowerShell@4
```

For `azureSubscription`, provide the name of the service connection you created.

```
inputs:
    azureSubscription: '<your-connection-name>'
```

For `scriptPath`, provide the relative path from the pipeline file to your script. You can look in your repository to see the path.

```
ScriptPath: '<your-relative-path>/<script-file-name>.ps1'
```

If you don't need to stage artifacts, just pass the name and location of a resource group to use for deployment. The Visual Studio script creates the resource group if it doesn't already exist.

```
ScriptArguments: -ResourceGroupName '<resource-group-name>' -ResourceGroupLocation '<location>'
```

If you need to stage artifacts to an existing storage account, use:

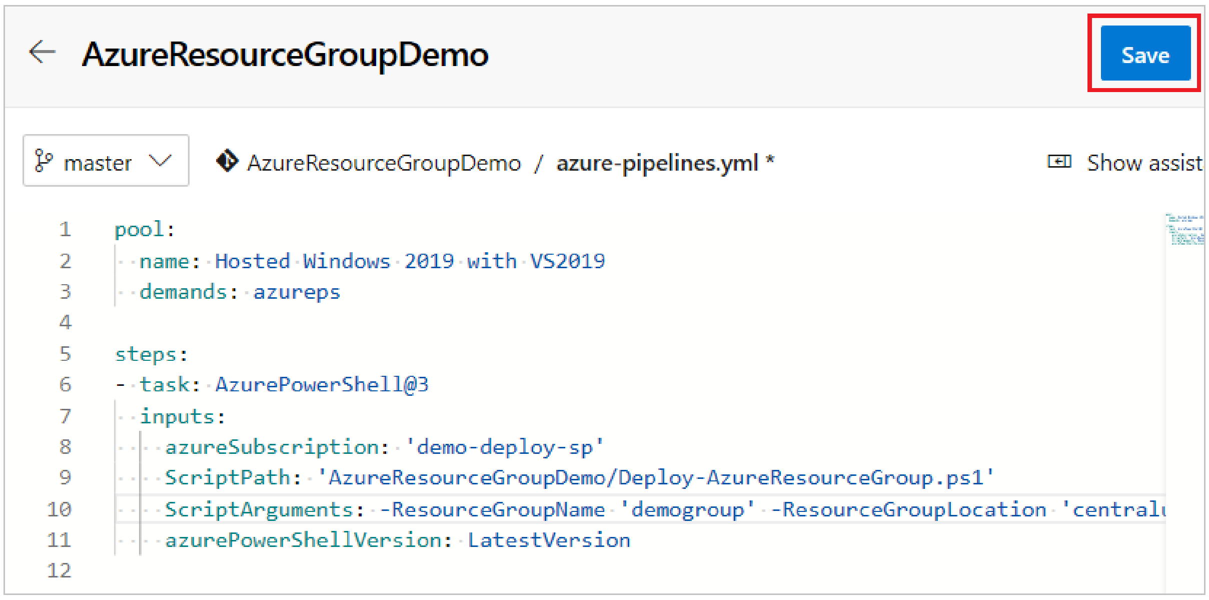
```
ScriptArguments: -ResourceGroupName '<resource-group-name>' -ResourceGroupLocation '<location>' -
UploadArtifacts -ArtifactStagingDirectory '$(Build.StagingDirectory)' -StorageAccountName '<your-storage-
account>'
```

Now, that you understand how to create the task, let's go through the steps to edit the pipeline.

1. Open your pipeline, and replace the contents with your YAML:

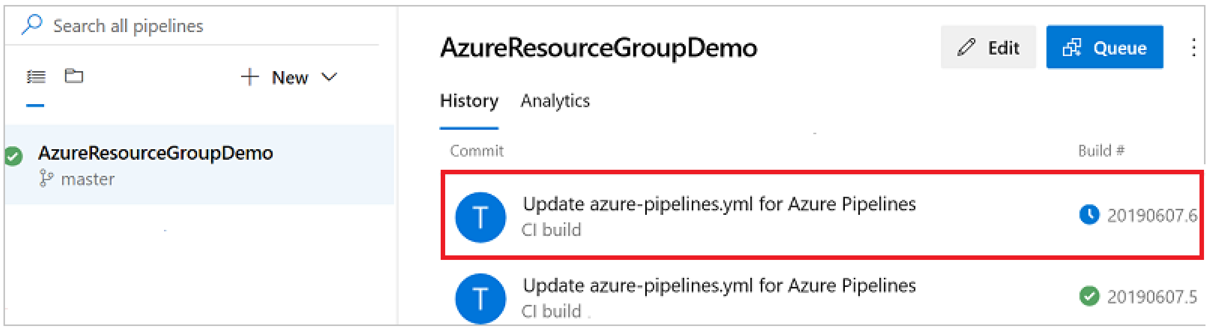
   ```
   pool:
     name: Hosted Windows 2019 with VS2019
     demands: azureps

   steps:
   - task: AzurePowerShell@3
     inputs:
       azureSubscription: 'demo-deploy-sp'
       ScriptPath: 'AzureResourceGroupDemo/Deploy-AzureResourceGroup.ps1'
       ScriptArguments: -ResourceGroupName 'demogroup' -ResourceGroupLocation 'centralus' -UploadArtifacts
   -ArtifactStagingDirectory '$(Build.StagingDirectory)' -StorageAccountName 'stage3a4176e058d34bb88cc'
       azurePowerShellVersion: LatestVersion
   ```

2. Select **Save**.



```
pool:
  name: Hosted Windows 2019 with VS2019
  demands: azureps

steps:
- task: AzurePowerShell@3
  inputs:
    azureSubscription: 'demo-deploy-sp'
    ScriptPath: 'AzureResourceGroupDemo/Deploy-AzureResourceGroup.ps1'
    ScriptArguments: -ResourceGroupName 'demogroup' -ResourceGroupLocation 'centralu
    azurePowerShellVersion: LatestVersion
```

3. Provide a message for the commit, and commit directly to **master**.
4. When you select **Save**, the build pipeline is automatically run. Go back to the summary for your build pipeline, and watch the status.



Update azure-pipelines.yml for Azure Pipelines
CI build
20190607.6

Update azure-pipelines.yml for Azure Pipelines
CI build
20190607.5

You can select the currently running pipeline to see details about the tasks. When it finishes, you see the results for each step.

Copy and deploy tasks

This section shows how to configure continuous deployment by using a two tasks to stage the artifacts and deploy the template.

The following YAML shows the Azure file copy task:

```
- task: AzureFileCopy@3
  displayName: 'Stage files'
  inputs:
    SourcePath: 'AzureResourceGroup1'
    azureSubscription: 'demo-deploy-sp'
    Destination: 'AzureBlob'
    storage: 'stage3a4176e058d34bb88cc'
    ContainerName: 'democontainer'
    outputStorageUri: 'artifactsLocation'
    outputStorageContainerSasToken: 'artifactsLocationSasToken'
    sasTokenTimeOutInMinutes: '240'
```

There are several parts of this task to revise for your environment. The `SourcePath` indicates the location of the artifacts relative to the pipeline file. In this example, the files exist in a folder named `AzureResourceGroup1` which was the name of the project.

```
SourcePath: '<path-to-artifacts>'
```

For `azureSubscription`, provide the name of the service connection you created.

```
azureSubscription: '<your-connection-name>'
```

For storage and container name, provide the names of the storage account and container you want to use for storing the artifacts. The storage account must exist.

```
storage: '<your-storage-account-name>'
ContainerName: '<container-name>'
```

The following YAML shows the Azure resource group deployment task:

```
- task: AzureResourceGroupDeployment@2
  displayName: 'Deploy template'
  inputs:
    azureSubscription: 'demo-deploy-sp'
    resourceGroupName: 'demogroup'
    location: 'centralus'
    templateLocation: 'URL of the file'
    csmFileLink: '$(artifactsLocation)WebSite.json$(artifactsLocationSasToken)'
    csmParametersFileLink: '$(artifactsLocation)WebSite.parameters.json$(artifactsLocationSasToken)'
    overrideParameters: '-_artifactsLocation $(artifactsLocation) -_artifactsLocationSasToken
"$(artifactsLocationSasToken)"'
```

There are several parts of this task to revise for your environment. For `azureSubscription`, provide the name of the service connection you created.

```
azureSubscription: '<your-connection-name>'
```

For `resourceGroupName` and `location`, provide the name and location of the resource group you want to deploy to. The task creates the resource group if it doesn't exist.

```
resourceGroupName: '<resource-group-name>'
location: '<location>'
```

The deployment task links to a template named `WebSite.json` and a parameters file named WebSite.parameters.json. Use the names of your template and parameter files.

Now, that you understand how to create the tasks, let's go through the steps to edit the pipeline.

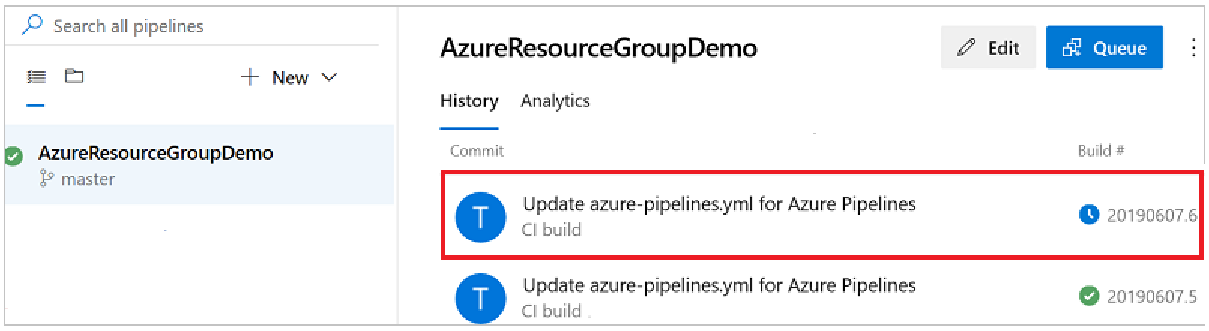
1. Open your pipeline, and replace the contents with your YAML:

```
pool:
  name: Hosted Windows 2019 with VS2019

steps:
- task: AzureFileCopy@3
  displayName: 'Stage files'
  inputs:
    SourcePath: 'AzureResourceGroup1'
    azureSubscription: 'demo-deploy-sp'
    Destination: 'AzureBlob'
    storage: 'stage3a4176e058d34bb88cc'
    ContainerName: 'democontainer'
    outputStorageUri: 'artifactsLocation'
    outputStorageContainerSasToken: 'artifactsLocationSasToken'
    sasTokenTimeOutInMinutes: '240'
- task: AzureResourceGroupDeployment@2
  displayName: 'Deploy template'
  inputs:
    azureSubscription: 'demo-deploy-sp'
    resourceGroupName: demogroup
    location: 'centralus'
    templateLocation: 'URL of the file'
    csmFileLink: '$(artifactsLocation)WebSite.json$(artifactsLocationSasToken)'
    csmParametersFileLink: '$(artifactsLocation)WebSite.parameters.json$(artifactsLocationSasToken)'
    overrideParameters: '-_artifactsLocation $(artifactsLocation) -_artifactsLocationSasToken
"$(artifactsLocationSasToken)"'
```

2. Select **Save**.

3. Provide a message for the commit, and commit directly to **master**.

4. When you select **Save**, the build pipeline is automatically run. Go back to the summary for your build pipeline, and watch the status.



Update azure-pipelines.yml for Azure Pipelines
CI build
20190607.6

Update azure-pipelines.yml for Azure Pipelines
CI build
20190607.5

You can select the currently running pipeline to see details about the tasks. When it finishes, you see the results for each step.

Next steps

For step-by-step process on using Azure Pipelines with Resource Manager templates, see Tutorial: Continuous integration of Azure Resource Manager templates with Azure Pipelines.

Single and multi-resource export to template in Azure portal

6/20/2019 • 3 minutes to read • Edit Online

To assist with creating Azure Resource Manager templates, you can export a template from existing resources. The exported template helps you understand the JSON syntax and properties that deploy your resources. To automate future deployments, start with the exported template and modify it for your scenario.

Resource Manager enables you to pick one or more resources for exporting to a template. You can focus on exactly the resources you need in the template.

This article shows how to export templates through the portal. You can also use Azure CLI, Azure PowerShell, or REST API.

Choose the right export option

There are two ways to export a template:

- **Export from resource group or resource**. This option generates a new template from existing resources. The exported template is a "snapshot" of the current state of the resource group. You can export an entire resource group or specific resources within that resource group.
- **Export before deployment or from history**. This option retrieves an exact copy of a template used for deployment.

Depending on the option you choose, the exported templates have different qualities.

FROM RESOURCE GROUP OR RESOURCE	BEFORE DEPLOYMENT OR FROM HISTORY
Template is snapshot of the resources' current state. It includes any manual changes you made after deployment.	Template only shows state of resources at the time of deployment. Any manual changes you made after deployment aren't included.
You can select which resources from a resource group to export.	All resources for a specific deployment are included. You can't pick a subset of those resources or add resources that were added at a different time.
Template includes all properties for the resources, including some properties you wouldn't normally set during deployment. You might want to remove or clean up these properties before reusing the template.	Template includes only the properties needed for the deployment. The template is ready-to-use.
Template probably doesn't include all of the parameters you need for reuse. Most property values are hard-coded in the template. To redeploy the template in other environments, you need to add parameters that increase the ability to configure the resources.	Template includes parameters that make it easy to redeploy in different environments.

Export the template from a resource group or resource, when:

- You need to capture changes to the resources that were made after the original deployment.
- You want to select which resources are exported.

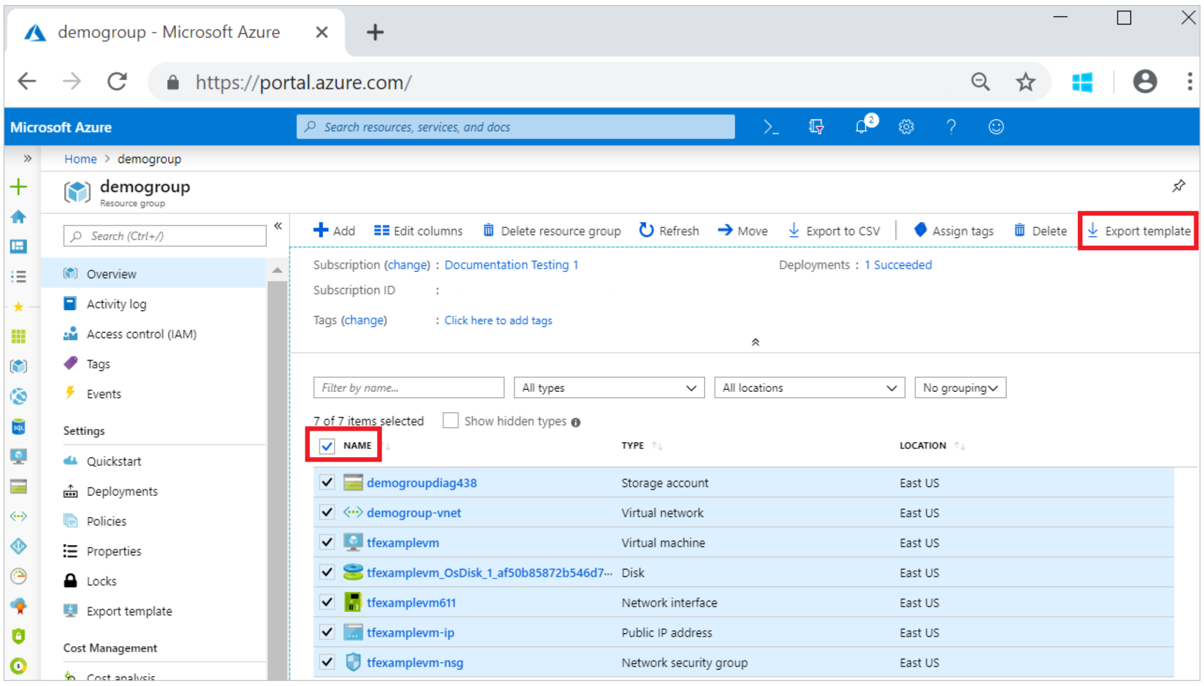
Export the template before deployment or from the history, when:

- You want an easy-to-reuse template.
- You don't need to include changes you made after the original deployment.

Export template from resource group

To export one or more resources from a resource group:

1. Select the resource group that contains the resources you want to export.
2. To export all resources in the resource group, select all and then **Export template**. The **Export template** option only becomes enabled after you've selected at least one resource.

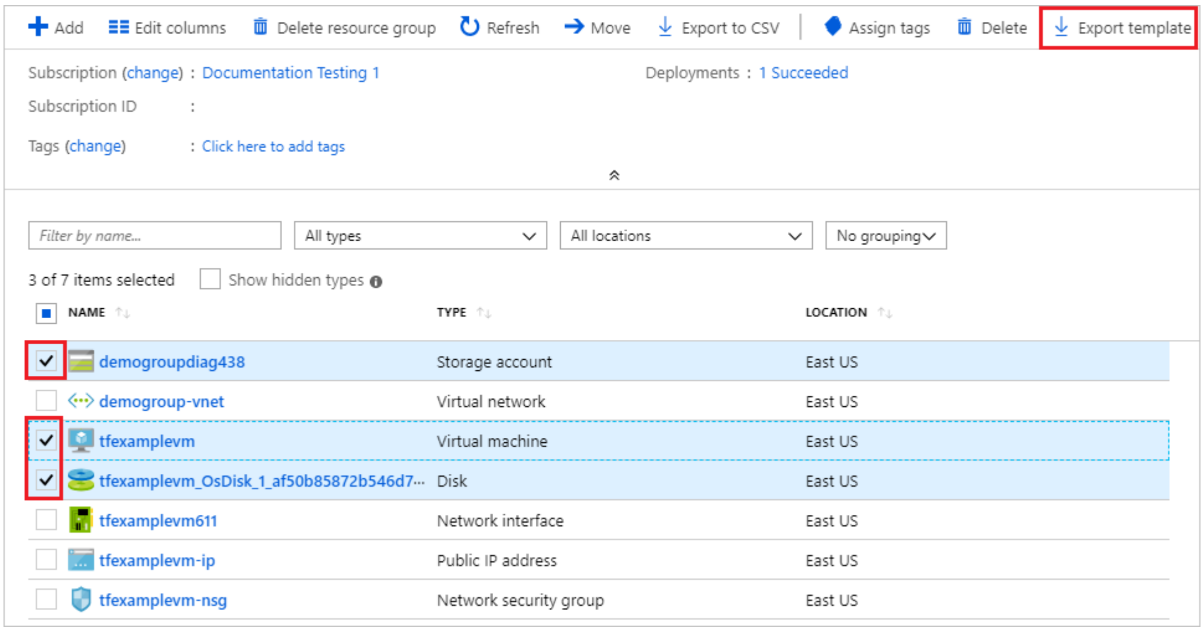


Subscription (change) : Documentation Testing 1
Deployments : 1 Succeeded
Subscription ID :
Tags (change) : Click here to add tags

7 of 7 items selected

	NAME	TYPE	LOCATION
✓	demogroupdiag438	Storage account	East US
✓	demogroup-vnet	Virtual network	East US
✓	tfexamplevm	Virtual machine	East US
✓	tfexamplevm_OsDisk_1_af50b85872b546d7...	Disk	East US
✓	tfexamplevm611	Network interface	East US
✓	tfexamplevm-ip	Public IP address	East US
✓	tfexamplevm-nsg	Network security group	East US

3. To pick specific resources for export, select the checkboxes next to those resources. Then, select **Export template**.

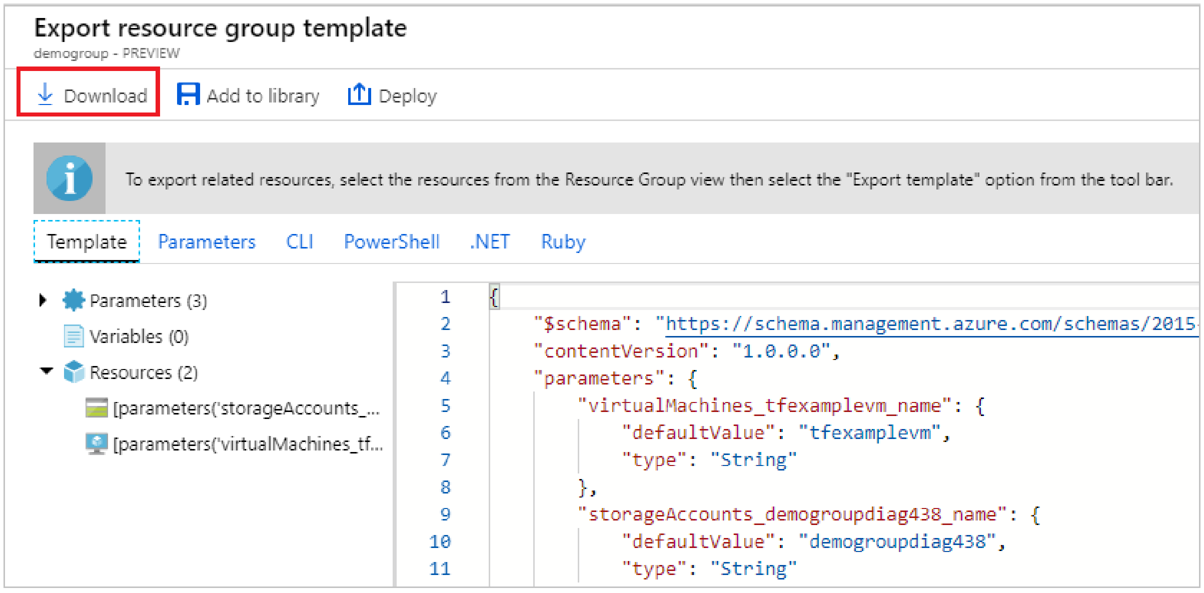


Subscription (change) : Documentation Testing 1
Deployments : 1 Succeeded
Subscription ID :
Tags (change) : Click here to add tags

3 of 7 items selected

	NAME	TYPE	LOCATION
✓	demogroupdiag438	Storage account	East US
	demogroup-vnet	Virtual network	East US
✓	tfexamplevm	Virtual machine	East US
✓	tfexamplevm_OsDisk_1_af50b85872b546d7...	Disk	East US
	tfexamplevm611	Network interface	East US
	tfexamplevm-ip	Public IP address	East US
	tfexamplevm-nsg	Network security group	East US

4. The exported template is displayed, and is available to download.


Export resource group template

demogroup - PREVIEW

Download | Add to library | Deploy

To export related resources, select the resources from the Resource Group view then select the "Export template" option from the tool bar.

Template | Parameters | CLI | PowerShell | .NET | Ruby

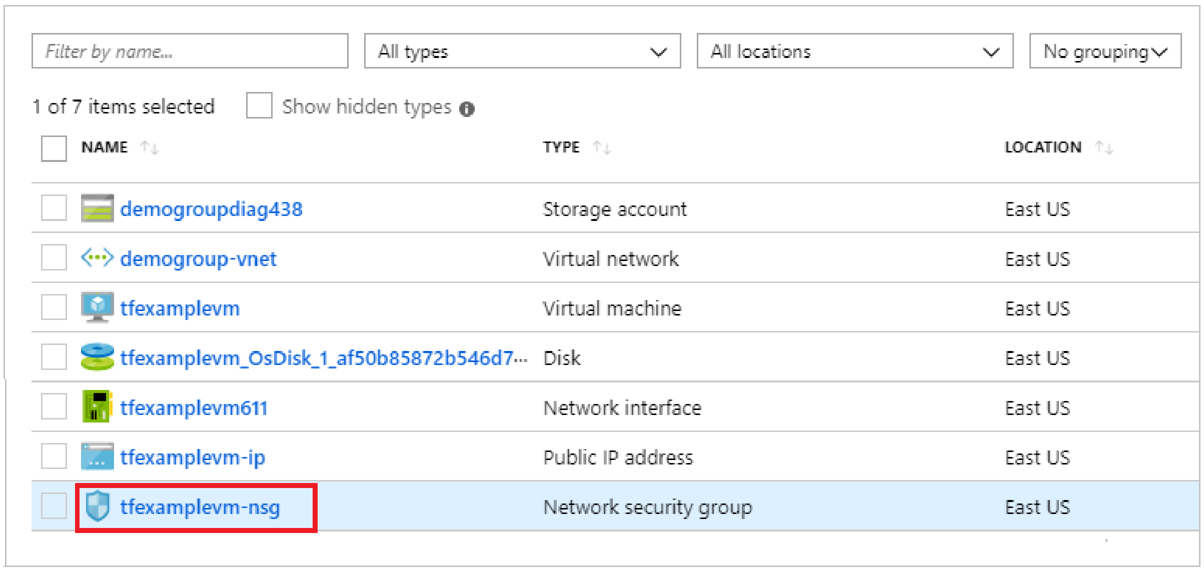
- Parameters (3)
- Variables (0)
- Resources (2)
  - [parameters('storageAccounts_...
  - [parameters('virtualMachines_tf...

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015
    "contentVersion": "1.0.0.0",
    "parameters": {
        "virtualMachines_tfexamplevm_name": {
            "defaultValue": "tfexamplevm",
            "type": "String"
        },
        "storageAccounts_demogroupdiag438_name": {
            "defaultValue": "demogroupdiag438",
            "type": "String"
```

Export template from resource

To export one resource:

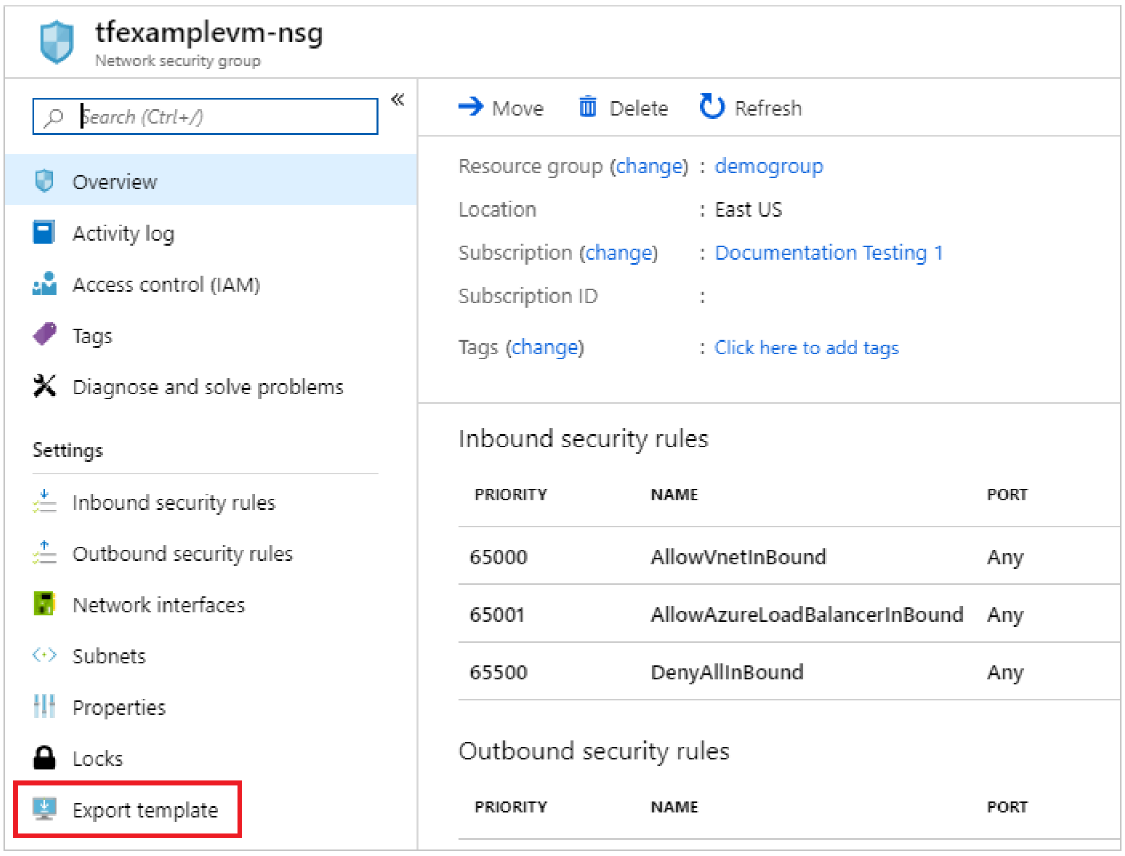
1. Select the resource group containing the resource you want to export.
2. Select the resource to export.


Filter by name... | All types | All locations | No grouping

1 of 7 items selected | Show hidden types

NAME	TYPE	LOCATION
demogroupdiag438	Storage account	East US
demogroup-vnet	Virtual network	East US
tfexamplevm	Virtual machine	East US
tfexamplevm_OsDisk_1_af50b85872b546d7...	Disk	East US
tfexamplevm611	Network interface	East US
tfexamplevm-ip	Public IP address	East US
tfexamplevm-nsg	Network security group	East US

3. For that resource, select **Export template** in the left pane.


tfexamplevm-nsg
Network security group

Resource group (change) : demogroup
Location : East US
Subscription (change) : Documentation Testing 1
Subscription ID :
Tags (change) : Click here to add tags

Inbound security rules

PRIORITY	NAME	PORT
65000	AllowVnetInBound	Any
65001	AllowAzureLoadBalancerInBound	Any
65500	DenyAllInBound	Any

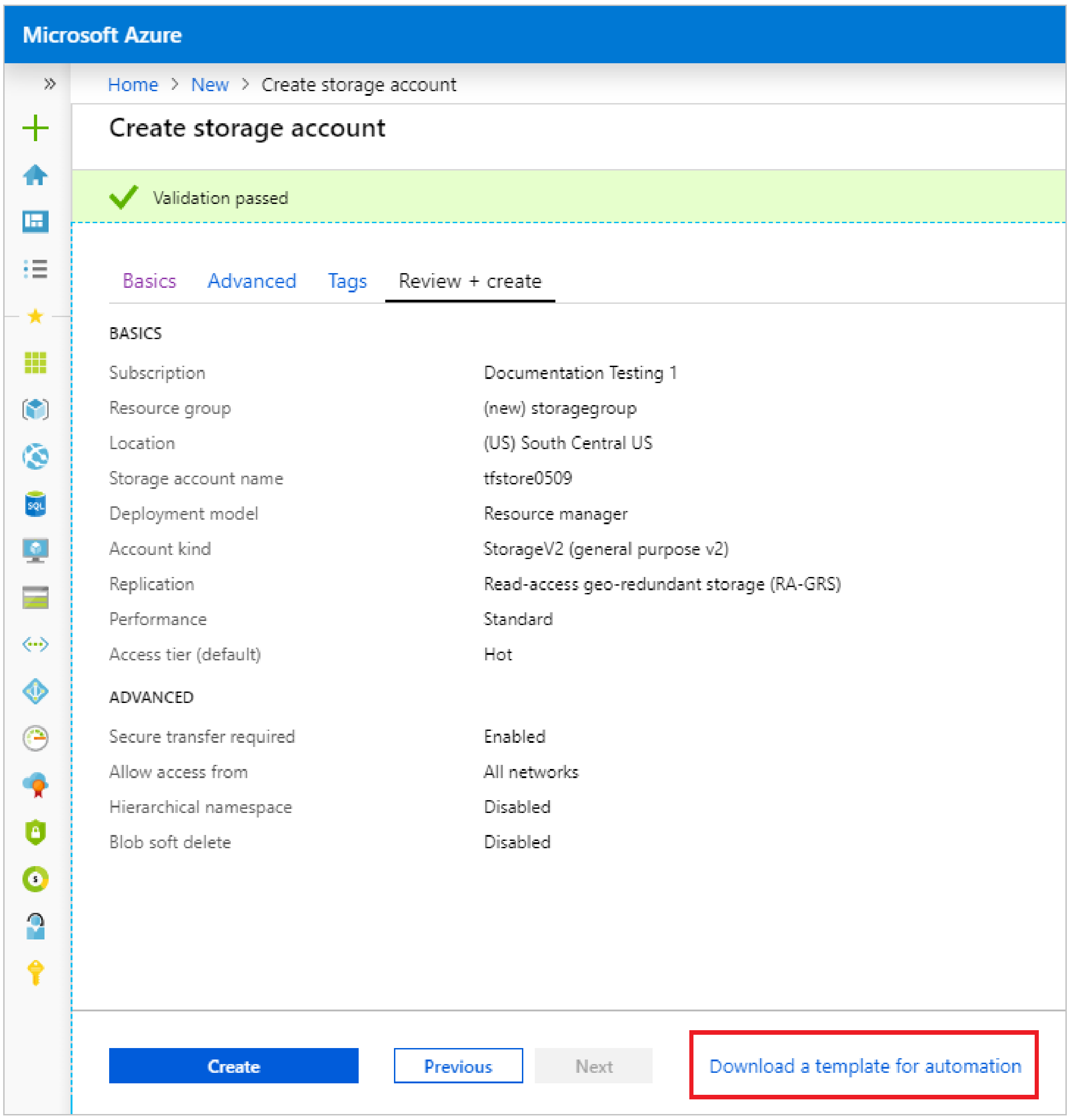
Outbound security rules

PRIORITY	NAME	PORT

4. The exported template is displayed, and is available to download. The template only contains the single resource.

Export template before deployment

1. Select the Azure service you want to deploy.
2. Fill in the values for the new service.
3. After passing validation, but before starting the deployment, select **Download a template for automation**.


Microsoft Azure
Home > New > Create storage account

Create storage account

Validation passed

Basics Advanced Tags Review + create

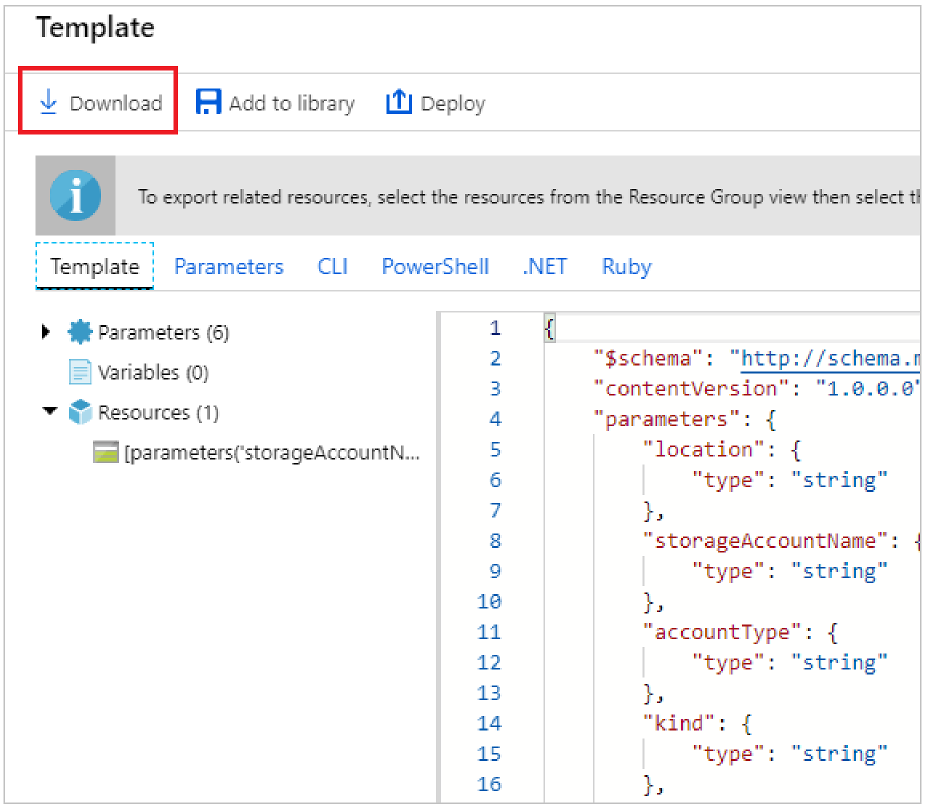
BASICS

Subscription	Documentation Testing 1
Resource group	(new) storagegroup
Location	(US) South Central US
Storage account name	tfstore0509
Deployment model	Resource manager
Account kind	StorageV2 (general purpose v2)
Replication	Read-access geo-redundant storage (RA-GRS)
Performance	Standard
Access tier (default)	Hot

ADVANCED

Secure transfer required	Enabled
Allow access from	All networks
Hierarchical namespace	Disabled
Blob soft delete	Disabled

4. The template is displayed and is available for download.


Template

Download Add to library Deploy

To export related resources, select the resources from the Resource Group view then select tl

Template Parameters CLI PowerShell .NET Ruby

- Parameters (6)
- Variables (0)
- Resources (1)
  - [parameters('storageAccountN...

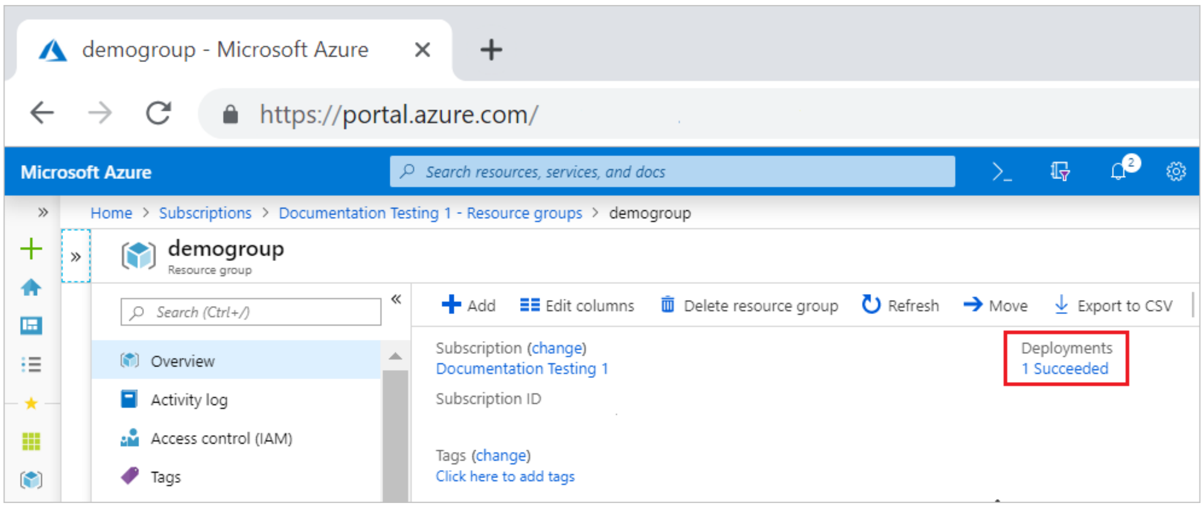
```
{
    "$schema": "http://schema.n
    "contentVersion": "1.0.0.0"
    "parameters": {
        "location": {
            "type": "string"
        },
        "storageAccountName": {
            "type": "string"
        },
        "accountType": {
            "type": "string"
        },
        "kind": {
            "type": "string"
        },
```

Export template after deployment

You can export the template that was used to deploy existing resources. The template you get is exactly the one that was used for deployment.

1. Select the resource group you want to export.

2. Select the link under **Deployments**.


demogroup - Microsoft Azure
https://portal.azure.com/
Microsoft Azure
Search resources, services, and docs
Home > Subscriptions > Documentation Testing 1 - Resource groups > demogroup
demogroup
Resource group
Search (Ctrl+/)
Overview
Activity log
Access control (IAM)
Tags
Add
Edit columns
Delete resource group
Refresh
Move
Export to CSV

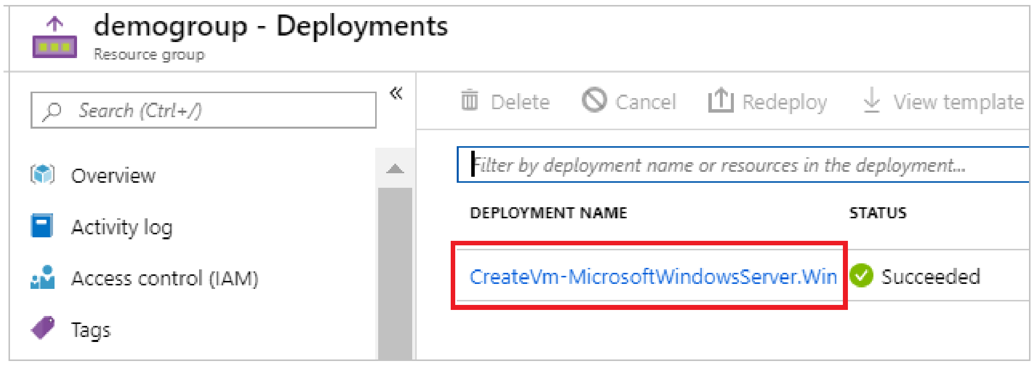
Subscription (change)
Documentation Testing 1

Subscription ID

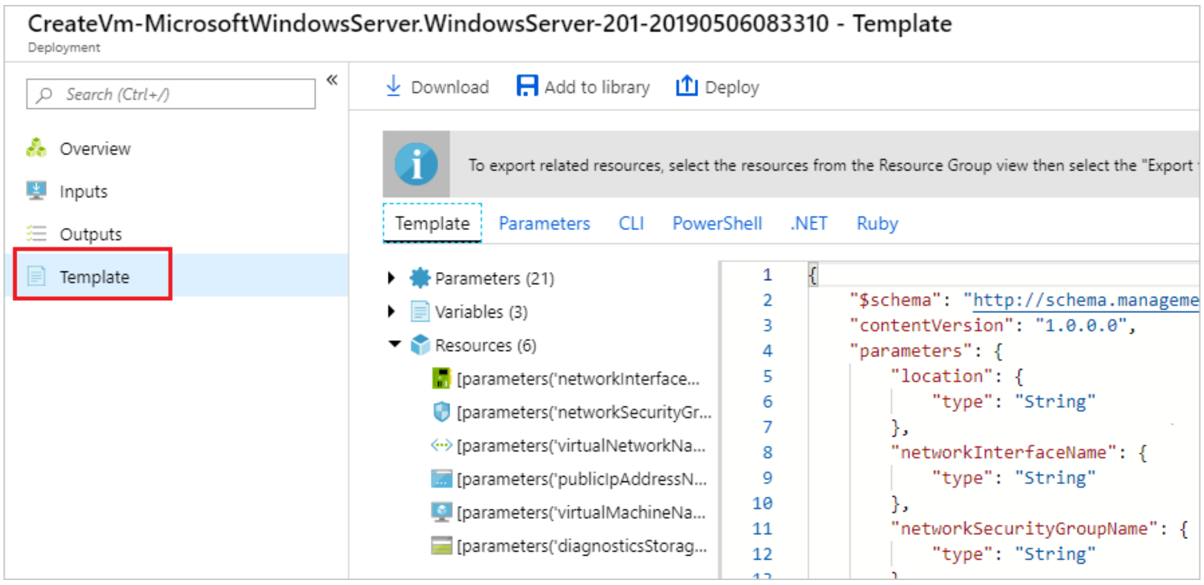
Tags (change)
Click here to add tags

Deployments
1 Succeeded

3. Select one of the deployments from the deployment history.



4. Select **Template**. The template used for this deployment is displayed, and is available for download.


CreateVm-MicrosoftWindowsServer.WindowsServer-201-20190506083310 - Template
Deployment
Search (Ctrl+/)
Overview
Inputs
Outputs
Template
Download
Add to library
Deploy
To export related resources, select the resources from the Resource Group view then select the "Export
Template
Parameters
CLI
PowerShell
.NET
Ruby
Parameters (21)
Variables (3)
Resources (6)
[parameters('networkInterface...
[parameters('networkSecurityGr...
[parameters('virtualNetworkNa...
[parameters('publicIpAddressN...
[parameters('virtualMachineNa...
[parameters('diagnosticsStorag...

```
{
    "$schema": "http://schema.manageme
    "contentVersion": "1.0.0.0",
    "parameters": {
        "location": {
            "type": "String"
        },
        "networkInterfaceName": {
            "type": "String"
        },
        "networkSecurityGroupName": {
            "type": "String"
```

Next steps

- Learn how to export templates with Azure CLI, Azure PowerShell, or REST API.
- To learn the Resource Manager template syntax, see Understand the structure and syntax of Azure Resource Manager templates.
- To learn how to develop templates, see the step-by-step tutorials.
- To view the Azure Resource Manager template schemas, see template reference.

Move resources to a new resource group or subscription

7/10/2019 • 5 minutes to read • Edit Online

This article shows you how to move Azure resources to either another Azure subscription or another resource group under the same subscription. You can use the Azure portal, Azure PowerShell, Azure CLI, or the REST API to move resources.

Both the source group and the target group are locked during the move operation. Write and delete operations are blocked on the resource groups until the move completes. This lock means you can't add, update, or delete resources in the resource groups, but it doesn't mean the resources are frozen. For example, if you move a SQL Server and its database to a new resource group, an application that uses the database experiences no downtime. It can still read and write to the database.

Moving a resource only moves it to a new resource group or subscription. It doesn't change the location of the resource.

Checklist before moving resources

There are some important steps to do before moving a resource. By verifying these conditions, you can avoid errors.

1. The resources you want to move must support the move operation. For a list of which resources support move, see Move operation support for resources.
2. Some services have specific limitations or requirements when moving resources. If you've moving any of the following services, check that guidance before moving.
   - App Services move guidance
   - Azure DevOps Services move guidance
   - Classic deployment model move guidance - Classic Compute, Classic Storage, Classic Virtual Networks, and Cloud Services
   - Recovery Services move guidance
   - Virtual Machines move guidance
   - Virtual Networks move guidance
3. The source and destination subscriptions must be active. If you have trouble enabling an account that has been disabled, create an Azure support request. Select **Subscription Management** for the issue type.
4. The source and destination subscriptions must exist within the same Azure Active Directory tenant. To check that both subscriptions have the same tenant ID, use Azure PowerShell or Azure CLI.

   For Azure PowerShell, use:

   ```
   (Get-AzSubscription -SubscriptionName <your-source-subscription>).TenantId
   (Get-AzSubscription -SubscriptionName <your-destination-subscription>).TenantId
   ```

   For Azure CLI, use:

```
az account show --subscription <your-source-subscription> --query tenantId
az account show --subscription <your-destination-subscription> --query tenantId
```

If the tenant IDs for the source and destination subscriptions aren't the same, use the following methods to reconcile the tenant IDs:

- Transfer ownership of an Azure subscription to another account
- How to associate or add an Azure subscription to Azure Active Directory

5. The destination subscription must be registered for the resource provider of the resource being moved. If not, you receive an error stating that the **subscription is not registered for a resource type**. You might see this error when moving a resource to a new subscription, but that subscription has never been used with that resource type.

   For PowerShell, use the following commands to get the registration status:

   ```
   Set-AzContext -Subscription <destination-subscription-name-or-id>
   Get-AzResourceProvider -ListAvailable | Select-Object ProviderNamespace, RegistrationState
   ```

   To register a resource provider, use:

   ```
   Register-AzResourceProvider -ProviderNamespace Microsoft.Batch
   ```

   For Azure CLI, use the following commands to get the registration status:

   ```
   az account set -s <destination-subscription-name-or-id>
   az provider list --query "[].{Provider:namespace, Status:registrationState}" --out table
   ```

   To register a resource provider, use:

   ```
   az provider register --namespace Microsoft.Batch
   ```

6. The account moving the resources must have at least the following permissions:

   - **Microsoft.Resources/subscriptions/resourceGroups/moveResources/action** on the source resource group.
   - **Microsoft.Resources/subscriptions/resourceGroups/write** on the destination resource group.

7. Before moving the resources, check the subscription quotas for the subscription you're moving the resources to. If moving the resources means the subscription will exceed its limits, you need to review whether you can request an increase in the quota. For a list of limits and how to request an increase, see Azure subscription and service limits, quotas, and constraints.

Validate move

The validate move operation lets you test your move scenario without actually moving the resources. Use this operation to check if the move will succeed. Validation is automatically called when you send a move request. Use this operation only when you need to predetermine the results. To run this operation, you need the:

- name of the source resource group
- resource ID of the target resource group
- resource ID of each resource to move
- the access token for your account

Send the following request:

```
POST https://management.azure.com/subscriptions/<subscription-id>/resourceGroups/<source-
group>/validateMoveResources?api-version=2019-05-10
Authorization: Bearer <access-token>
Content-type: application/json
```

With a request body:

```
{
 "resources": ["<resource-id-1>", "<resource-id-2>"],
 "targetResourceGroup": "/subscriptions/<subscription-id>/resourceGroups/<target-group>"
}
```

If the request is formatted correctly, the operation returns:

```
Response Code: 202
cache-control: no-cache
pragma: no-cache
expires: -1
location: https://management.azure.com/subscriptions/<subscription-id>/operationresults/<operation-id>?api-
version=2018-02-01
retry-after: 15
...
```

The 202 status code indicates the validation request was accepted, but it hasn't yet determined if the move operation will succeed. The `location` value contains a URL that you use to check the status of the long-running operation.

To check the status, send the following request:

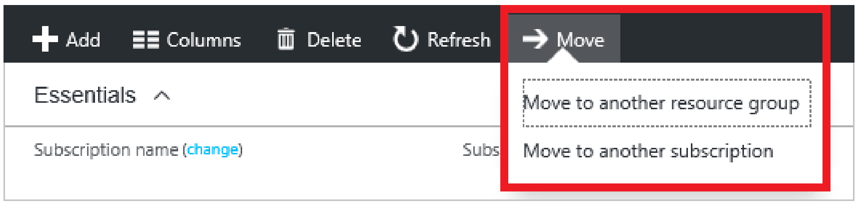
```
GET <location-url>
Authorization: Bearer <access-token>
```

While the operation is still running, you continue to receive the 202 status code. Wait the number of seconds indicated in the `retry-after` value before trying again. If the move operation validates successfully, you receive the 204 status code. If the move validation fails, you receive an error message, such as:

```
{"error":{"code":"ResourceMoveProviderValidationFailed","message":"<message>"...}}
```

Use the portal

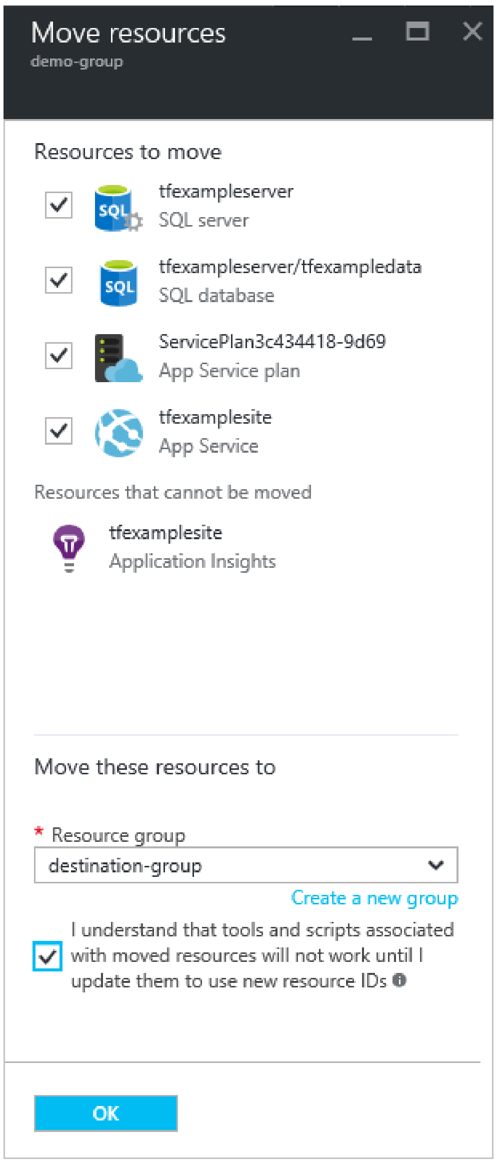
To move resources, select the resource group with those resources, and then select the **Move** button.



Add
Columns
Delete
Refresh
Move
Essentials
Subscription name (change)
Subs
Move to another resource group
Move to another subscription

Select whether you're moving the resources to a new resource group or a new subscription.

Select the resources to move and the destination resource group. Acknowledge that you need to update scripts for these resources and select **OK**. If you selected the edit subscription icon in the previous step, you must also select the destination subscription.


Move resources
demo-group

Resources to move

- ✓ tfexampleserver — SQL server
- ✓ tfexampleserver/tfexampledata — SQL database
- ✓ ServicePlan3c434418-9d69 — App Service plan
- ✓ tfexamplesite — App Service

Resources that cannot be moved

- tfexamplesite — Application Insights

Move these resources to

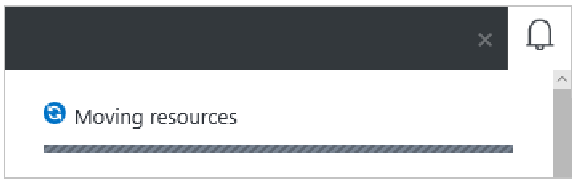
* Resource group: destination-group

Create a new group

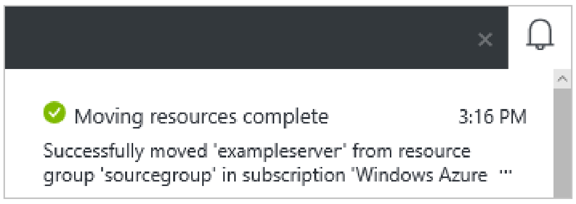
✓ I understand that tools and scripts associated with moved resources will not work until I update them to use new resource IDs

OK

In **Notifications**, you see that the move operation is running.


Moving resources

When it has completed, you're notified of the result.


Moving resources complete 3:16 PM
Successfully moved 'exampleserver' from resource group 'sourcegroup' in subscription 'Windows Azure ···

If you get an error, see Troubleshoot moving Azure resources to new resource group or subscription.

Use Azure PowerShell

To move existing resources to another resource group or subscription, use the Move-AzResource command. The following example shows how to move several resources to a new resource group.

```
$webapp = Get-AzResource -ResourceGroupName OldRG -ResourceName ExampleSite
$plan = Get-AzResource -ResourceGroupName OldRG -ResourceName ExamplePlan
Move-AzResource -DestinationResourceGroupName NewRG -ResourceId $webapp.ResourceId, $plan.ResourceId
```

To move to a new subscription, include a value for the `DestinationSubscriptionId` parameter.

If you get an error, see Troubleshoot moving Azure resources to new resource group or subscription.

Use Azure CLI

To move existing resources to another resource group or subscription, use the az resource move command. Provide the resource IDs of the resources to move. The following example shows how to move several resources to a new resource group. In the `--ids` parameter, provide a space-separated list of the resource IDs to move.

```
webapp=$(az resource show -g OldRG -n ExampleSite --resource-type "Microsoft.Web/sites" --query id --output
tsv)
plan=$(az resource show -g OldRG -n ExamplePlan --resource-type "Microsoft.Web/serverfarms" --query id --
output tsv)
az resource move --destination-group newgroup --ids $webapp $plan
```

To move to a new subscription, provide the `--destination-subscription-id` parameter.

If you get an error, see Troubleshoot moving Azure resources to new resource group or subscription.

Use REST API

To move existing resources to another resource group or subscription, use the Move resources operation.

```
POST https://management.azure.com/subscriptions/{source-subscription-id}/resourcegroups/{source-resource-
group-name}/moveResources?api-version={api-version}
```

In the request body, you specify the target resource group and the resources to move.

```
{
 "resources": ["<resource-id-1>", "<resource-id-2>"],
 "targetResourceGroup": "/subscriptions/<subscription-id>/resourceGroups/<target-group>"
}
```

If you get an error, see Troubleshoot moving Azure resources to new resource group or subscription.

Next steps

For a list of which resources support move, see Move operation support for resources.

Move operation support for resources

7/15/2019 • 11 minutes to read • Edit Online

This article lists whether an Azure resource type supports the move operation. It also provides information about special conditions to consider when moving a resource.

Jump to a resource provider namespace:

Microsoft.AAD

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
domainservices	No	No

microsoft.aadiam

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
tenants	No	No

Microsoft.AlertsManagement

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
actionrules	Yes	Yes

Microsoft.AnalysisServices

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
servers	Yes	Yes

Microsoft.ApiManagement

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
service	Yes	Yes

Microsoft.AppConfiguration

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
configurationstores	Yes	Yes

Microsoft.AppService

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
apiapps	No	No
appidentities	No	No
gateways	No	No

IMPORTANT

See App Service move guidance.

Microsoft.Authorization

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
policyassignments	No	No

Microsoft.Automation

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
automationaccounts	Yes	Yes
automationaccounts/configurations	Yes	Yes
automationaccounts/runbooks	Yes	Yes

IMPORTANT

Runbooks must exist in the same resource group as the Automation Account.

Microsoft.AzureActiveDirectory

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
b2cdirectories	Yes	Yes

Microsoft.AzureStack

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
registrations	Yes	Yes

Microsoft.Backup

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
backupvault	No	No

Microsoft.Batch

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
batchaccounts	Yes	Yes

Microsoft.BatchAI

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
clusters	No	No
fileservers	No	No
jobs	No	No
workspaces	No	No

Microsoft.BingMaps

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
mapapis	No	No

Microsoft.BizTalkServices

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
biztalk	Yes	Yes

Microsoft.Blockchain

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
blockchainmembers	Yes	Yes

Microsoft.Blueprint

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
blueprintassignments	No	No

Microsoft.BotService

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
botservices	Yes	Yes

Microsoft.Cache

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
redis	Yes	Yes

IMPORTANT

If the Azure Cache for Redis instance is configured with a virtual network, the instance can't be moved to a different subscription. See Virtual Networks move limitations.

Microsoft.Cdn

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
profiles	Yes	Yes
profiles/endpoints	Yes	Yes

Microsoft.CertificateRegistration

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
certificateorders	Yes	Yes

IMPORTANT

See App Service move guidance.

Microsoft.ClassicCompute

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
domainnames	Yes	No
virtualmachines	Yes	No

IMPORTANT

See Classic deployment move guidance. Classic deployment resources can be moved across subscriptions with an operation specific to that scenario.

Microsoft.ClassicNetwork

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
networksecuritygroups	No	No
reservedips	No	No
virtualnetworks	No	No

IMPORTANT

See Classic deployment move guidance. Classic deployment resources can be moved across subscriptions with an operation specific to that scenario.

Microsoft.ClassicStorage

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
storageaccounts	Yes	No

IMPORTANT

See Classic deployment move guidance. Classic deployment resources can be moved across subscriptions with an operation specific to that scenario.

Microsoft.CognitiveServices

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
accounts	Yes	Yes

Microsoft.Compute

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
availabilitysets	Yes	Yes
disks	Yes	Yes
galleries	No	No
galleries/images	No	No
galleries/images/versions	No	No
hostgroups	No	No
hostgroups/hosts	No	No
images	Yes	Yes

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
proximityplacementgroups	No	No
restorepointcollections	No	No
sharedvmimages	No	No
sharedvmimages/versions	No	No
snapshots	Yes	Yes
virtualmachines	Yes	Yes
virtualmachines/extensions	Yes	Yes
virtualmachinescalesets	Yes	Yes

IMPORTANT

See Virtual Machines move guidance.

Microsoft.Container

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
containergroups	No	No

Microsoft.ContainerInstance

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
containergroups	No	No

Microsoft.ContainerRegistry

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
registries	Yes	Yes
registries/buildtasks	Yes	Yes
registries/replications	Yes	Yes
registries/tasks	Yes	Yes
registries/webhooks	Yes	Yes

Microsoft.ContainerService

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
containerservices	No	No
managedclusters	No	No
openshiftmanagedclusters	No	No

Microsoft.ContentModerator

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
applications	Yes	Yes

Microsoft.CortanaAnalytics

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
accounts	No	No

Microsoft.CostManagement

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
connectors	Yes	Yes

Microsoft.CustomerInsights

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
hubs	Yes	Yes

Microsoft.DataBox

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
jobs	No	No

Microsoft.DataBoxEdge

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
databoxedgedevices	No	No

Microsoft.Databricks

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
workspaces	No	No

Microsoft.DataCatalog

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
catalogs	Yes	Yes
datacatalogs	No	No

Microsoft.DataConnect

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
connectionmanagers	No	No

Microsoft.DataExchange

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
packages	No	No
plans	No	No

Microsoft.DataFactory

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
datafactories	Yes	Yes
factories	Yes	Yes

Microsoft.DataLake

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
datalakeaccounts	No	No

Microsoft.DataLakeAnalytics

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
accounts	Yes	Yes

Microsoft.DataLakeStore

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
accounts	Yes	Yes

Microsoft.DataMigration

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
services	No	No
services/projects	No	No
slots	No	No

Microsoft.DBforMariaDB

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
servers	Yes	Yes

Microsoft.DBforMySQL

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
servers	Yes	Yes

Microsoft.DBforPostgreSQL

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
servergroups	No	No
servers	Yes	Yes
serversv2	Yes	Yes

Microsoft.DeploymentManager

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
artifactsources	Yes	Yes
rollouts	Yes	Yes
servicetopologies	Yes	Yes
servicetopologies/services	Yes	Yes
servicetopologies/services/serviceunits	Yes	Yes

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
steps	Yes	Yes

Microsoft.Devices

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
elasticpools	No	No
elasticpools/iothubtenants	No	No
iothubs	Yes	Yes
provisioningservices	Yes	Yes

Microsoft.DevSpaces

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
controllers	No	No

Microsoft.DevTestLab

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
labcenters	No	No
labs	Yes	No
labs/environments	Yes	Yes
labs/servicerunners	Yes	Yes
labs/virtualmachines	Yes	No
schedules	Yes	Yes

microsoft.dns

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
dnszones	No	No
dnszones/a	No	No
dnszones/aaaa	No	No
dnszones/cname	No	No

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
dnszones/mx	No	No
dnszones/ptr	No	No
dnszones/srv	No	No
dnszones/txt	No	No
trafficmanagerprofiles	No	No

Microsoft.DocumentDB

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
databaseaccounts	Yes	Yes

Microsoft.DomainRegistration

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
domains	Yes	Yes

Microsoft.EnterpriseKnowledgeGraph

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
services	Yes	Yes

Microsoft.EventGrid

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
domains	Yes	Yes
topics	Yes	Yes

Microsoft.EventHub

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
clusters	Yes	Yes
namespaces	Yes	Yes

Microsoft.Genomics

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
accounts	No	No

Microsoft.HanaOnAzure

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
hanainstances	Yes	Yes

Microsoft.HDInsight

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
clusters	Yes	Yes

IMPORTANT

You can move HDInsight clusters to a new subscription or resource group. However, you can't move across subscriptions the networking resources linked to the HDInsight cluster (such as the virtual network, NIC, or load balancer). In addition, you can't move to a new resource group a NIC that is attached to a virtual machine for the cluster.

When moving an HDInsight cluster to a new subscription, first move other resources (like the storage account). Then, move the HDInsight cluster by itself.

Microsoft.HealthcareApis

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
services	Yes	Yes

Microsoft.HybridCompute

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
machines	No	No

Microsoft.HybridData

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
datamanagers	Yes	Yes

Microsoft.ImportExport

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
jobs	Yes	Yes

microsoft.insights

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
accounts	No	No
actiongroups	Yes	Yes
activitylogalerts	No	No
alertrules	Yes	Yes
autoscalesettings	Yes	Yes
components	Yes	Yes
guestdiagnosticsettings	No	No
metricalerts	No	No
notificationgroups	No	No
notificationrules	No	No
scheduledqueryrules	Yes	Yes
webtests	Yes	Yes
workbooks	Yes	Yes

IMPORTANT

Make sure moving to new subscription doesn't exceed subscription quotas.

Microsoft.IoTCentral

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
iotapps	Yes	Yes

Microsoft.IoTSpaces

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
checknameavailability	Yes	Yes
graph	Yes	Yes

Microsoft.KeyVault

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
hsmpools	No	No
vaults	Yes	Yes

IMPORTANT

Key Vaults used for disk encryption can't be moved to a resource group in the same subscription or across subscriptions.

Microsoft.Kusto

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
clusters	Yes	Yes

Microsoft.LabServices

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
labaccounts	No	No

Microsoft.LocationBasedServices

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
accounts	Yes	Yes

Microsoft.LocationServices

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
accounts	No	No

Microsoft.Logic

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
hostingenvironments	No	No
integrationaccounts	Yes	Yes
integrationserviceenvironments	No	No
isolatedenvironments	No	No
workflows	Yes	Yes

Microsoft.MachineLearning

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
commitmentplans	Yes	Yes
webservices	Yes	No
workspaces	Yes	Yes

Microsoft.MachineLearningCompute

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
operationalizationclusters	Yes	Yes

Microsoft.MachineLearningExperimentation

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
accounts	No	No
accounts/workspaces	No	No
accounts/workspaces/projects	No	No
teamaccounts	No	No
teamaccounts/workspaces	No	No
teamaccounts/workspaces/projects	No	No

Microsoft.MachineLearningModelManagement

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
accounts	Yes	Yes

Microsoft.MachineLearningOperationalization

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
hostingaccounts	No	No

Microsoft.MachineLearningServices

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
workspaces	No	No

Microsoft.ManagedIdentity

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
userassignedidentities	No	No

Microsoft.Maps

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
accounts	Yes	Yes

Microsoft.MarketplaceApps

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
classicdevservices	No	No

Microsoft.Media

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
mediaservices	Yes	Yes
mediaservices/liveevents	Yes	Yes
mediaservices/streamingendpoints	Yes	Yes

Microsoft.Migrate

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
assessmentprojects	No	No
migrateprojects	No	No
projects	No	No

Microsoft.NetApp

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
netappaccounts	No	No
netappaccounts/capacitypools	No	No
netappaccounts/capacitypools/volumes	No	No

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
netappaccounts/capacitypools/volumes/mounttargets	No	No
netappaccounts/capacitypools/volumes/snapshots	No	No

Microsoft.Network

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
applicationgateways	No	No
applicationgatewaywebapplicationfirewallpolicies	No	No
applicationsecuritygroups	Yes	Yes
azurefirewalls	Yes	Yes
bastionhosts	No	No
connections	Yes	Yes
ddoscustompolicies	Yes	Yes
ddosprotectionplans	No	No
dnszones	Yes	Yes
expressroutecircuits	No	No
expressroutecrossconnections	No	No
expressroutegateways	No	No
expressrouteports	No	No
frontdoors	No	No
frontdoorwebapplicationfirewallpolicies	No	No
loadbalancers	Yes - Basic SKU No - Standard SKU	Yes - Basic SKU No - Standard SKU
localnetworkgateways	Yes	Yes
natgateways	Yes	Yes
networkintentpolicies	Yes	Yes

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
networkinterfaces	Yes	Yes
networkprofiles	No	No
networksecuritygroups	Yes	Yes
networkwatchers	Yes	Yes
networkwatchers/connectionmonitors	Yes	Yes
networkwatchers/lenses	Yes	Yes
networkwatchers/pingmeshes	Yes	Yes
p2svpngateways	No	No
privatednszones	Yes	Yes
privatednszones/virtualnetworklinks	Yes	Yes
privateendpoints	No	No
privatelinkservices	No	No
publicipaddresses	Yes - Basic SKU No - Standard SKU	Yes - Basic SKU No - Standard SKU
publicipprefixes	Yes	Yes
routefilters	No	No
routetables	Yes	Yes
securegateways	Yes	Yes
serviceendpointpolicies	Yes	Yes
trafficmanagerprofiles	Yes	Yes
virtualhubs	No	No
virtualnetworkgateways	Yes	Yes
virtualnetworks	Yes	Yes
virtualnetworktaps	No	No
virtualwans	No	No
vpngateways	No	No

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
vpnsites	No	No
webapplicationfirewallpolicies	Yes	Yes

IMPORTANT

See Virtual Networks move guidance.

Microsoft.NotificationHubs

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
namespaces	Yes	Yes
namespaces/notificationhubs	Yes	Yes

Microsoft.OperationalInsights

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
workspaces	Yes	Yes

IMPORTANT

Make sure moving to new subscription doesn't exceed subscription quotas.

Microsoft.OperationsManagement

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
managementconfigurations	Yes	Yes
solutions	Yes	Yes
views	Yes	Yes

Microsoft.Peering

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
peerings	No	No

Microsoft.Portal

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
dashboards	Yes	Yes

Microsoft.PortalSdk

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
rootresources	No	No

Microsoft.PowerBI

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
workspacecollections	Yes	Yes

Microsoft.PowerBIDedicated

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
capacities	Yes	Yes

Microsoft.ProjectOxford

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
accounts	No	No

Microsoft.RecoveryServices

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
vaults	Yes	Yes

IMPORTANT

See Recovery Services move guidance.

Microsoft.Relay

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
namespaces	Yes	Yes

Microsoft.SaaS

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
applications	Yes	No

Microsoft.Scheduler

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
flows	Yes	Yes
jobcollections	Yes	Yes

Microsoft.Search

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
searchservices	Yes	Yes

IMPORTANT

You can't move several Search resources in different regions in one operation. Instead, move them in separate operations.

Microsoft.Security

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
iotsecuritysolutions	Yes	Yes

Microsoft.ServerManagement

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
gateways	No	No
nodes	No	No

Microsoft.ServiceBus

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
namespaces	Yes	Yes

Microsoft.ServiceFabric

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
applications	No	No

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
clusters	Yes	Yes
containergroups	No	No
containergroupsets	No	No
edgeclusters	No	No
networks	No	No
secretstores	No	No
volumes	No	No

Microsoft.ServiceFabricMesh

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
applications	Yes	Yes
containergroups	No	No
gateways	Yes	Yes
networks	Yes	Yes
secrets	Yes	Yes
volumes	Yes	Yes

Microsoft.SignalRService

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
signalr	Yes	Yes

Microsoft.SiteRecovery

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
siterecoveryvault	No	No

IMPORTANT

See Recovery Services move guidance.

Microsoft.Solutions

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
appliancedefinitions	No	No
appliances	No	No
applicationdefinitions	No	No
applications	No	No
jitrequests	No	No

Microsoft.Sql

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
instancepools	No	No
managedinstances	No	No
managedinstances/databases	No	No
servers	Yes	Yes
servers/databases	Yes	Yes
servers/elasticpools	Yes	Yes
virtualclusters	Yes	Yes

IMPORTANT
A database and server must be in the same resource group. When you move a SQL server, all its databases are also moved. This behavior applies to Azure SQL Database and Azure SQL Data Warehouse databases.

Microsoft.SqlVirtualMachine

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
sqlvirtualmachinegroups	Yes	Yes
sqlvirtualmachines	Yes	Yes

Microsoft.SqlVM

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
dwvm	No	No

Microsoft.Storage

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
storageaccounts	Yes	Yes

Microsoft.StorageCache

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
caches	No	No

Microsoft.StorageSync

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
storagesyncservices	Yes	Yes

Microsoft.StorageSyncDev

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
storagesyncservices	No	No

Microsoft.StorageSyncInt

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
storagesyncservices	No	No

Microsoft.StorSimple

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
managers	No	No

Microsoft.StreamAnalytics

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
streamingjobs	Yes	Yes

IMPORTANT

Stream Analytics jobs can't be moved when in running state.

Microsoft.StreamAnalyticsExplorer

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
environments	No	No
environments/eventsources	No	No
instances	No	No
instances/environments	No	No
instances/environments/eventsources	No	No

Microsoft.TerraformOSS

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
providerregistrations	No	No
resources	No	No

Microsoft.TimeSeriesInsights

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
environments	Yes	Yes
environments/eventsources	Yes	Yes
environments/referencedatasets	Yes	Yes

Microsoft.Token

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
stores	No	No

Microsoft.VirtualMachineImages

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
imagetemplates	No	No

microsoft.visualstudio

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
account	Yes	Yes
account/extension	Yes	Yes

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
account/project	Yes	Yes

IMPORTANT

To change the subscription for Azure DevOps, see change the Azure subscription used for billing.

Microsoft.VMwareCloudSimple

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
dedicatedcloudnodes	Yes	Yes
dedicatedcloudservices	Yes	Yes
virtualmachines	Yes	Yes

Microsoft.Web

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
certificates	No	Yes
connectiongateways	Yes	Yes
connections	Yes	Yes
customapis	Yes	Yes
hostingenvironments	No	No
serverfarms	Yes	Yes
sites	Yes	Yes
sites/premieraddons	Yes	Yes
sites/slots	Yes	Yes

IMPORTANT

See App Service move guidance.

Microsoft.WindowsIoT

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
deviceservices	No	No

Microsoft.WindowsVirtualDesktop

RESOURCE TYPE	RESOURCE GROUP	SUBSCRIPTION
applicationgroups	No	No
hostpools	No	No
workspaces	No	No

Third-party services

Third-party services currently don't support the move operation.

Next steps

For commands to move resources, see Move resources to new resource group or subscription.

To get the same data as a file of comma-separated values, download move-support-resources.csv.

Troubleshoot moving Azure resources to new resource group or subscription

7/10/2019 • 2 minutes to read • Edit Online

This article provides suggestions to help resolve problems when moving resources.

Upgrade a subscription

If you actually want to upgrade your Azure subscription (such as switching from free to pay-as-you-go), you need to convert your subscription.

- To upgrade a free trial, see Upgrade your Free Trial or Microsoft Imagine Azure subscription to Pay-As-You-Go.
- To change a pay-as-you-go account, see Change your Azure Pay-As-You-Go subscription to a different offer.

If you can't convert the subscription, create an Azure support request. Select **Subscription Management** for the issue type.

Service limitations

Some services require additional considerations when moving resources. If you're moving the following services, make sure you check the guidance and limitations.

- App Services
- Azure DevOps Services
- Classic deployment model
- Recovery Services
- Virtual Machines
- Virtual Networks

Large requests

When possible, break large moves into separate move operations. Resource Manager immediately returns an error when there are more than 800 resources in a single operation. However, moving less than 800 resources may also fail by timing out.

Next steps

For commands to move resources, see Move resources to new resource group or subscription.

Move guidance for App Service resources

7/10/2019 • 2 minutes to read • Edit Online

The steps to move App Service resources differ based on whether you're moving the resources within a subscription or to a new subscription.

Move in same subscription

When moving a Web App *within the same subscription*, you can't move third-party SSL certificates. However, you can move a Web App to the new resource group without moving its third-party certificate, and your app's SSL functionality still works.

If you want to move the SSL certificate with the Web App, follow these steps:

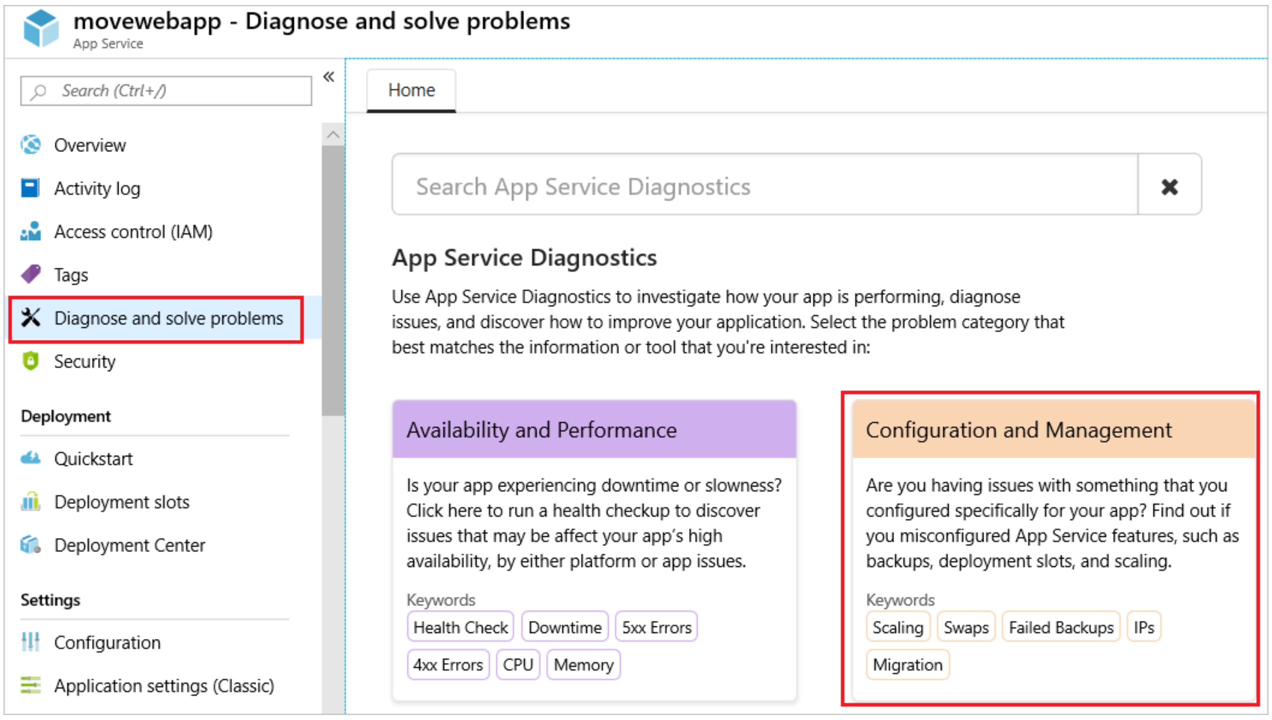
1. Delete the third-party certificate from the Web App, but keep a copy of your certificate
2. Move the Web App.
3. Upload the third-party certificate to the moved Web App.

Move across subscriptions

When moving a Web App *across subscriptions*, the following limitations apply:

- The destination resource group must not have any existing App Service resources. App Service resources include:
  - Web Apps
  - App Service plans
  - Uploaded or imported SSL certificates
  - App Service Environments
- All App Service resources in the resource group must be moved together.
- App Service resources can only be moved from the resource group in which they were originally created. If an App Service resource is no longer in its original resource group, move it back to its original resource group. Then, move the resource across subscriptions.

If you don't remember the original resource group, you can find it through diagnostics. For your web app, select **Diagnose and solve problems**. Then, select **Configuration and Management**.


movewebapp - Diagnose and solve problems
App Service

App Service Diagnostics

Use App Service Diagnostics to investigate how your app is performing, diagnose issues, and discover how to improve your application. Select the problem category that best matches the information or tool that you're interested in:

Availability and Performance

Is your app experiencing downtime or slowness? Click here to run a health checkup to discover issues that may be affect your app's high availability, by either platform or app issues.

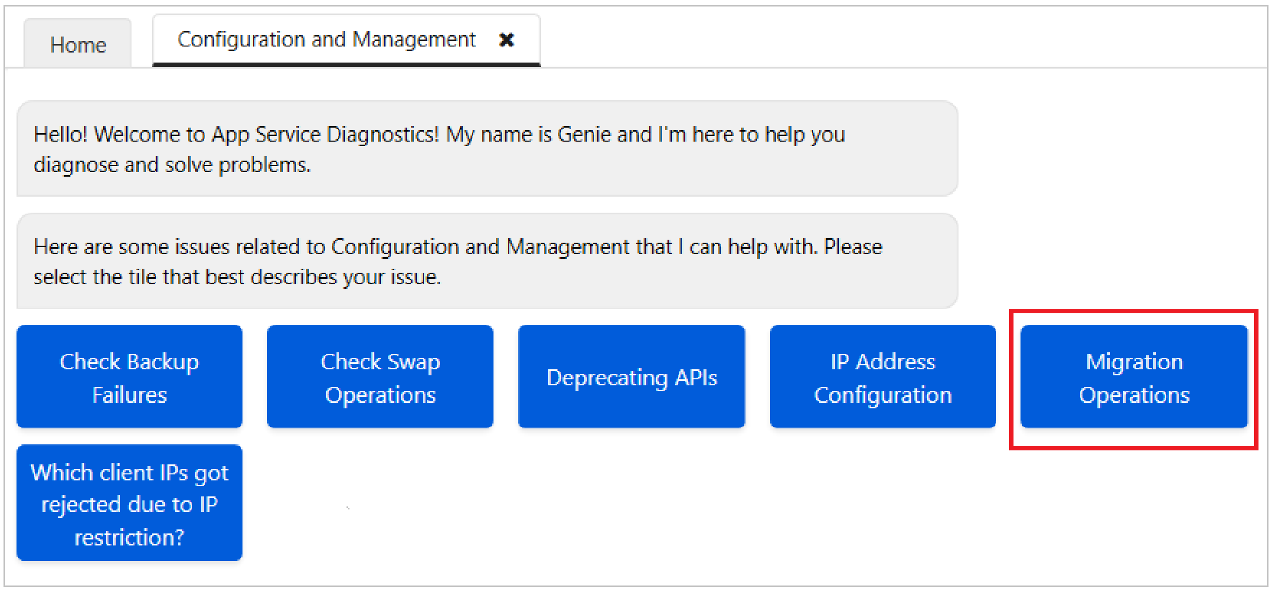
Keywords: Health Check, Downtime, 5xx Errors, 4xx Errors, CPU, Memory

Configuration and Management

Are you having issues with something that you configured specifically for your app? Find out if you misconfigured App Service features, such as backups, deployment slots, and scaling.

Keywords: Scaling, Swaps, Failed Backups, IPs, Migration

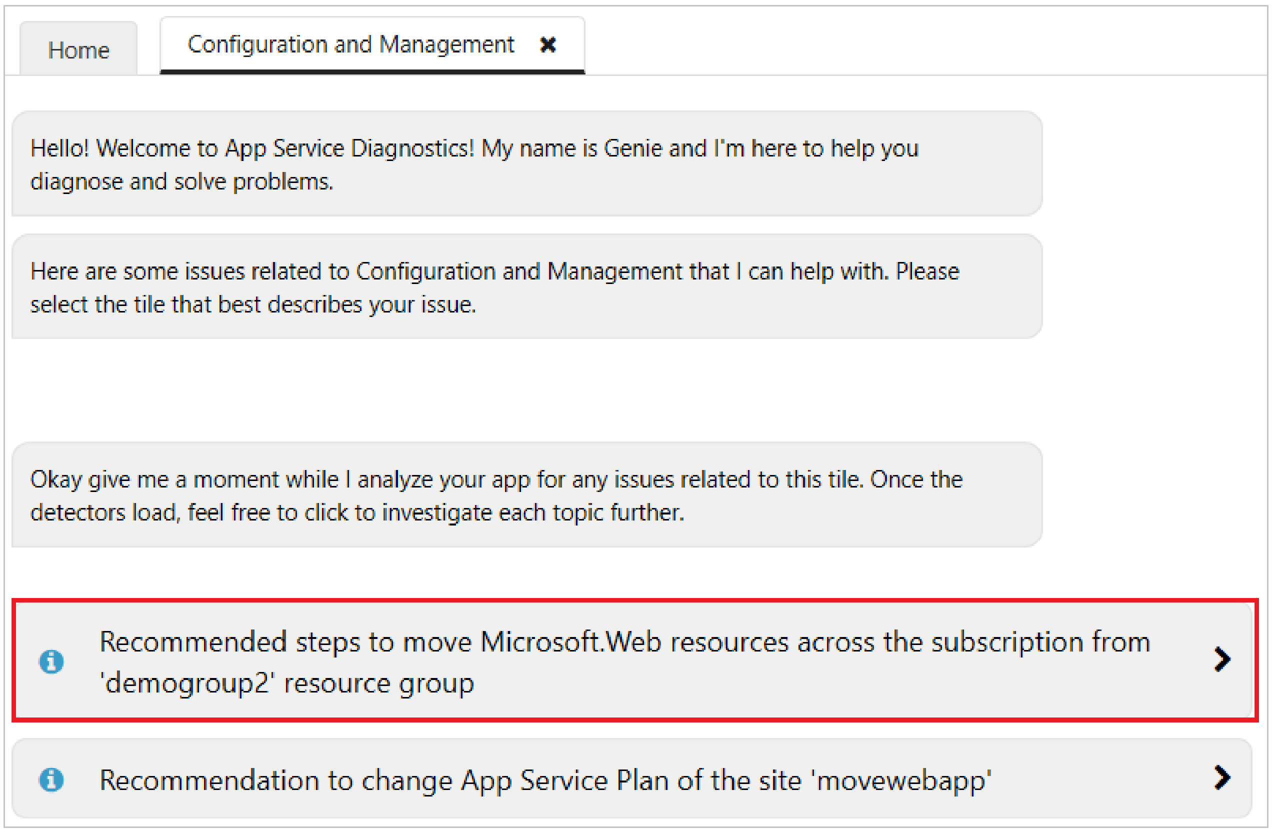
Select **Migration Options**.



Hello! Welcome to App Service Diagnostics! My name is Genie and I'm here to help you diagnose and solve problems.

Here are some issues related to Configuration and Management that I can help with. Please select the tile that best describes your issue.

Select the option for recommended steps to move the web app.



Hello! Welcome to App Service Diagnostics! My name is Genie and I'm here to help you diagnose and solve problems.

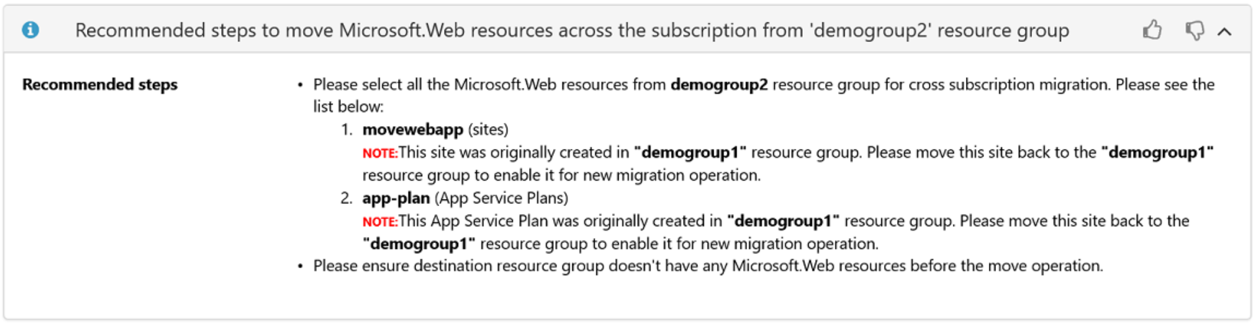
Here are some issues related to Configuration and Management that I can help with. Please select the tile that best describes your issue.

Okay give me a moment while I analyze your app for any issues related to this tile. Once the detectors load, feel free to click to investigate each topic further.

Recommended steps to move Microsoft.Web resources across the subscription from 'demogroup2' resource group

Recommendation to change App Service Plan of the site 'movewebapp'

You see the recommended actions to take before moving the resources. The information includes the original resource group for the web app.



Recommended steps to move Microsoft.Web resources across the subscription from 'demogroup2' resource group

Recommended steps

- Please select all the Microsoft.Web resources from **demogroup2** resource group for cross subscription migration. Please see the list below:
  1. **movewebapp** (sites)
     NOTE:This site was originally created in **"demogroup1"** resource group. Please move this site back to the **"demogroup1"** resource group to enable it for new migration operation.
  2. **app-plan** (App Service Plans)
     NOTE:This App Service Plan was originally created in **"demogroup1"** resource group. Please move this site back to the **"demogroup1"** resource group to enable it for new migration operation.
- Please ensure destination resource group doesn't have any Microsoft.Web resources before the move operation.

Move App Service Certificate

You can move your App Service Certificate to a new resource group or subscription. If your App Service Certificate is bound to a web app, you must take some steps before moving the resources to a new subscription. Delete the SSL binding and private certificate from the web app before moving the resources. The App Service Certificate doesn't need to be deleted, just the private certificate in the web app.

Move support

To determine which App Service resources can be moved, see move support status for:

- Microsoft.AppService
- Microsoft.CertificateRegistration
- Microsoft.DomainRegistration
- Microsoft.Web

Next steps

For commands to move resources, see Move resources to new resource group or subscription.

Move guidance for Classic deployment model resources

7/10/2019 • 2 minutes to read • Edit Online

The steps to move resources deployed through the classic model differ based on whether you're moving the resources within a subscription or to a new subscription.

Move in the same subscription

When moving resources from one resource group to another resource group within the same subscription, the following restrictions apply:

- Virtual networks (classic) can't be moved.
- Virtual machines (classic) must be moved with the cloud service.
- Cloud service can only be moved when the move includes all its virtual machines.
- Only one cloud service can be moved at a time.
- Only one storage account (classic) can be moved at a time.
- Storage account (classic) can't be moved in the same operation with a virtual machine or a cloud service.

To move classic resources to a new resource group within the same subscription, use the standard move operations through the portal, Azure PowerShell, Azure CLI, or REST API. You use the same operations as you use for moving Resource Manager resources.

Move across subscriptions

When moving resources to a new subscription, the following restrictions apply:

- All classic resources in the subscription must be moved in the same operation.
- The target subscription must not have any other classic resources.
- The move can only be requested through a separate REST API for classic moves. The standard Resource Manager move commands don't work when moving classic resources to a new subscription.

To move classic resources to a new subscription, use the REST operations that are specific to classic resources. To use REST, do the following steps:

1. Check if the source subscription can participate in a cross-subscription move. Use the following operation:

   ```
   POST
   https://management.azure.com/subscriptions/{sourceSubscriptionId}/providers/Microsoft.ClassicCompute/va
   lidateSubscriptionMoveAvailability?api-version=2016-04-01
   ```

   In the request body, include:

   ```
   {
    "role": "source"
   }
   ```

   The response for the validation operation is in the following format:

```
{
 "status": "{status}",
 "reasons": [
   "reason1",
   "reason2"
 ]
}
```

2. Check if the destination subscription can participate in a cross-subscription move. Use the following operation:

   ```
   POST
   https://management.azure.com/subscriptions/{destinationSubscriptionId}/providers/Microsoft.ClassicCompu
   te/validateSubscriptionMoveAvailability?api-version=2016-04-01
   ```

   In the request body, include:

   ```
   {
    "role": "target"
   }
   ```

   The response is in the same format as the source subscription validation.

3. If both subscriptions pass validation, move all classic resources from one subscription to another subscription with the following operation:

   ```
   POST https://management.azure.com/subscriptions/{subscription-
   id}/providers/Microsoft.ClassicCompute/moveSubscriptionResources?api-version=2016-04-01
   ```

   In the request body, include:

   ```
   {
    "target": "/subscriptions/{target-subscription-id}"
   }
   ```

The operation may run for several minutes.

Next steps

If you have trouble moving classic resources, contact Support.

For commands to move resources, see Move resources to new resource group or subscription.

Move a Recovery Services vault across Azure Subscriptions and Resource Groups

7/31/2019 • 5 minutes to read • Edit Online

This article explains how to move a Recovery Services vault configured for Azure Backup across Azure subscriptions, or to another resource group in the same subscription. You can use the Azure portal or PowerShell to move a Recovery Services vault.

Supported region

Resource move for Recovery Services vault is supported in Australia East, Australia South East, Canada Central, Canada East, South East Asia, East Asia, Central US, North Central US, East US, East US2, South central US, West Central US, West Central US2, West US, Central India, South India, Japan East, Japan West, Korea Central, Korea South, North Europe, West Europe, South Africa North, South Africa West, UK South, and UK West.

Prerequisites for moving Recovery Services vault

- During vault move across resource groups, both the source and target resource groups are locked preventing the write and delete operations. For more information, see this article.
- Only admin subscription has the permissions to move a vault.
- For moving vault across subscriptions, the target subscription must reside in the same tenant as the source subscription and its state should be enabled.
- You must have permission to perform write operations on the target resource group.
- Moving the vault only changes the resource group. The Recovery Services vault will reside on the same location and it cannot be changed.
- You can move only one Recovery Services vault, per region, at a time.
- If a VM doesn't move with the Recovery Services vault across subscriptions, or to a new resource group, the current VM recovery points will remain intact in the vault until they expire.
- Whether the VM is moved with the vault or not, you can always restore the VM from the retained backup history in the vault.
- The Azure Disk Encryption requires that the key vault and VMs reside in the same Azure region and subscription.
- To move a virtual machine with managed disks, see this article.
- The options for moving resources deployed through the Classic model differ depending on whether you are moving the resources within a subscription, or to a new subscription. For more information, see this article.
- Backup policies defined for the vault are retained after the vault moves across subscriptions or to a new resource group.
- Moving vault with the Azure Files, Azure File Sync, or SQL in IaaS VMs across subscriptions and resource groups is not supported.
- If you move a vault containing VM backup data, across subscriptions, you must move your VMs to the same subscription, and use the same target resource group to continue backups.

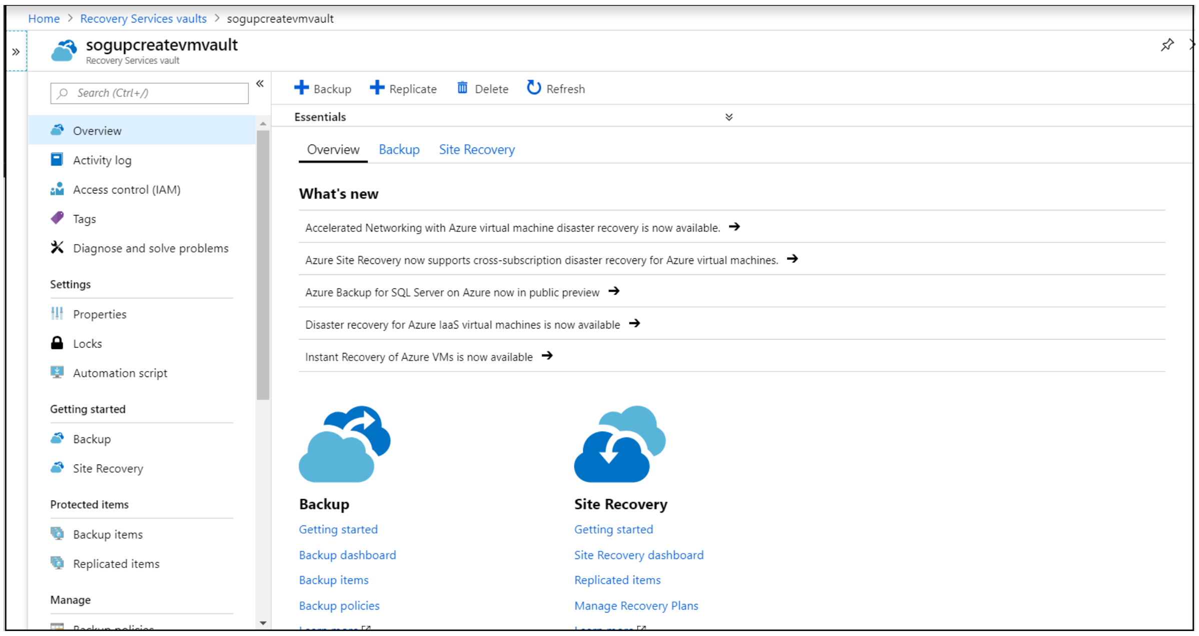
NOTE

Recovery Services vaults configured to use with **Azure Site Recovery** can't move, yet. If you have configured any VMs (Azure IaaS, Hyper-V, VMware) or physical machines for disaster recovery using the **Azure Site Recovery**, the move operation will be blocked. The resource move feature for Site Recovery service is not yet available.

Use Azure portal to move Recovery Services vault to different resource group

To move a recovery services vault and its associated resources to different resource group

1. Sign in to the Azure portal.
2. Open the list of **Recovery Services vaults** and select the vault you want to move. When the vault dashboard opens, it appears as shown in the following image.



Home > Recovery Services vaults > sogupcreatevmvault

sogupcreatevmvault
Recovery Services vault

Backup | Replicate | Delete | Refresh

Essentials

Overview | Backup | Site Recovery

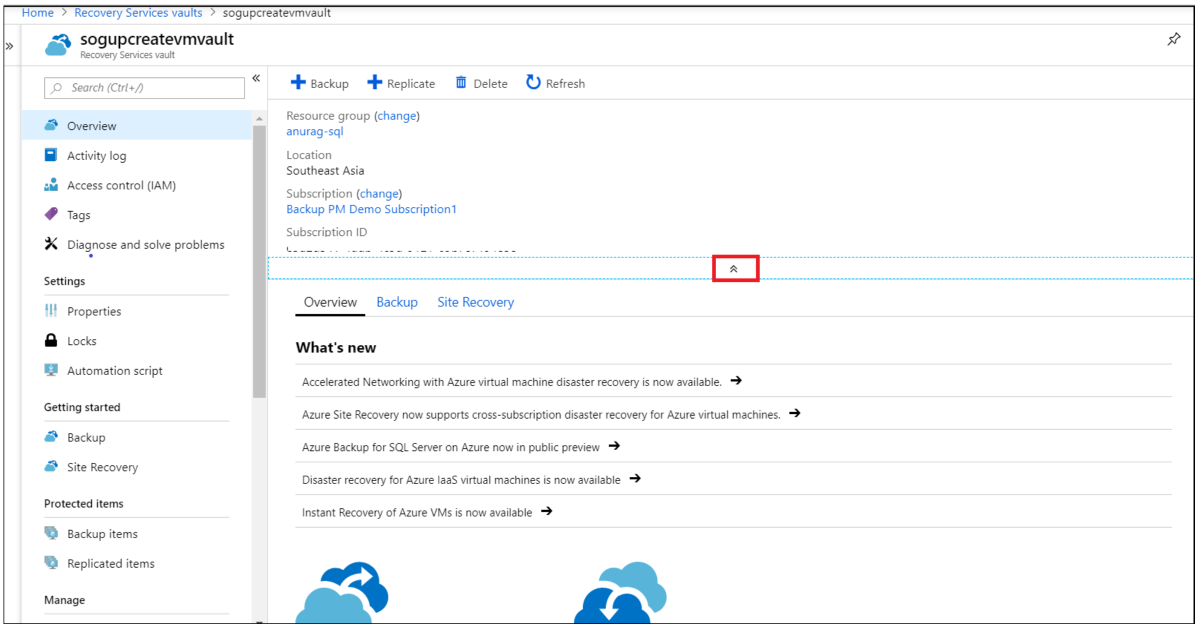
What's new

- Accelerated Networking with Azure virtual machine disaster recovery is now available.
- Azure Site Recovery now supports cross-subscription disaster recovery for Azure virtual machines.
- Azure Backup for SQL Server on Azure now in public preview
- Disaster recovery for Azure IaaS virtual machines is now available
- Instant Recovery of Azure VMs is now available

Backup: Getting started, Backup dashboard, Backup items, Backup policies

Site Recovery: Getting started, Site Recovery dashboard, Replicated items, Manage Recovery Plans

If you do not see the **Essentials** information for your vault, click the drop-down icon. You should now see the Essentials information for your vault.



Home > Recovery Services vaults > sogupcreatevmvault

sogupcreatevmvault
Recovery Services vault

Backup | Replicate | Delete | Refresh

Resource group (change)
anurag-sql

Location
Southeast Asia

Subscription (change)
Backup PM Demo Subscription1

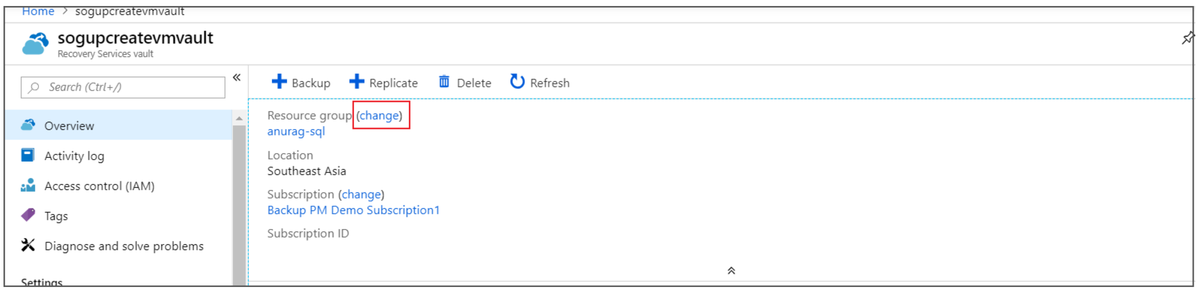
Subscription ID

Overview | Backup | Site Recovery

What's new

- Accelerated Networking with Azure virtual machine disaster recovery is now available.
- Azure Site Recovery now supports cross-subscription disaster recovery for Azure virtual machines.
- Azure Backup for SQL Server on Azure now in public preview
- Disaster recovery for Azure IaaS virtual machines is now available
- Instant Recovery of Azure VMs is now available

3. In the vault overview menu, click **change** next to the **Resource group**, to open the **Move resources** blade.


Home > sogupcreatevmvault

sogupcreatevmvault
Recovery Services vault

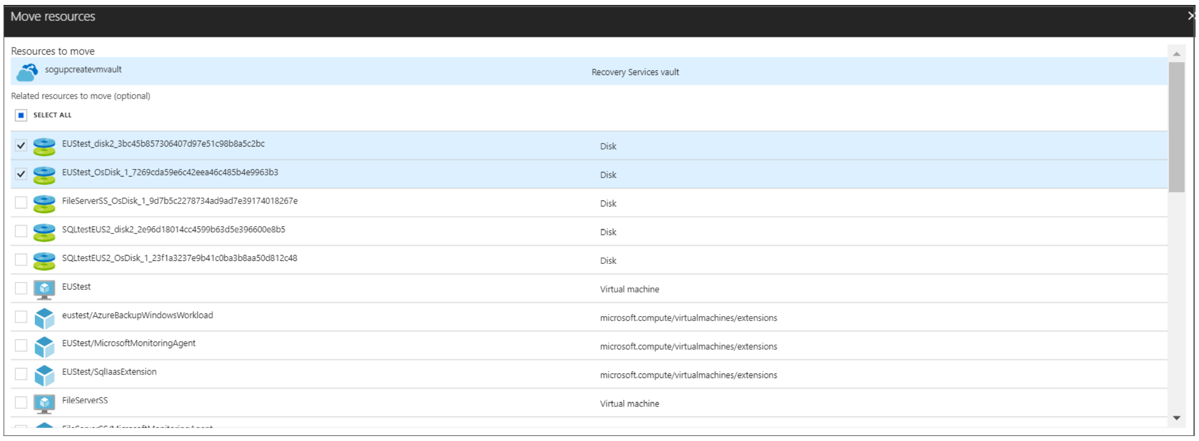
Resource group (change)
anurag-sql

Location
Southeast Asia

Subscription (change)
Backup PM Demo Subscription1

Subscription ID

4. In the **Move resources** blade, for the selected vault it is recommended to move the optional related resources by selecting the checkbox as shown in the following image.


Move resources

Resources to move

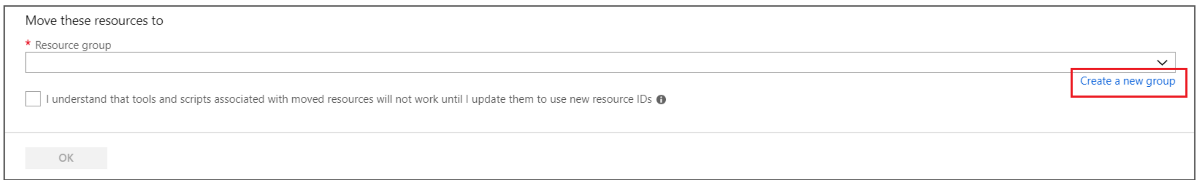
sogupcreatevmvault	Recovery Services vault

Related resources to move (optional)

SELECT ALL

✓	EUStest_disk2_3bc45b857306407d97e51c98b8a5c2bc	Disk
✓	EUStest_OsDisk_1_7269cda59e6c42eea46c485b4e9963b3	Disk
	FileServerSS_OsDisk_1_9d7b5c2278734ad9ad7e39174018267e	Disk
	SQLtestEUS2_disk2_2e96d18014cc4599b63d5e396600e8b5	Disk
	SQLtestEUS2_OsDisk_1_23f1a3237e9b41c0ba3b8aa50d812c48	Disk
	EUStest	Virtual machine
	eustest/AzureBackupWindowsWorkload	microsoft.compute/virtualmachines/extensions
	EUStest/MicrosoftMonitoringAgent	microsoft.compute/virtualmachines/extensions
	EUStest/SqlIaasExtension	microsoft.compute/virtualmachines/extensions
	FileServerSS	Virtual machine

5. To add the target resource group, in the **Resource group** drop-down list select an existing resource group or click **create a new group** option.

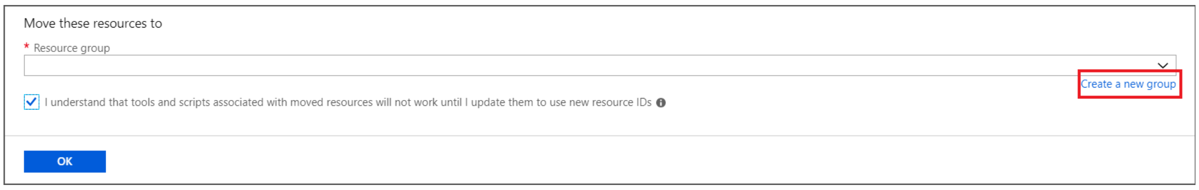

Move these resources to

* Resource group

Create a new group

I understand that tools and scripts associated with moved resources will not work until I update them to use new resource IDs

6. After adding the resource group, confirm **I understand that tools and scripts associated with moved resources will not work until I update them to use new resource IDs** option and then click **OK** to complete moving the vault.


Move these resources to

* Resource group

Create a new group

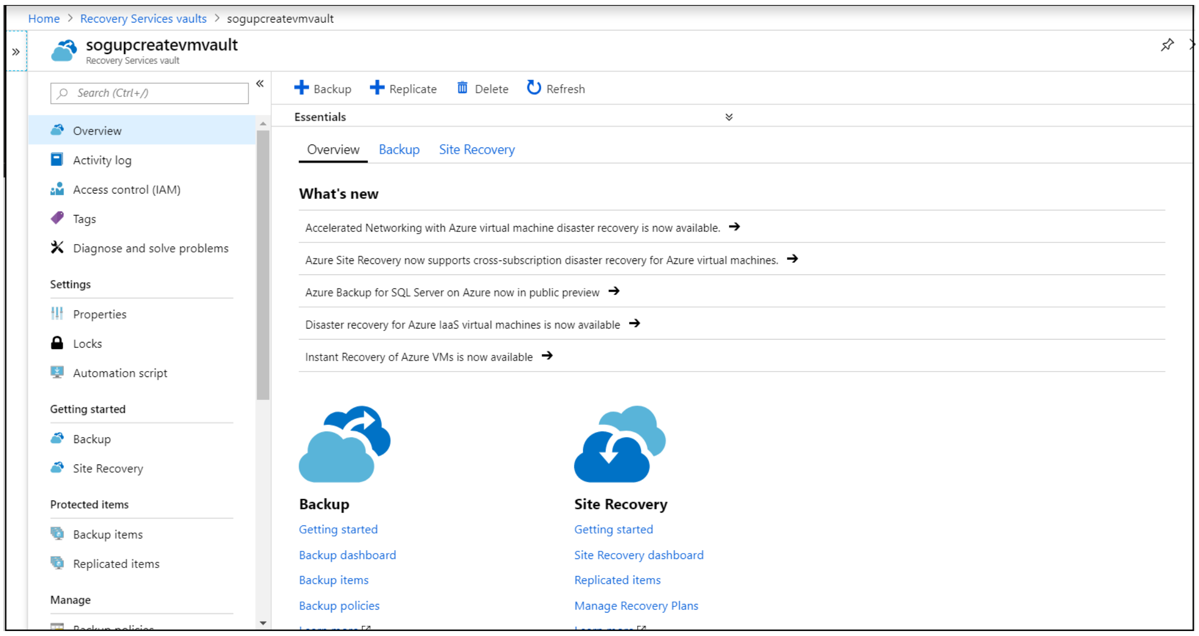
✓ I understand that tools and scripts associated with moved resources will not work until I update them to use new resource IDs

Use Azure portal to move Recovery Services vault to a different subscription

You can move a Recovery Services vault and its associated resources to a different subscription

1. Sign in to the Azure portal.

2. Open the list of Recovery Services vaults and select the vault you want to move. When the vault dashboard opens, it appears as shown the following image.



What's new

Accelerated Networking with Azure virtual machine disaster recovery is now available.

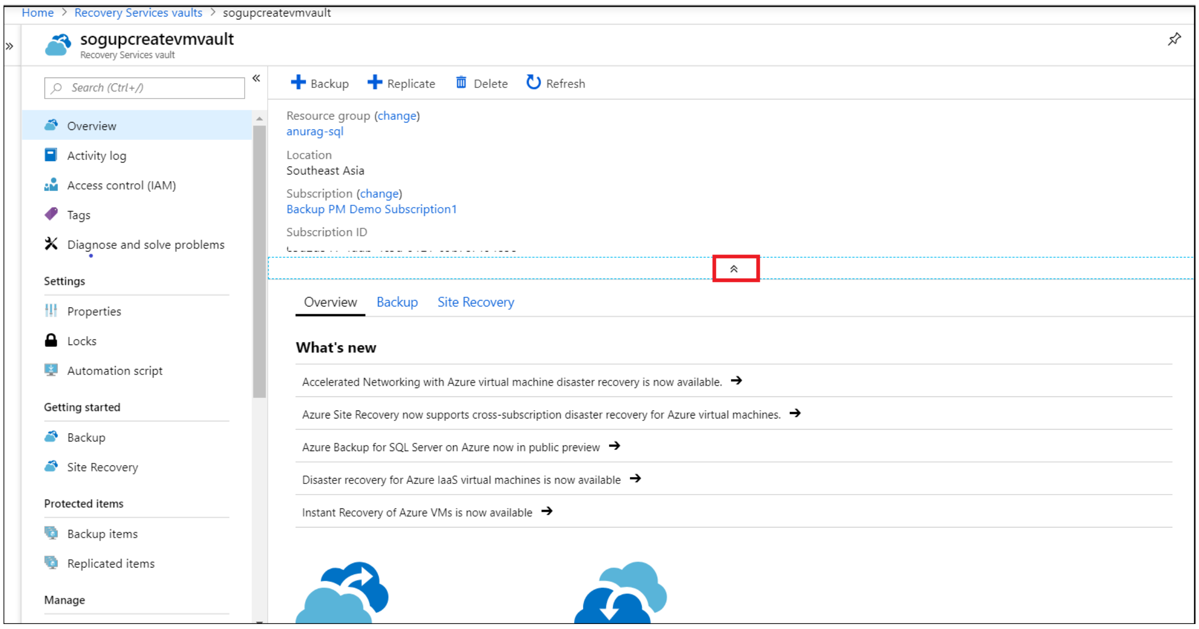
Azure Site Recovery now supports cross-subscription disaster recovery for Azure virtual machines.

Azure Backup for SQL Server on Azure now in public preview

Disaster recovery for Azure IaaS virtual machines is now available

Instant Recovery of Azure VMs is now available

If you do not see the **Essentials** information for your vault, click the drop-down icon. You should now see the Essentials information for your vault.



Resource group (change)
anurag-sql

Location
Southeast Asia

Subscription (change)
Backup PM Demo Subscription1

Subscription ID

What's new

Accelerated Networking with Azure virtual machine disaster recovery is now available.

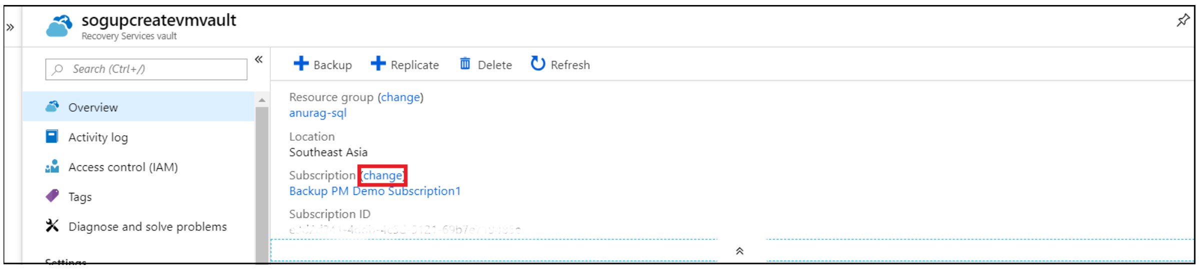
Azure Site Recovery now supports cross-subscription disaster recovery for Azure virtual machines.

Azure Backup for SQL Server on Azure now in public preview

Disaster recovery for Azure IaaS virtual machines is now available

Instant Recovery of Azure VMs is now available

3. In the vault overview menu, click **change** next to **Subscription**, to open the **Move resources** blade.



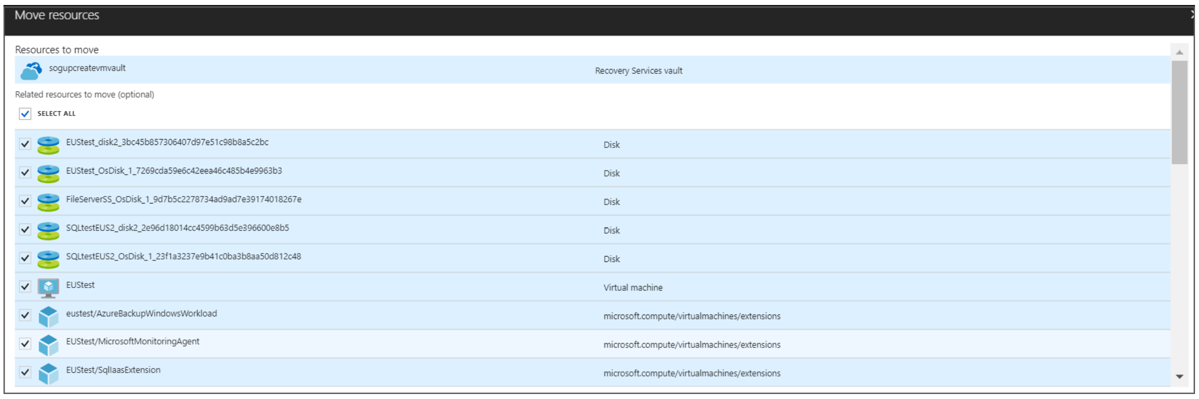
Resource group (change)
anurag-sql

Location
Southeast Asia

Subscription (change)
Backup PM Demo Subscription1

Subscription ID

4. Select the resources to be moved, here we recommend you to use the **Select All** option to select all the listed optional resources.


Move resources

Resources to move

sogupcreatevmvault	Recovery Services vault

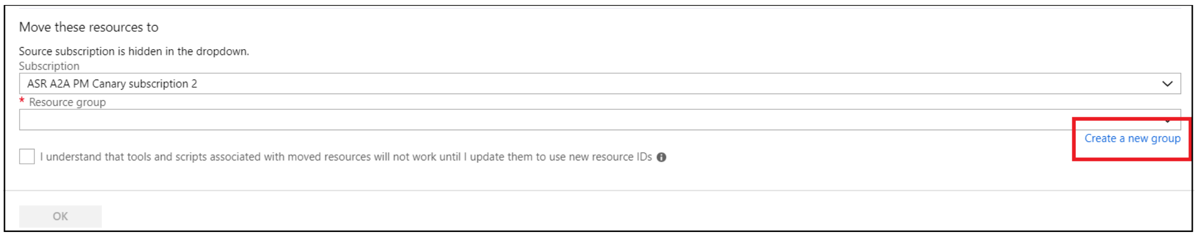
Related resources to move (optional)

✓ SELECT ALL

✓	EUStest_disk2_3bc45b857306407d97e51c98b8a5c2bc	Disk
✓	EUStest_OsDisk_1_7269cda59e6c42eea46c485b4e9963b3	Disk
✓	FileServerSS_OsDisk_1_9d7b5c2278734ad9ad7e39174018267e	Disk
✓	SQLtestEUS2_disk2_2e96d18014cc4599b63d5e396600e8b5	Disk
✓	SQLtestEUS2_OsDisk_1_23f1a3237e9b41c0ba3b8aa50d812c48	Disk
✓	EUStest	Virtual machine
✓	eustest/AzureBackupWindowsWorkload	microsoft.compute/virtualmachines/extensions
✓	EUStest/MicrosoftMonitoringAgent	microsoft.compute/virtualmachines/extensions
✓	EUStest/SqlIaasExtension	microsoft.compute/virtualmachines/extensions

5. Select the target subscription from the **Subscription** drop-down list, where you want the vault to be moved.

6. To add the target resource group, in the **Resource group** drop-down list select an existing resource group or click **create a new group** option.


Move these resources to

Source subscription is hidden in the dropdown.

Subscription: ASR A2A PM Canary subscription 2

* Resource group

Create a new group

☐ I understand that tools and scripts associated with moved resources will not work until I update them to use new resource IDs

7. Click **I understand that tools and scripts associated with moved resources will not work until I update them to use new resource IDs** option to confirm, and then click **OK**.

NOTE

Cross subscription backup (RS vault and protected VMs are in different subscriptions) is not a supported scenario. Also, storage redundancy option from local redundant storage (LRS) to global redundant storage (GRS) and vice versa cannot be modified during the vault move operation.

Use PowerShell to move Recovery Services vault

To move a Recovery Services vault to another resource group, use the `Move-AzureRMResource` cmdlet. `Move-AzureRMResource` requires the resource name and type of resource. You can get both from the `Get-AzureRmRecoveryServicesVault` cmdlet.

```
$destinationRG = "<destinationResourceGroupName>"
$vault = Get-AzureRmRecoveryServicesVault -Name <vaultname> -ResourceGroupName <vaultRGname>
Move-AzureRmResource -DestinationResourceGroupName $destinationRG -ResourceId $vault.ID
```

To move the resources to different subscription, include the `-DestinationSubscriptionId` parameter.

```
Move-AzureRmResource -DestinationSubscriptionId "<destinationSubscriptionID>" -DestinationResourceGroupName
$destinationRG -ResourceId $vault.ID
```

After executing the above cmdlets, you will be asked to confirm that you want to move the specified resources. Type **Y** to confirm. After a successful validation, the resource moves.

Use CLI to move Recovery Services vault

To move a Recovery Services vault to another resource group, use the following cmdlet:

```
az resource move --destination-group <destinationResourceGroupName> --ids <VaultResourceID>
```

To move to a new subscription, provide the `--destination-subscription-id` parameter.

Post migration

1. You need to set/verify the access controls for the resource groups.
2. The Backup reporting and monitoring feature needs to be configured again for the vault post the move completes. The previous configuration will be lost during the move operation.

Next steps

You can move many different types of resources between resource groups and subscriptions.

For more information, see Move resources to new resource group or subscription.

Move guidance for virtual machines

7/10/2019 • 2 minutes to read • Edit Online

This article describes the scenarios that aren't currently supported and the steps to move virtual machines with backup.

Scenarios not supported

The following scenarios aren't yet supported:

- Managed Disks in Availability Zones can't be moved to a different subscription.
- Virtual Machines with certificate stored in Key Vault can be moved to a new resource group in the same subscription, but not across subscriptions.
- Virtual Machine Scale Sets with Standard SKU Load Balancer or Standard SKU Public IP can't be moved.
- Virtual machines created from Marketplace resources with plans attached can't be moved across resource groups or subscriptions. Deprovision the virtual machine in the current subscription, and deploy again in the new subscription.
- Virtual machines in an existing virtual network but you aren't moving all resources in the virtual network.

Virtual machines with Azure Backup

To move virtual machines configured with Azure Backup, use the following workaround:

- Find the location of your Virtual Machine.
- Find a resource group with the following naming pattern: `AzureBackupRG_<location of your VM>_1` for example, AzureBackupRG_westus2_1
- If in Azure portal, then check "Show hidden types"
- If in PowerShell, use the `Get-AzResource -ResourceGroupName AzureBackupRG_<location of your VM>_1` cmdlet
- If in CLI, use the `az resource list -g AzureBackupRG_<location of your VM>_1`
- Find the resource with type `Microsoft.Compute/restorePointCollections` that has the naming pattern `AzureBackup_<name of your VM that you're trying to move>_###########`
- Delete this resource. This operation deletes only the instant recovery points, not the backed-up data in the vault.
- After delete is complete, you can move the vault and virtual machine to the target subscription. After the move, you can continue backups with no loss in data.
- For information about moving Recovery Service vaults for backup, see Recovery Services limitations.

Next steps

For commands to move resources, see Move resources to new resource group or subscription.

Move guidance for virtual networks

7/10/2019 • 2 minutes to read • Edit Online

This article describes how to move virtual networks for specific scenarios.

Dependent resources

When moving a virtual network, you must also move its dependent resources. For VPN Gateways, you must move IP addresses, virtual network gateways, and all associated connection resources. Local network gateways can be in a different resource group.

To move a virtual machine with a network interface card, you must move all dependent resources. Move the virtual network for the network interface card, all other network interface cards for the virtual network, and the VPN gateways.

Peered virtual network

To move a peered virtual network, you must first disable the virtual network peering. Once disabled, you can move the virtual network. After the move, reenable the virtual network peering.

Subnet links

You can't move a virtual network to a different subscription if the virtual network contains a subnet with resource navigation links. For example, if an Azure Cache for Redis resource is deployed into a subnet, that subnet has a resource navigation link.

Next steps

For commands to move resources, see Move resources to new resource group or subscription.

Use tags to organize your Azure resources

7/18/2019 • 11 minutes to read • Edit Online

You apply tags to your Azure resources giving metadata to logically organize them into a taxonomy. Each tag consists of a name and a value pair. For example, you can apply the name "Environment" and the value "Production" to all the resources in production.

After you apply tags, you can retrieve all the resources in your subscription with that tag name and value. Tags enable you to retrieve related resources from different resource groups. This approach is helpful when you need to organize resources for billing or management.

Your taxonomy should consider a self-service metadata tagging strategy in addition to an auto-tagging strategy to reduce the burden on users and increase accuracy.

The following limitations apply to tags:

- Not all resource types support tags. To determine if you can apply a tag to a resource type, see Tag support for Azure resources.
- Each resource or resource group can have a maximum of 50 tag name/value pairs. Currently, storage accounts only support 15 tags, but that limit will be raised to 50 in a future release. If you need to apply more tags than the maximum allowed number, use a JSON string for the tag value. The JSON string can contain many values that are applied to a single tag name. A resource group can contain many resources that each have 50 tag name/value pairs.
- The tag name is limited to 512 characters, and the tag value is limited to 256 characters. For storage accounts, the tag name is limited to 128 characters, and the tag value is limited to 256 characters.
- Generalized VMs don't support tags.
- Tags applied to the resource group are not inherited by the resources in that resource group.
- Tags can't be applied to classic resources such as Cloud Services.
- Tag names can't contain these characters: `<`, `>`, `%`, `&`, `\`, `?`, `/`

To apply tags to resources, the user must have write access to that resource type. To apply tags to all resource types, use the Contributor role. To apply tags to only one resource type, use the contributor role for that resource. For example, to apply tags to virtual machines, use the Virtual Machine Contributor.

NOTE

This article provides steps for how to delete personal data from the device or service and can be used to support your obligations under the GDPR. If you're looking for general info about GDPR, see the GDPR section of the Service Trust portal.

Policies

You can use Azure Policy to enforce tagging rules and conventions. By creating a policy, you avoid the scenario of resources being deployed to your subscription that don't comply with the expected tags for your organization. Instead of manually applying tags or searching for resources that aren't compliant, you can create a policy that automatically applies the needed tags during deployment. The following section shows example policies for tags.

Tags

Apply tag and its default value	Appends a specified tag name and value, if that tag is not provided. You specify the tag name and value to apply.
Billing Tags Policy Initiative	Requires specified tag values for cost center and product name. Uses built-in policies to apply and enforce required tags. You specify the required values for the tags.
Enforce tag and its value	Requires a specified tag name and value. You specify the tag name and value to enforce.
Enforce tag and its value on resource groups	Requires a tag and value on a resource group. You specify the required tag name and value.

PowerShell

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

To see the existing tags for a *resource group*, use:

```
(Get-AzResourceGroup -Name examplegroup).Tags
```

That script returns the following format:

```
Name                           Value
----                           -----
Dept                           IT
Environment                    Test
```

To see the existing tags for a *resource that has a specified resource ID*, use:

```
(Get-AzResource -ResourceId /subscriptions/<subscription-id>/resourceGroups/<rg-
name>/providers/Microsoft.Storage/storageAccounts/<storage-name>).Tags
```

Or, to see the existing tags for a *resource that has a specified name and resource group*, use:

```
(Get-AzResource -ResourceName examplevnet -ResourceGroupName examplegroup).Tags
```

To get *resource groups that have a specific tag*, use:

```
(Get-AzResourceGroup -Tag @{ Dept="Finance" }).ResourceGroupName
```

To get *resources that have a specific tag*, use:

```
(Get-AzResource -Tag @{ Dept="Finance"}).Name
```

To get *resources that have a specific tag name*, use:

```
(Get-AzResource -TagName Dept).Name
```

Every time you apply tags to a resource or a resource group, you overwrite the existing tags on that resource or resource group. Therefore, you must use a different approach based on whether the resource or resource group has existing tags.

To add tags to a *resource group without existing tags*, use:

```
Set-AzResourceGroup -Name examplegroup -Tag @{ Dept="IT"; Environment="Test" }
```

To add tags to a *resource group that has existing tags*, retrieve the existing tags, add the new tag, and reapply the tags:

```
$tags = (Get-AzResourceGroup -Name examplegroup).Tags
$tags.Add("Status", "Approved")
Set-AzResourceGroup -Tag $tags -Name examplegroup
```

To add tags to a *resource without existing tags*, use:

```
$r = Get-AzResource -ResourceName examplevnet -ResourceGroupName examplegroup
Set-AzResource -Tag @{ Dept="IT"; Environment="Test" } -ResourceId $r.ResourceId -Force
```

To add tags to a *resource that has existing tags*, use:

```
$r = Get-AzResource -ResourceName examplevnet -ResourceGroupName examplegroup
$r.Tags.Add("Status", "Approved")
Set-AzResource -Tag $r.Tags -ResourceId $r.ResourceId -Force
```

To apply all tags from a resource group to its resources, and *not keep existing tags on the resources*, use the following script:

```
$groups = Get-AzResourceGroup
foreach ($g in $groups)
{
    Get-AzResource -ResourceGroupName $g.ResourceGroupName | ForEach-Object {Set-AzResource -ResourceId
$_.ResourceId -Tag $g.Tags -Force }
}
```

To apply all tags from a resource group to its resources, and *keep existing tags on resources that aren't duplicates*, use the following script:

```
$group = Get-AzResourceGroup "examplegroup"
if ($null -ne $group.Tags) {
    $resources = Get-AzResource -ResourceGroupName $group.ResourceGroupName
    foreach ($r in $resources)
    {
        $resourcetags = (Get-AzResource -ResourceId $r.ResourceId).Tags
        if ($resourcetags)
        {
            foreach ($key in $group.Tags.Keys)
            {
                if (-not($resourcetags.ContainsKey($key)))
                {
                    $resourcetags.Add($key, $group.Tags[$key])
                }
            }
            Set-AzResource -Tag $resourcetags -ResourceId $r.ResourceId -Force
        }
        else
        {
            Set-AzResource -Tag $group.Tags -ResourceId $r.ResourceId -Force
        }
    }
}
```

To remove all tags, pass an empty hash table:

```
Set-AzResourceGroup -Tag @{} -Name examplegroup
```

Azure CLI

To see the existing tags for a *resource group*, use:

```
az group show -n examplegroup --query tags
```

That script returns the following format:

```
{
  "Dept"        : "IT",
  "Environment" : "Test"
}
```

Or, to see the existing tags for a *resource that has a specified name, type, and resource group*, use:

```
az resource show -n examplevnet -g examplegroup --resource-type "Microsoft.Network/virtualNetworks" --query
tags
```

When looping through a collection of resources, you might want to show the resource by resource ID. A complete example is shown later in this article. To see the existing tags for a *resource that has a specified resource ID*, use:

```
az resource show --id <resource-id> --query tags
```

To get resource groups that have a specific tag, use `az group list`:

```
az group list --tag Dept=IT
```

To get all the resources that have a particular tag and value, use `az resource list`:

```
az resource list --tag Dept=Finance
```

Every time you apply tags to a resource or a resource group, you overwrite the existing tags on that resource or resource group. Therefore, you must use a different approach based on whether the resource or resource group has existing tags.

To add tags to a *resource group without existing tags*, use:

```
az group update -n examplegroup --set tags.Environment=Test tags.Dept=IT
```

To add tags to a *resource without existing tags*, use:

```
az resource tag --tags Dept=IT Environment=Test -g examplegroup -n examplevnet --resource-type
"Microsoft.Network/virtualNetworks"
```

To add tags to a resource that already has tags, retrieve the existing tags, reformat that value, and reapply the existing and new tags:

```
jsonrtag=$(az resource show -g examplegroup -n examplevnet --resource-type "Microsoft.Network/virtualNetworks"
--query tags)
rt=$(echo $jsonrtag | tr -d '"{},' | sed 's/: /=/g')
az resource tag --tags $rt Project=Redesign -g examplegroup -n examplevnet --resource-type
"Microsoft.Network/virtualNetworks"
```

To apply all tags from a resource group to its resources, and *not keep existing tags on the resources*, use the following script:

```
groups=$(az group list --query [].name --output tsv)
for rg in $groups
do
  jsontag=$(az group show -n $rg --query tags)
  t=$(echo $jsontag | tr -d '"{},' | sed 's/: /=/g')
  r=$(az resource list -g $rg --query [].id --output tsv)
  for resid in $r
  do
    az resource tag --tags $t --id $resid
  done
done
```

To apply all tags from a resource group to its resources, and *keep existing tags on resources*, use the following script:

```
groups=$(az group list --query [].name --output tsv)
for rg in $groups
do
  jsontag=$(az group show -n $rg --query tags)
  t=$(echo $jsontag | tr -d '"{},' | sed 's/: /=/g')
  r=$(az resource list -g $rg --query [].id --output tsv)
  for resid in $r
  do
    jsonrtag=$(az resource show --id $resid --query tags)
    rt=$(echo $jsonrtag | tr -d '"{},' | sed 's/: /=/g')
    az resource tag --tags $t$rt --id $resid
  done
done
```

Templates

To tag a resource during deployment, add the `tags` element to the resource you're deploying. Provide the tag name and value.

Apply a literal value to the tag name

The following example shows a storage account with two tags (`Dept` and `Environment`) that are set to literal values:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "location": {
            "type": "string",
            "defaultValue": "[resourceGroup().location]"
        }
    },
    "resources": [
        {
            "apiVersion": "2019-04-01",
            "type": "Microsoft.Storage/storageAccounts",
            "name": "[concat('storage', uniqueString(resourceGroup().id))]",
            "location": "[parameters('location')]",
            "tags": {
                "Dept": "Finance",
                "Environment": "Production"
            },
            "sku": {
                "name": "Standard_LRS"
            },
            "kind": "Storage",
            "properties": {}
        }
    ]
}
```

To set a tag to a datetime value, use the utcNow function.

Apply an object to the tag element

You can define an object parameter that stores several tags, and apply that object to the tag element. Each property in the object becomes a separate tag for the resource. The following example has a parameter named `tagValues` that is applied to the tag element.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "location": {
            "type": "string",
            "defaultValue": "[resourceGroup().location]"
        },
        "tagValues": {
            "type": "object",
            "defaultValue": {
                "Dept": "Finance",
                "Environment": "Production"
            }
        }
    },
    "resources": [
        {
            "apiVersion": "2019-04-01",
            "type": "Microsoft.Storage/storageAccounts",
            "name": "[concat('storage', uniqueString(resourceGroup().id))]",
            "location": "[parameters('location')]",
            "tags": "[parameters('tagValues')]",
            "sku": {
                "name": "Standard_LRS"
            },
            "kind": "Storage",
            "properties": {}
        }
    ]
}
```

Apply a JSON string to the tag name

To store many values in a single tag, apply a JSON string that represents the values. The entire JSON string is stored as one tag that can't exceed 256 characters. The following example has a single tag named `CostCenter` that contains several values from a JSON string:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "location": {
            "type": "string",
            "defaultValue": "[resourceGroup().location]"
        }
    },
    "resources": [
        {
            "apiVersion": "2019-04-01",
            "type": "Microsoft.Storage/storageAccounts",
            "name": "[concat('storage', uniqueString(resourceGroup().id))]",
            "location": "[parameters('location')]",
            "tags": {
                "CostCenter": "{\"Dept\":\"Finance\",\"Environment\":\"Production\"}"
            },
            "sku": {
                "name": "Standard_LRS"
            },
            "kind": "Storage",
            "properties": {}
        }
    ]
}
```

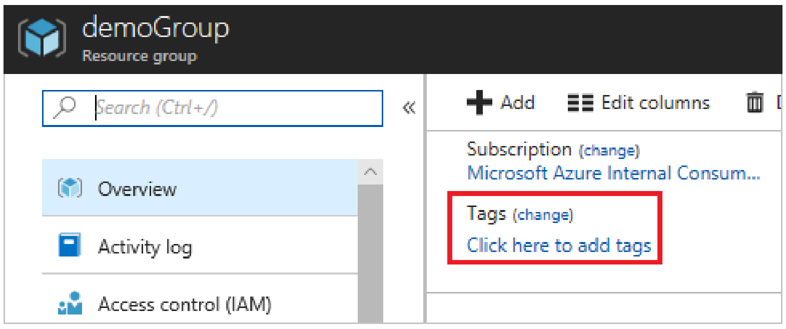
Apply tags from resource group

To apply tags from a resource group to a resource, use the resourceGroup function. When getting the tag value, use the `tags.[tag-name]` syntax instead of the `tags.tag-name` syntax, because some characters aren't parsed correctly in the dot notation.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "location": {
            "type": "string",
            "defaultValue": "[resourceGroup().location]"
        }
    },
    "resources": [
        {
            "apiVersion": "2019-04-01",
            "type": "Microsoft.Storage/storageAccounts",
            "name": "[concat('storage', uniqueString(resourceGroup().id))]",
            "location": "[parameters('location')]",
            "tags": {
                "Dept": "[resourceGroup().tags['Dept']]",
                "Environment": "[resourceGroup().tags['Environment']]"
            },
            "sku": {
                "name": "Standard_LRS"
            },
            "kind": "Storage",
            "properties": {}
        }
    ]
}
```

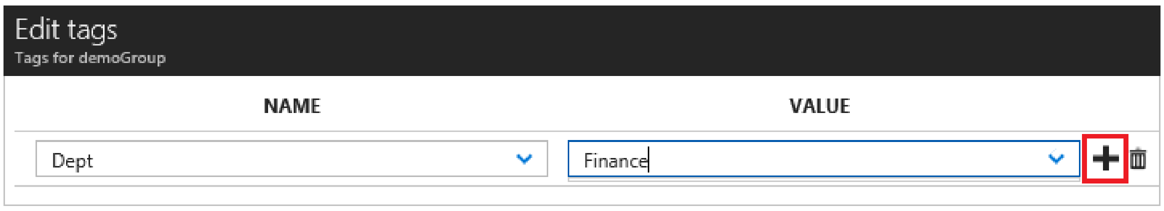
Portal

1. To view the tags for a resource or a resource group, looks for existing tags in the overview. If you have not previously applied tags, the list is empty.

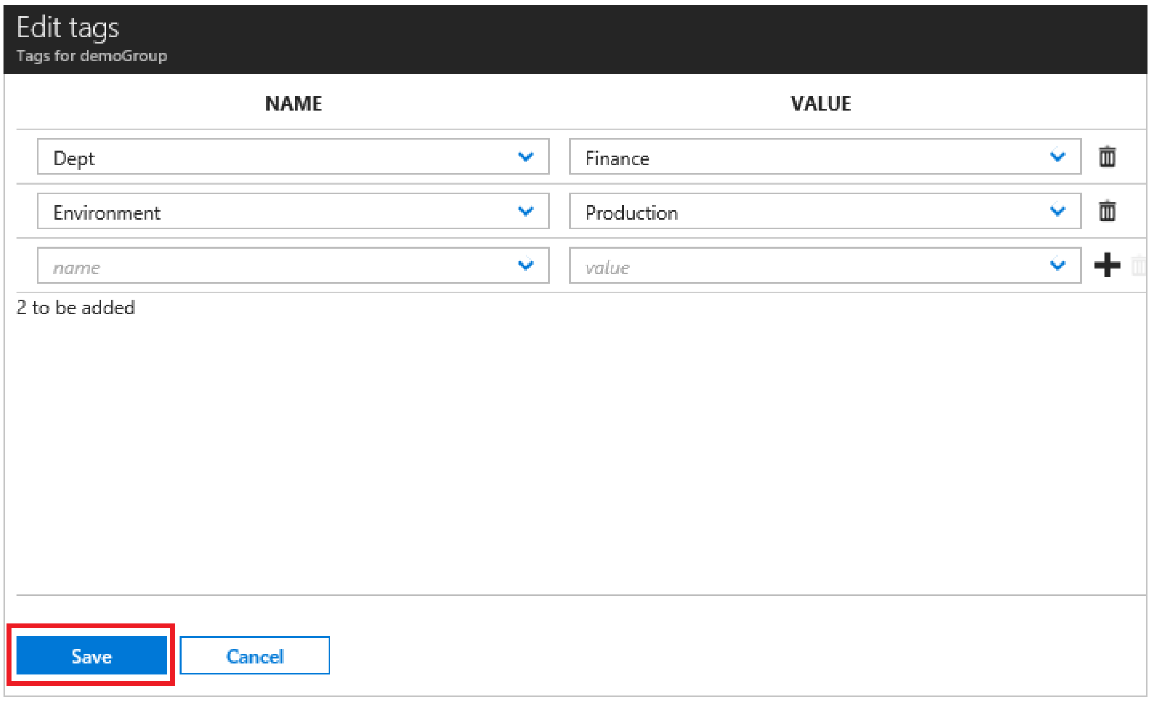


2. To add a tag, select **Click here to add tags**.

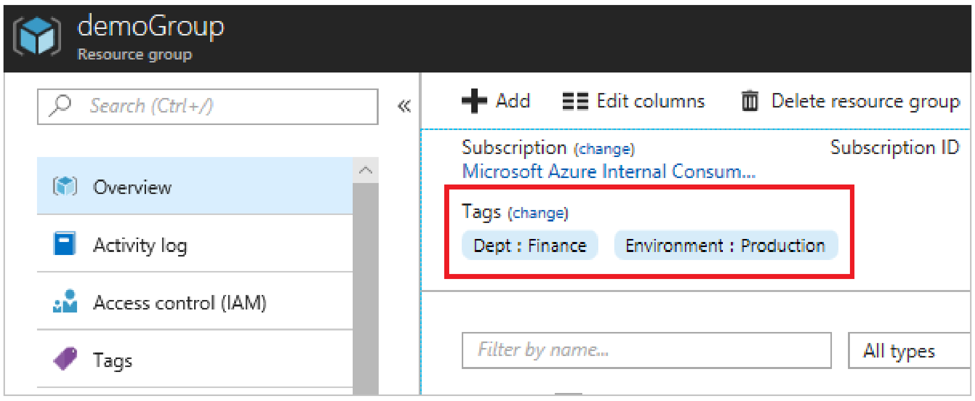
3. Provide a name and value. Select **+** to add the tag.



4. Continue adding tags as needed. When done, select **Save**.

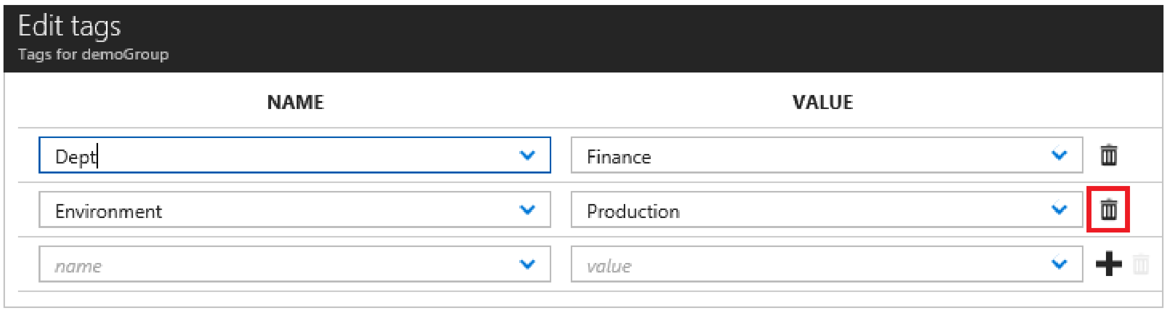


5. The tags are now displayed in the overview.



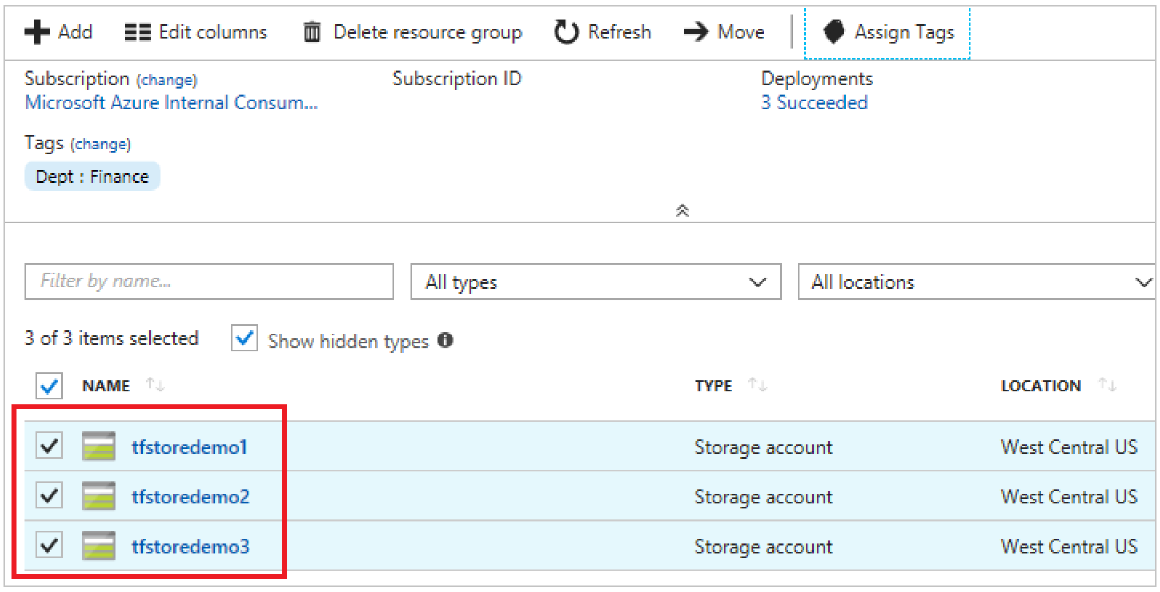
6. To add or delete a tag, select **change**.

7. To delete a tag, select the trash icon. Then, select **Save**.



To bulk assign tags to multiple resources:

1. From any list of resources, select the checkbox for the resources you want to assign the tag.

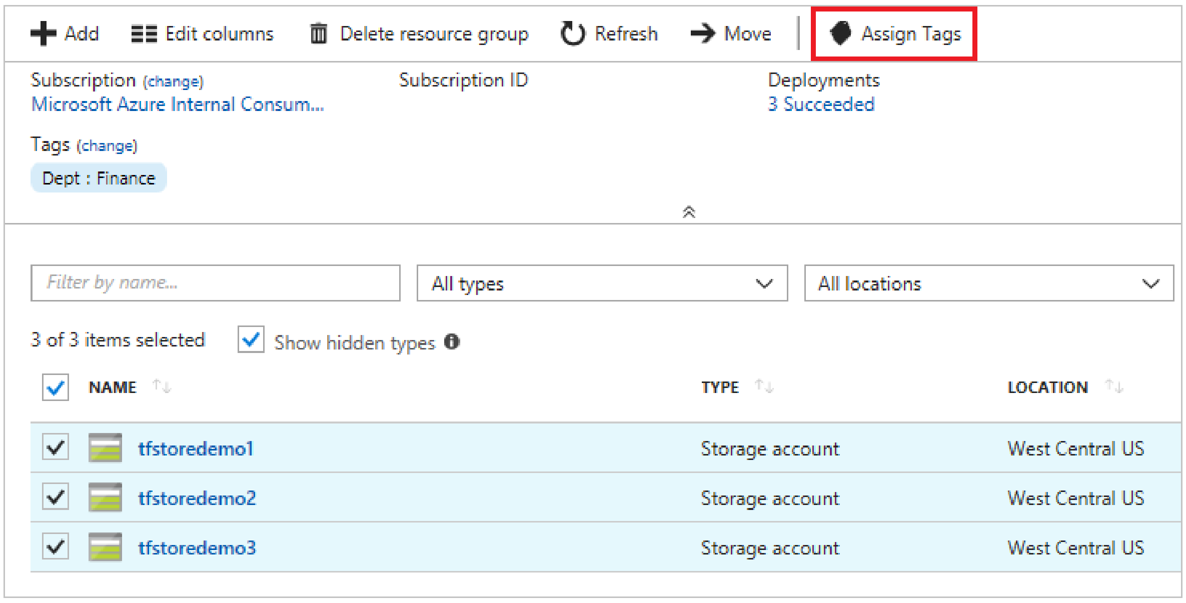


Subscription (change)
Microsoft Azure Internal Consum...
Subscription ID
Deployments
3 Succeeded
Tags (change)
Dept : Finance

Filter by name...
All types
All locations
3 of 3 items selected ✓ Show hidden types

	NAME	TYPE	LOCATION
✓	tfstoredemo1	Storage account	West Central US
✓	tfstoredemo2	Storage account	West Central US
✓	tfstoredemo3	Storage account	West Central US

2. Select **Assign tags**

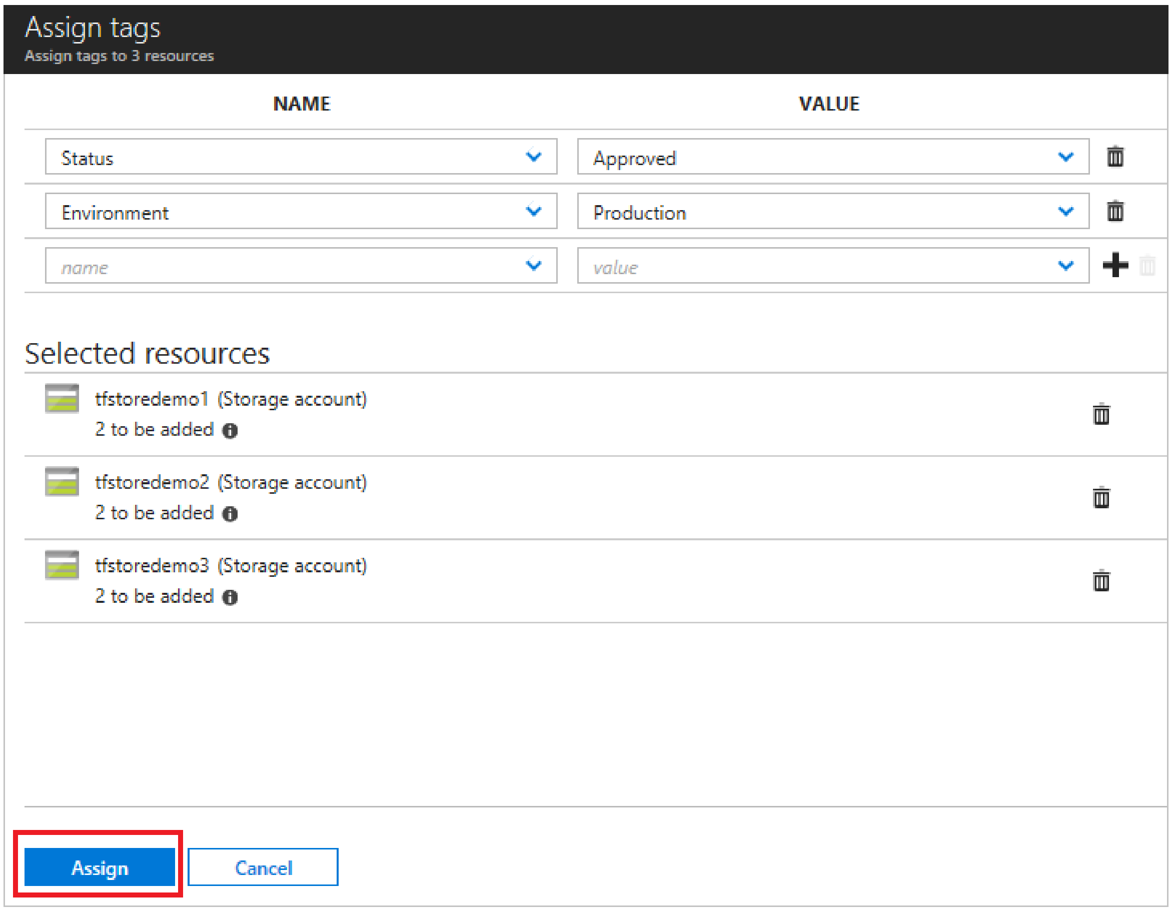


Subscription (change)
Microsoft Azure Internal Consum...
Subscription ID
Deployments
3 Succeeded
Tags (change)
Dept : Finance

Filter by name...
All types
All locations
3 of 3 items selected ✓ Show hidden types

	NAME	TYPE	LOCATION
✓	tfstoredemo1	Storage account	West Central US
✓	tfstoredemo2	Storage account	West Central US
✓	tfstoredemo3	Storage account	West Central US

3. After each name and value, select **+**. When done, select **Assign**.


Assign tags
Assign tags to 3 resources

NAME	VALUE
Status	Approved
Environment	Production
name	value

Selected resources

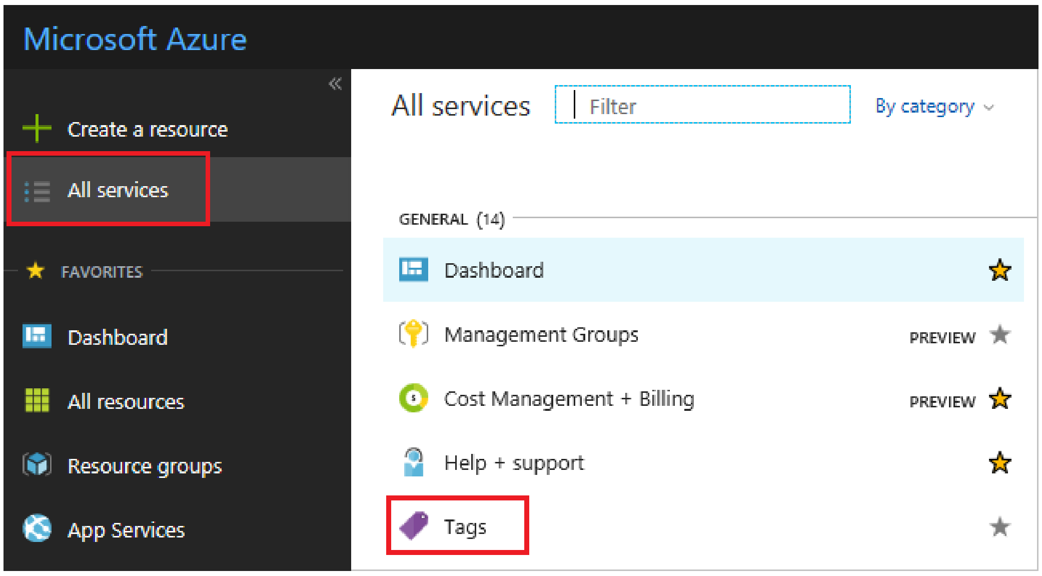
tfstoredemo1 (Storage account)
2 to be added

tfstoredemo2 (Storage account)
2 to be added

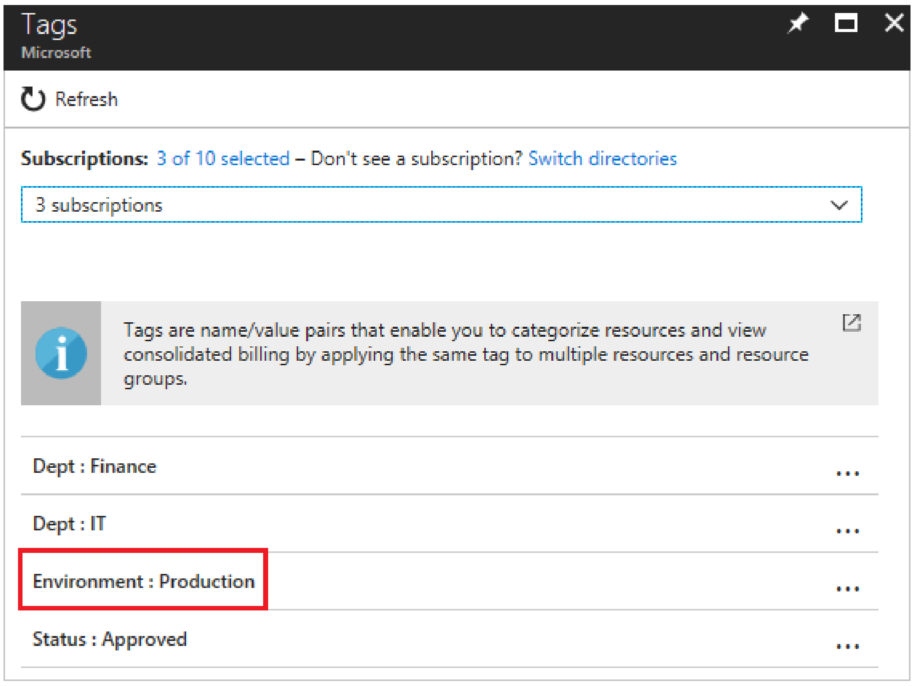
tfstoredemo3 (Storage account)
2 to be added

To view all resources with a tag:

1. Select **All services** and **Tags**.



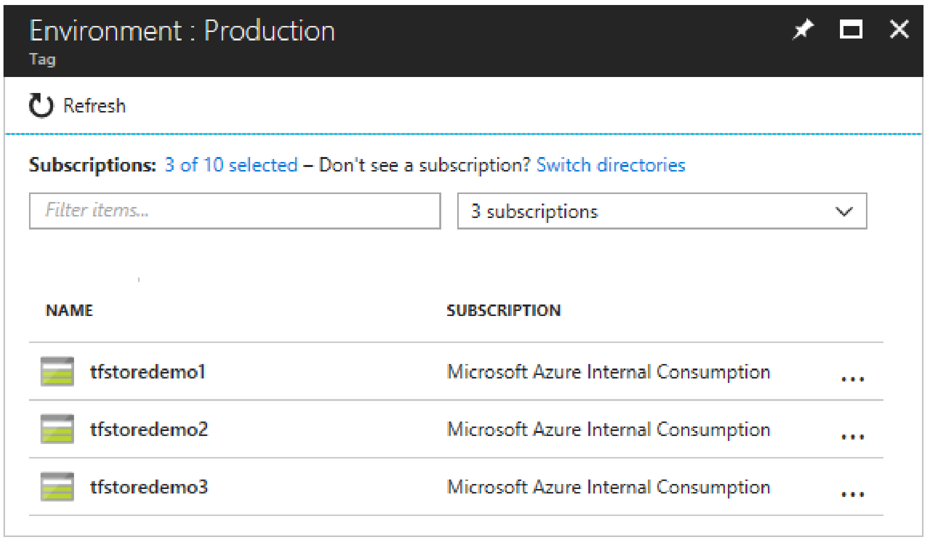
2. Select the tag for viewing resources.


Tags
Microsoft
Refresh
Subscriptions: 3 of 10 selected – Don't see a subscription? Switch directories
3 subscriptions

Tags are name/value pairs that enable you to categorize resources and view consolidated billing by applying the same tag to multiple resources and resource groups.

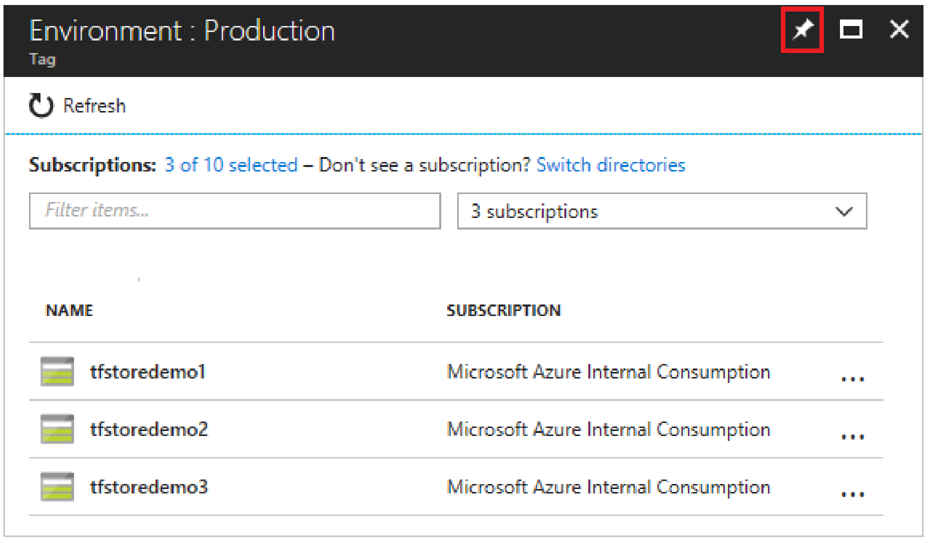
Dept : Finance
Dept : IT
Environment : Production
Status : Approved

3. All resources with that tag are displayed.


Environment : Production
Tag
Refresh
Subscriptions: 3 of 10 selected – Don't see a subscription? Switch directories
Filter items...
3 subscriptions

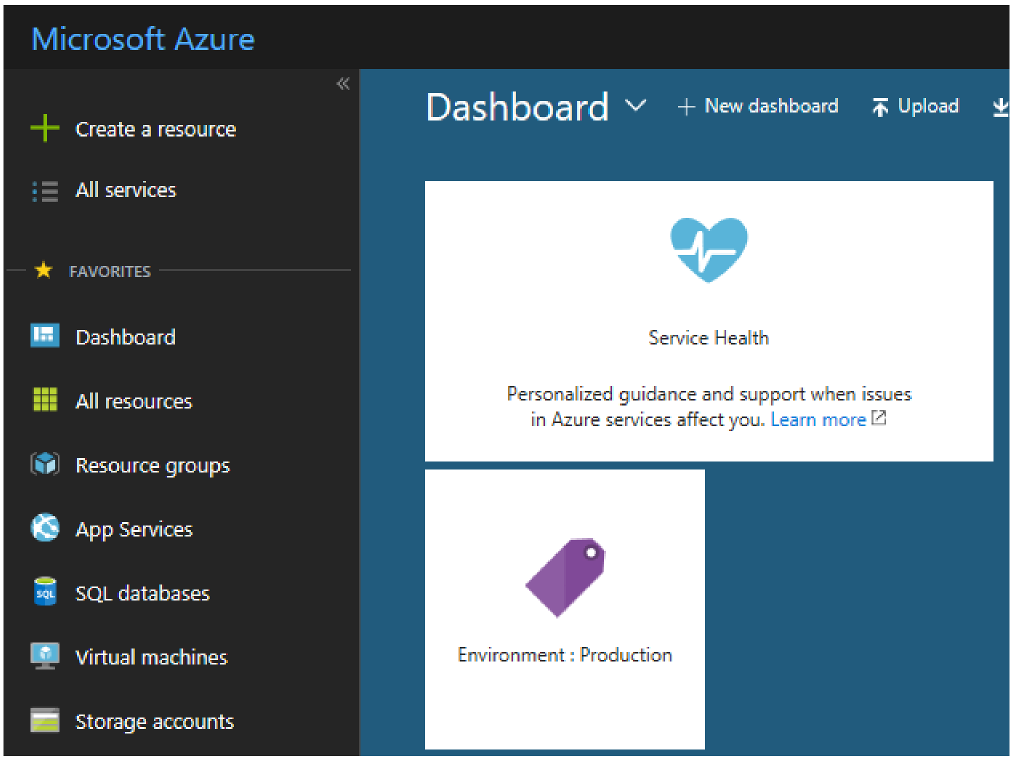
NAME	SUBSCRIPTION
tfstoredemo1	Microsoft Azure Internal Consumption
tfstoredemo2	Microsoft Azure Internal Consumption
tfstoredemo3	Microsoft Azure Internal Consumption

4. For quick access, pin the view to the dashboard.


Environment : Production
Tag
Refresh
Subscriptions: 3 of 10 selected – Don't see a subscription? Switch directories
Filter items...
3 subscriptions

NAME	SUBSCRIPTION
tfstoredemo1	Microsoft Azure Internal Consumption
tfstoredemo2	Microsoft Azure Internal Consumption
tfstoredemo3	Microsoft Azure Internal Consumption

5. The view is available from the dashboard.



REST API

The Azure portal and PowerShell both use the Resource Manager REST API behind the scenes. If you need to integrate tagging into another environment, you can get tags by using **GET** on the resource ID and update the set of tags by using a **PATCH** call.

Tags and billing

You can use tags to group your billing data. For example, if you're running multiple VMs for different organizations, use the tags to group usage by cost center. You can also use tags to categorize costs by runtime environment, such as the billing usage for VMs running in the production environment.

You can retrieve information about tags through the Azure Resource Usage and RateCard APIs or the usage comma-separated values (CSV) file. You download the usage file from the Azure Account Center or Azure portal. For more information, see Download or view your Azure billing invoice and daily usage data. When downloading the usage file from the Azure Account Center, select **Version 2**. For services that support tags with billing, the tags appear in the **Tags** column.

For REST API operations, see Azure Billing REST API Reference.

Next steps

- Not all resource types support tags. To determine if you can apply a tag to a resource type, see Tag support for Azure resources.
- For an introduction to using the portal, see Using the Azure portal to manage your Azure resources.

Tag support for Azure resources

7/17/2019 • 17 minutes to read • Edit Online

This article describes whether a resource type supports tags. The column labeled **Supports tags** indicates whether the resource type has a property for the tag. The column labeled **Tag in cost report** indicates whether that resource type passes the tag to the cost report.

To get the same data as a file of comma-separated values, download tag-support.csv.

Microsoft.AAD

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
DomainServices	Yes	Yes
DomainServices/oucontainer	No	No

microsoft.aadiam

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
diagnosticSettings	No	No
diagnosticSettingsCategories	No	No

Microsoft.Addons

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
supportProviders	No	No

Microsoft.ADHybridHealthService

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
aadsupportcases	No	No
addsservices	No	No
agents	No	No
anonymousapiusers	No	No
configuration	No	No
logs	No	No

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
reports	No	No
services	No	No

Microsoft.Advisor

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
configurations	No	No
generateRecommendations	No	No
recommendations	No	No
suppressions	No	No

Microsoft.AlertsManagement

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
actionRules	No	No
alerts	No	No
alertsList	No	No
alertsSummary	No	No
alertsSummaryList	No	No
smartDetectorAlertRules	No	No
smartDetectorRuntimeEnvironments	No	No
smartGroups	No	No

Microsoft.AnalysisServices

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
servers	Yes	Yes

Microsoft.ApiManagement

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
reportFeedback	No	No

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
service	Yes	Yes
validateServiceName	No	No

Microsoft.Attestation

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
attestationProviders	No	No

Microsoft.Authorization

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
classicAdministrators	No	No
denyAssignments	No	No
elevateAccess	No	No
locks	No	No
permissions	No	No
policyAssignments	No	No
policyDefinitions	No	No
policySetDefinitions	No	No
providerOperations	No	No
roleAssignments	No	No
roleDefinitions	No	No

Microsoft.Automation

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
automationAccounts	Yes	Yes
automationAccounts/configurations	Yes	Yes
automationAccounts/jobs	No	No
automationAccounts/runbooks	Yes	Yes

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
automationAccounts/softwareUpdateConfigurations	No	No
automationAccounts/webhooks	No	No

Microsoft.Azure.Geneva

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
environments	No	No
environments/accounts	No	No
environments/accounts/namespaces	No	No
environments/accounts/namespaces/configurations	No	No

Microsoft.AzureActiveDirectory

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
b2cDirectories	Yes	No

Microsoft.AzureStack

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
registrations	Yes	Yes
registrations/customerSubscriptions	No	No
registrations/products	No	No

Microsoft.Batch

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
batchAccounts	Yes	Yes

Microsoft.Billing

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
billingAccounts	No	No
billingAccounts/billingProfiles	No	No

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
billingAccounts/billingProfiles/billingSubscriptions	No	No
billingAccounts/billingProfiles/invoices	No	No
billingAccounts/billingProfiles/invoices/pricesheet	No	No
billingAccounts/billingProfiles/operationStatus	No	No
billingAccounts/billingProfiles/paymentMethods	No	No
billingAccounts/billingProfiles/policies	No	No
billingAccounts/billingProfiles/pricesheet	No	No
billingAccounts/billingProfiles/products	No	No
billingAccounts/billingProfiles/transactions	No	No
billingAccounts/billingSubscriptions	No	No
billingAccounts/departments	No	No
billingAccounts/eligibleOffers	No	No
billingAccounts/enrollmentAccounts	No	No
billingAccounts/invoices	No	No
billingAccounts/invoiceSections	No	No
billingAccounts/invoiceSections/billingSubscriptions	No	No
billingAccounts/invoiceSections/billingSubscriptions/transfer	No	No
billingAccounts/invoiceSections/importRequests	No	No
billingAccounts/invoiceSections/initiateImportRequest	No	No
billingAccounts/invoiceSections/initiateTransfer	No	No
billingAccounts/invoiceSections/operationStatus	No	No

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
billingAccounts/invoiceSections/products	No	No
billingAccounts/invoiceSections/transfers	No	No
billingAccounts/products	No	No
billingAccounts/projects	No	No
billingAccounts/projects/billingSubscriptions	No	No
billingAccounts/projects/importRequests	No	No
billingAccounts/projects/initiateImportRequest	No	No
billingAccounts/projects/operationStatus	No	No
billingAccounts/projects/products	No	No
billingAccounts/transactions	No	No
billingPeriods	No	No
BillingPermissions	No	No
billingProperty	No	No
BillingRoleAssignments	No	No
BillingRoleDefinitions	No	No
CreateBillingRoleAssignment	No	No
departments	No	No
enrollmentAccounts	No	No
importRequests	No	No
importRequests/acceptImportRequest	No	No
importRequests/declineImportRequest	No	No
invoices	No	No
transfers	No	No

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
transfers/acceptTransfer	No	No
transfers/declineTransfer	No	No
transfers/operationStatus	No	No
usagePlans	No	No

Microsoft.BingMaps

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
mapApis	Yes	Yes
updateCommunicationPreference	No	No

Microsoft.BizTalkServices

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
BizTalk	Yes	Yes

Microsoft.Blueprint

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
blueprintAssignments	No	No
blueprintAssignments/assignmentOperations	No	No
blueprintAssignments/operations	No	No
blueprints	No	No
blueprints/artifacts	No	No
blueprints/versions	No	No
blueprints/versions/artifacts	No	No

Microsoft.BotService

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
botServices	Yes	Yes
botServices/channels	No	No

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
botServices/connections	No	No

Microsoft.Cache

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
Redis	Yes	Yes
RedisConfigDefinition	No	No

Microsoft.Capacity

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
appliedReservations	No	No
calculatePrice	No	No
catalogs	No	No
commercialReservationOrders	No	No
reservationOrders	No	No
reservationOrders/calculateRefund	No	No
reservationOrders/merge	No	No
reservationOrders/reservations	No	No
reservationOrders/reservations/revisions	No	No
reservationOrders/return	No	No
reservationOrders/split	No	No
reservationOrders/swap	No	No
reservations	No	No
resources	No	No
validateReservationOrder	No	No

Microsoft.Cdn

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
edgenodes	No	No
profiles	Yes	Yes
profiles/endpoints	Yes	Yes
profiles/endpoints/customdomains	No	No
profiles/endpoints/origins	No	No
validateProbe	No	No

Microsoft.CertificateRegistration

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
certificateOrders	Yes	Yes
certificateOrders/certificates	No	No
validateCertificateRegistrationInformation	No	No

Microsoft.ClassicCompute

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
capabilities	No	No
domainNames	No	No
domainNames/capabilities	No	No
domainNames/internalLoadBalancers	No	No
domainNames/serviceCertificates	No	No
domainNames/slots	No	No
domainNames/slots/roles	No	No
moveSubscriptionResources	No	No
operatingSystemFamilies	No	No
operatingSystems	No	No
quotas	No	No

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
resourceTypes	No	No
validateSubscriptionMoveAvailability	No	No
virtualMachines	No	No
virtualMachines/diagnosticSettings	No	No

Microsoft.ClassicInfrastructureMigrate

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
classicInfrastructureResources	No	No

Microsoft.ClassicNetwork

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
capabilities	No	No
expressRouteCrossConnections	No	No
expressRouteCrossConnections/peerings	No	No
gatewaySupportedDevices	No	No
networkSecurityGroups	No	No
quotas	No	No
reservedIps	No	No
virtualNetworks	No	No
virtualNetworks/remoteVirtualNetworkPeeringProxies	No	No
virtualNetworks/virtualNetworkPeerings	No	No

Microsoft.ClassicStorage

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
capabilities	No	No
disks	No	No
images	No	No

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
osImages	No	No
osPlatformImages	No	No
publicImages	No	No
quotas	No	No
storageAccounts	No	No
storageAccounts/services	No	No
storageAccounts/services/diagnosticSettings	No	No
storageAccounts/vmImages	No	No
vmImages	No	No

Microsoft.CognitiveServices

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
accounts	Yes	Yes

Microsoft.Commerce

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
RateCard	No	No
UsageAggregates	No	No

Microsoft.Compute

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
availabilitySets	Yes	Yes
disks	Yes	Yes
images	Yes	Yes
restorePointCollections	Yes	Yes
restorePointCollections/restorePoints	No	No

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
sharedVMImages	Yes	Yes
sharedVMImages/versions	Yes	Yes
snapshots	Yes	Yes
virtualMachines	Yes	Yes
virtualMachines/diagnosticSettings	No	No
virtualMachines/extensions	Yes	Yes
virtualMachineScaleSets	Yes	Yes
virtualMachineScaleSets/extensions	No	No
virtualMachineScaleSets/networkInterfaces	No	No
virtualMachineScaleSets/publicIPAddresses	No	No
virtualMachineScaleSets/virtualMachines	No	No
virtualMachineScaleSets/virtualMachines/networkInterfaces	No	No

Microsoft.Consumption

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
AggregatedCost	No	No
Balances	No	No
Budgets	No	No
Charges	No	No
CostTags	No	No
credits	No	No
events	No	No
Forecasts	No	No
lots	No	No

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
Marketplaces	No	No
Pricesheets	No	No
products	No	No
ReservationDetails	No	No
ReservationRecommendations	No	No
ReservationSummaries	No	No
ReservationTransactions	No	No
Tags	No	No
Terms	No	No
UsageDetails	No	No

Microsoft.ContainerInstance

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
containerGroups	Yes	Yes
serviceAssociationLinks	No	No

Microsoft.ContainerRegistry

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
registries	Yes	Yes
registries/builds	No	No
registries/builds/cancel	No	No
registries/builds/getLogLink	No	No
registries/buildTasks	Yes	Yes
registries/buildTasks/steps	No	No
registries/eventGridFilters	No	No
registries/getBuildSourceUploadUrl	No	No
registries/GetCredentials	No	No

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
registries/importImage	No	No
registries/queueBuild	No	No
registries/regenerateCredential	No	No
registries/regenerateCredentials	No	No
registries/replications	Yes	Yes
registries/runs	No	No
registries/runs/cancel	No	No
registries/scheduleRun	No	No
registries/tasks	Yes	Yes
registries/updatePolicies	No	No
registries/webhooks	Yes	Yes
registries/webhooks/getCallbackConfig	No	No
registries/webhooks/ping	No	No

Microsoft.ContainerService

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
containerServices	Yes	Yes
managedClusters	Yes	Yes

Microsoft.ContentModerator

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
applications	Yes	Yes
updateCommunicationPreference	No	No

Microsoft.CortanaAnalytics

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
accounts	Yes	Yes

Microsoft.CostManagement

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
Alerts	No	No
BillingAccounts	No	No
Connectors	Yes	Yes
Departments	No	No
Dimensions	No	No
EnrollmentAccounts	No	No
Query	No	No
register	No	No
Reportconfigs	No	No
Reports	No	No

Microsoft.CustomerInsights

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
hubs	Yes	Yes
hubs/authorizationPolicies	No	No
hubs/connectors	No	No
hubs/connectors/mappings	No	No
hubs/interactions	No	No
hubs/kpi	No	No
hubs/links	No	No
hubs/profiles	No	No
hubs/roleAssignments	No	No
hubs/roles	No	No
hubs/suggestTypeSchema	No	No
hubs/views	No	No

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
hubs/widgetTypes	No	No

Microsoft.DataBox

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
jobs	Yes	Yes

Microsoft.DataBoxEdge

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
DataBoxEdgeDevices	Yes	Yes

Microsoft.Databricks

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
workspaces	Yes	No
workspaces/virtualNetworkPeerings	No	No

Microsoft.DataCatalog

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
catalogs	Yes	Yes

Microsoft.DataConnect

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
connectionManagers	Yes	Yes

Microsoft.DataFactory

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
dataFactories	Yes	No
dataFactories/diagnosticSettings	No	No
dataFactorySchema	No	No
factories	Yes	No

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
factories/integrationRuntimes	No	No

Microsoft.DataLakeAnalytics

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
accounts	Yes	Yes
accounts/dataLakeStoreAccounts	No	No
accounts/storageAccounts	No	No
accounts/storageAccounts/containers	No	No

Microsoft.DataLakeStore

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
accounts	Yes	Yes
accounts/eventGridFilters	No	No
accounts/firewallRules	No	No

Microsoft.DataMigration

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
services	Yes	Yes
services/projects	Yes	Yes

Microsoft.DBforMariaDB

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
servers	Yes	Yes
servers/recoverableServers	No	No
servers/virtualNetworkRules	No	No

Microsoft.DBforMySQL

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
servers	Yes	Yes

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
servers/recoverableServers	No	No
servers/virtualNetworkRules	No	No

Microsoft.DBforPostgreSQL

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
servers	Yes	Yes
servers/advisors	No	No
servers/queryTexts	No	No
servers/recoverableServers	No	No
servers/topQueryStatistics	No	No
servers/virtualNetworkRules	No	No
servers/waitStatistics	No	No

Microsoft.Devices

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
IotHubs	Yes	Yes
IotHubs/eventGridFilters	No	No
ProvisioningServices	Yes	Yes
usages	No	No

Microsoft.DevSpaces

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
controllers	Yes	Yes

Microsoft.DevTestLab

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
labs	Yes	Yes
labs/serviceRunners	Yes	Yes

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
labs/virtualMachines	Yes	Yes
schedules	Yes	Yes

Microsoft.DocumentDB

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
databaseAccountNames	No	No
databaseAccounts	Yes	Yes

Microsoft.DomainRegistration

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
domains	Yes	Yes
domains/domainOwnershipIdentifiers	No	No
generateSsoRequest	No	No
topLevelDomains	No	No
validateDomainRegistrationInformation	No	No

Microsoft.DynamicsLcs

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
lcsprojects	No	No
lcsprojects/clouddeployments	No	No
lcsprojects/connectors	No	No

Microsoft.EventGrid

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
domains	Yes	No
domains/topics	No	No
eventSubscriptions	No	No
extensionTopics	No	No

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
topics	Yes	No
topicTypes	No	No

Microsoft.EventHub

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
clusters	Yes	No
namespaces	Yes	No
namespaces/authorizationrules	No	No
namespaces/disasterrecoveryconfigs	No	No
namespaces/eventhubs	No	No
namespaces/eventhubs/authorizationrules	No	No
namespaces/eventhubs/consumergroups	No	No

Microsoft.Features

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
features	No	No
providers	No	No

Microsoft.Gallery

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
enroll	No	No
galleryitems	No	No
generateartifactaccessuri	No	No
myareas	No	No
myareas/areas	No	No
myareas/areas/areas	No	No
myareas/areas/areas/galleryitems	No	No

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
myareas/areas/galleryitems	No	No
myareas/galleryitems	No	No
register	No	No
resources	No	No
retrieveresourcesbyid	No	No

Microsoft.GuestConfiguration

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
guestConfigurationAssignments	No	No
software	No	No

Microsoft.HanaOnAzure

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
hanaInstances	Yes	Yes

Microsoft.HDInsight

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
clusters	Yes	Yes
clusters/applications	No	No

Microsoft.ImportExport

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
jobs	Yes	Yes

Microsoft.InformationProtection

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
labelGroups	No	No
labelGroups/labels	No	No

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
labelGroups/labels/conditions	No	No
labelGroups/labels/subLabels	No	No
labelGroups/labels/subLabels/conditions	No	No

microsoft.insights

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
actiongroups	Yes	Yes
activityLogAlerts	Yes	Yes
alertrules	Yes	Yes
automatedExportSettings	No	No
autoscalesettings	Yes	Yes
baseline	No	No
calculatebaseline	No	No
components	Yes	Yes
components/events	No	No
components/pricingPlans	No	No
components/query	No	No
diagnosticSettings	No	No
diagnosticSettingsCategories	No	No
eventCategories	No	No
eventtypes	No	No
extendedDiagnosticSettings	No	No
logDefinitions	No	No
logprofiles	No	No
logs	No	No
metricAlerts	Yes	Yes

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
migrateToNewPricingModel	No	No
myWorkbooks	No	No
queries	No	No
rollbackToLegacyPricingModel	No	No
scheduledqueryrules	Yes	Yes
vmInsightsOnboardingStatuses	No	No
webtests	Yes	Yes
workbooks	Yes	Yes

Microsoft.Intune

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
diagnosticSettings	No	No
diagnosticSettingsCategories	No	No

Microsoft.IoTCentral

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
IoTApps	Yes	Yes

Microsoft.IoTSpaces

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
Graph	Yes	Yes

Microsoft.KeyVault

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
deletedVaults	No	No
vaults	Yes	Yes
vaults/accessPolicies	No	No
vaults/secrets	No	No

Microsoft.Kusto

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
clusters	Yes	Yes
clusters/databases	No	No
clusters/databases/dataconnections	No	No
clusters/databases/eventhubconnections	No	No

Microsoft.LabServices

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
labaccounts	Yes	Yes
users	No	No

Microsoft.LocationBasedServices

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
accounts	Yes	Yes

Microsoft.LocationServices

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
accounts	Yes	Yes

Microsoft.LogAnalytics

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
logs	No	No

Microsoft.Logic

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
integrationAccounts	Yes	Yes
workflows	Yes	Yes

Microsoft.MachineLearning

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
commitmentPlans	Yes	Yes
webServices	Yes	Yes
Workspaces	Yes	Yes

Microsoft.MachineLearningExperimentation

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
accounts	Yes	Yes
accounts/workspaces	Yes	Yes
accounts/workspaces/projects	Yes	Yes
teamAccounts	Yes	Yes
teamAccounts/workspaces	Yes	Yes
teamAccounts/workspaces/projects	Yes	Yes

Microsoft.MachineLearningModelManagement

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
accounts	Yes	Yes

Microsoft.MachineLearningServices

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
workspaces	Yes	Yes
workspaces/computes	No	No

Microsoft.ManagedIdentity

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
Identities	No	No
userAssignedIdentities	Yes	

Microsoft.Management

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
getEntities	No	No
managementGroups	No	No
resources	No	No
startTenantBackfill	No	No
tenantBackfillStatus	No	No

Microsoft.Maps

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
accounts	Yes	Yes
accounts/eventGridFilters	No	No

Microsoft.Marketplace

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
offers	No	No
offerTypes	No	No
offerTypes/publishers	No	No
offerTypes/publishers/offers	No	No
offerTypes/publishers/offers/plans	No	No
offerTypes/publishers/offers/plans/agreements	No	No
offerTypes/publishers/offers/plans/configs	No	No
offerTypes/publishers/offers/plans/configs/importImage	No	No
privategalleryitems	No	No
products	No	No

Microsoft.MarketplaceApps

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
classicDevServices	Yes	Yes
updateCommunicationPreference	No	No

Microsoft.MarketplaceOrdering

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
agreements	No	No
offertypes	No	No

Microsoft.Media

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
mediaservices	Yes	Yes
mediaservices/accountFilters	No	No
mediaservices/assets	No	No
mediaservices/assets/assetFilters	No	No
mediaservices/contentKeyPolicies	No	No
mediaservices/eventGridFilters	No	No
mediaservices/liveEventOperations	No	No
mediaservices/liveEvents	Yes	Yes
mediaservices/liveEvents/liveOutputs	No	No
mediaservices/liveOutputOperations	No	No
mediaservices/streamingEndpointOperations	No	No
mediaservices/streamingEndpoints	Yes	Yes
mediaservices/streamingLocators	No	No
mediaservices/streamingPolicies	No	No
mediaservices/transforms	No	No
mediaservices/transforms/jobs	No	No

Microsoft.Migrate

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
projects	Yes	Yes

Microsoft.Network

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
applicationGateways	Yes	No
applicationSecurityGroups	Yes	Yes
azureFirewallFqdnTags	No	No
azureFirewalls	Yes	No
bgpServiceCommunities	No	No
connections	Yes	Yes
ddosCustomPolicies	Yes	Yes
ddosProtectionPlans	Yes	Yes
dnsOperationStatuses	No	No
dnszones	Yes	Yes
dnszones/A	No	No
dnszones/AAAA	No	No
dnszones/all	No	No
dnszones/CAA	No	No
dnszones/CNAME	No	No
dnszones/MX	No	No
dnszones/NS	No	No
dnszones/PTR	No	No
dnszones/recordsets	No	No
dnszones/SOA	No	No
dnszones/SRV	No	No

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
dnszones/TXT	No	No
expressRouteCircuits	Yes	No
expressRouteServiceProviders	No	No
frontdoors	Yes, but limited (see note below)	Yes
frontdoorWebApplicationFirewallPolicies	Yes, but limited (see note below)	Yes
getDnsResourceReference	No	No
interfaceEndpoints	Yes	Yes
internalNotify	No	No
loadBalancers	Yes	No
localNetworkGateways	Yes	Yes
natGateways	Yes	Yes
networkIntentPolicies	Yes	Yes
networkInterfaces	Yes	Yes
networkProfiles	Yes	Yes
networkSecurityGroups	Yes	Yes
networkWatchers	Yes	No
networkWatchers/connectionMonitors	Yes	No
networkWatchers/lenses	Yes	No
networkWatchers/pingMeshes	Yes	No
privateLinkServices	Yes	Yes
publicIPAddresses	Yes	Yes
publicIPPrefixes	Yes	Yes
routeFilters	Yes	Yes
routeTables	Yes	Yes
serviceEndpointPolicies	Yes	Yes

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
trafficManagerGeographicHierarchies	No	No
trafficmanagerprofiles	Yes	Yes
trafficmanagerprofiles/heatMaps	No	No
virtualHubs	Yes	Yes
virtualNetworkGateways	Yes	Yes
virtualNetworks	Yes	Yes
virtualNetworks/subnets	No	No
virtualNetworkTaps	Yes	Yes
virtualWans	Yes	Yes
vpnGateways	Yes	No
vpnSites	Yes	Yes
webApplicationFirewallPolicies	Yes	Yes

For Azure Front Door Service, you can apply tags when creating the resource, but updating or adding tags is not currently supported.

Microsoft.NotificationHubs

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
namespaces	Yes	No
namespaces/notificationHubs	Yes	No

Microsoft.OperationalInsights

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
devices	No	No
linkTargets	No	No
storageInsightConfigs	No	No
workspaces	Yes	Yes
workspaces/dataSources	No	No

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
workspaces/linkedServices	No	No
workspaces/query	No	No

Microsoft.OperationsManagement

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
managementassociations	No	No
managementconfigurations	Yes	Yes
solutions	Yes	Yes
views	Yes	Yes

Microsoft.PolicyInsights

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
policyEvents	No	No
policyStates	No	No
policyTrackedResources	No	No
remediations	No	No

Microsoft.Portal

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
consoles	No	No
dashboards	Yes	Yes
userSettings	No	No

Microsoft.PowerBI

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
workspaceCollections	Yes	Yes

Microsoft.PowerBIDedicated

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
capacities	Yes	Yes

Microsoft.ProjectOxford

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
accounts	Yes	Yes

Microsoft.RecoveryServices

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
backupProtectedItems	No	No
vaults	Yes	Yes

Microsoft.Relay

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
namespaces	Yes	Yes
namespaces/authorizationrules	No	No
namespaces/hybridconnections	No	No
namespaces/hybridconnections/authorizationrules	No	No
namespaces/wcfrelays	No	No
namespaces/wcfrelays/authorizationrules	No	No

Microsoft.ResourceGraph

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
resources	No	No
subscriptionsStatus	No	No

Microsoft.ResourceHealth

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
availabilityStatuses	No	No

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
childAvailabilityStatuses	No	No
childResources	No	No
events	No	No
impactedResources	No	No
notifications	No	No

Microsoft.Resources

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
deployments	No	No
deployments/operations	No	No
links	No	No
notifyResourceJobs	No	No
providers	No	No
resourceGroups	No	No
resources	No	No
subscriptions	No	No
subscriptions/providers	No	No
subscriptions/resourceGroups	No	No
subscriptions/resourcegroups/resources	No	No
subscriptions/resources	No	No
subscriptions/tagnames	No	No
subscriptions/tagNames/tagValues	No	No
tenants	No	No

Microsoft.SaaS

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
applications	Yes	Yes

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
saasresources	No	No

Microsoft.Scheduler

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
flows	Yes	Yes
jobcollections	Yes	Yes

Microsoft.Search

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
resourceHealthMetadata	No	No
searchServices	Yes	Yes

Microsoft.Security

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
advancedThreatProtectionSettings	No	No
alerts	No	No
allowedConnections	No	No
appliances	No	No
applicationWhitelistings	No	No
AutoProvisioningSettings	No	No
Compliances	No	No
dataCollectionAgents	No	No
discoveredSecuritySolutions	No	No
externalSecuritySolutions	No	No
InformationProtectionPolicies	No	No
jitNetworkAccessPolicies	No	No
monitoring	No	No

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
monitoring/antimalware	No	No
monitoring/baseline	No	No
monitoring/patch	No	No
policies	No	No
pricings	No	No
securityContacts	No	No
securitySolutions	No	No
securitySolutionsReferenceData	No	No
securityStatus	No	No
securityStatus/endpoints	No	No
securityStatus/subnets	No	No
securityStatus/virtualMachines	No	No
securityStatuses	No	No
securityStatusesSummaries	No	No
settings	No	No
tasks	No	No
topologies	No	No
workspaceSettings	No	No

Microsoft.SecurityGraph

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
diagnosticSettings	No	No
diagnosticSettingsCategories	No	No

Microsoft.ServiceBus

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
namespaces	Yes	No

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
namespaces/authorizationrules	No	No
namespaces/disasterrecoveryconfigs	No	No
namespaces/eventgridfilters	No	No
namespaces/queues	No	No
namespaces/queues/authorizationrules	No	No
namespaces/topics	No	No
namespaces/topics/authorizationrules	No	No
namespaces/topics/subscriptions	No	No
namespaces/topics/subscriptions/rules	No	No
premiumMessagingRegions	No	No

Microsoft.ServiceFabric

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
clusters	Yes	Yes
clusters/applications	No	No

Microsoft.ServiceFabricMesh

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
applications	Yes	Yes
gateways	Yes	Yes
networks	Yes	Yes
secrets	Yes	Yes
volumes	Yes	Yes

Microsoft.SignalRService

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
SignalR	Yes	Yes

Microsoft.Solutions

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
applianceDefinitions	Yes	Yes
appliances	Yes	Yes
applicationDefinitions	Yes	Yes
applications	Yes	Yes
jitRequests	Yes	Yes

Microsoft.SQL

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
managedInstances	Yes	Yes
managedInstances/databases	Yes (see note below)	Yes
managedInstances/databases/backupShortTermRetentionPolicies	No	No
managedInstances/databases/schemas/tables/columns/sensitivityLabels	No	No
managedInstances/databases/vulnerabilityAssessments	No	No
managedInstances/databases/vulnerabilityAssessments/rules/baselines	No	No
managedInstances/encryptionProtector	No	No
managedInstances/keys	No	No
managedInstances/restorableDroppedDatabases/backupShortTermRetentionPolicies	No	No
managedInstances/vulnerabilityAssessments	No	No
servers	Yes	Yes
servers/administrators	No	No
servers/communicationLinks	No	No
servers/databases	Yes (see note below)	Yes

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
servers/encryptionProtector	No	No
servers/firewallRules	No	No
servers/keys	No	No
servers/restorableDroppedDatabases	No	No
servers/serviceobjectives	No	No
servers/tdeCertificates	No	No

NOTE
The Master database doesn't support tags, but other databases, including Azure SQL Data Warehouse databases, support tags. Azure SQL Data Warehouse databases must be in Active (not Paused) state.

Microsoft.SqlVirtualMachine

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
SqlVirtualMachineGroups	Yes	Yes
SqlVirtualMachineGroups/AvailabilityGroupListeners	No	No
SqlVirtualMachines	Yes	Yes

Microsoft.Storage

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
storageAccounts	Yes	Yes
storageAccounts/blobServices	No	No
storageAccounts/fileServices	No	No
storageAccounts/queueServices	No	No
storageAccounts/services	No	No
storageAccounts/tableServices	No	No
usages	No	No

Microsoft.StorageSync

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
storageSyncServices	Yes	Yes
storageSyncServices/registeredServers	No	No
storageSyncServices/syncGroups	No	No
storageSyncServices/syncGroups/cloudEndpoints	No	No
storageSyncServices/syncGroups/serverEndpoints	No	No
storageSyncServices/workflows	No	No

Microsoft.StorSimple

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
managers	Yes	Yes

Microsoft.StreamAnalytics

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
streamingjobs	Yes (see note below)	Yes
streamingjobs/diagnosticSettings	No	No

NOTE

You can't add a tag when streamingjobs is running. Stop the resource to add a tag.

Microsoft.Subscription

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
CreateSubscription	No	No
SubscriptionDefinitions	No	No
SubscriptionOperations	No	No

microsoft.support

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
supporttickets	No	No

Microsoft.TerraformOSS

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
providerRegistrations	Yes	Yes
resources	Yes	Yes

Microsoft.TimeSeriesInsights

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
environments	Yes	No
environments/accessPolicies	No	No
environments/eventsources	Yes	No
environments/referenceDataSets	Yes	No

microsoft.visualstudio

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
account	Yes	Yes
account/extension	Yes	Yes
account/project	Yes	Yes

Microsoft.Web

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
apiManagementAccounts	No	No
apiManagementAccounts/apiAcls	No	No
apiManagementAccounts/apis	No	No
apiManagementAccounts/apis/apiAcls	No	No
apiManagementAccounts/apis/connectionAcls	No	No
apiManagementAccounts/apis/connections	No	No
apiManagementAccounts/apis/connections/connectionAcls	No	No

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
apiManagementAccounts/apis/localizedDefinitions	No	No
apiManagementAccounts/connectionAcls	No	No
apiManagementAccounts/connections	No	No
billingMeters	No	No
certificates	Yes	Yes
connectionGateways	Yes	Yes
connections	Yes	Yes
customApis	Yes	Yes
deletedSites	No	No
functions	No	No
hostingEnvironments	Yes	Yes
hostingEnvironments/multiRolePools	No	No
hostingEnvironments/multiRolePools/instances	No	No
hostingEnvironments/workerPools	No	No
hostingEnvironments/workerPools/instances	No	No
publishingUsers	No	No
recommendations	No	No
resourceHealthMetadata	No	No
runtimes	No	No
serverFarms	Yes	Yes
serverFarms/workers	No	No
sites	Yes	Yes
sites/domainOwnershipIdentifiers	No	No
sites/hostNameBindings	No	No

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
sites/instances	No	No
sites/instances/extensions	No	No
sites/premieraddons	Yes	Yes
sites/recommendations	No	No
sites/resourceHealthMetadata	No	No
sites/slots	Yes	Yes
sites/slots/hostNameBindings	No	No
sites/slots/instances	No	No
sites/slots/instances/extensions	No	No
sourceControls	No	No
validate	No	No
verifyHostingEnvironmentVnet	No	No

Microsoft.WindowsDefenderATP

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
diagnosticSettings	No	No
diagnosticSettingsCategories	No	No

Microsoft.WindowsIoT

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
DeviceServices	Yes	Yes

Microsoft.WorkloadMonitor

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
components	No	No
componentsSummary	No	No
monitorInstances	No	No

RESOURCE TYPE	SUPPORTS TAGS	TAG IN COST REPORT
monitorInstancesSummary	No	No
monitors	No	No
notificationSettings	No	No

Next steps

To learn how to apply tags to resources, see Use tags to organize your Azure resources.

Manage Azure Resource Manager resource groups by using the Azure portal

6/20/2019 • 3 minutes to read • Edit Online

Learn how to use the Azure portal with Azure Resource Manager to manage your Azure resource groups. For managing Azure resources, see Manage Azure resources by using the Azure portal.

Other articles about managing resource groups:

- Manage Azure resource groups by using Azure CLI
- Manage Azure resource groups by using Azure PowerShell

NOTE

This article provides steps for how to delete personal data from the device or service and can be used to support your obligations under the GDPR. If you're looking for general info about GDPR, see the GDPR section of the Service Trust portal.

What is a resource group

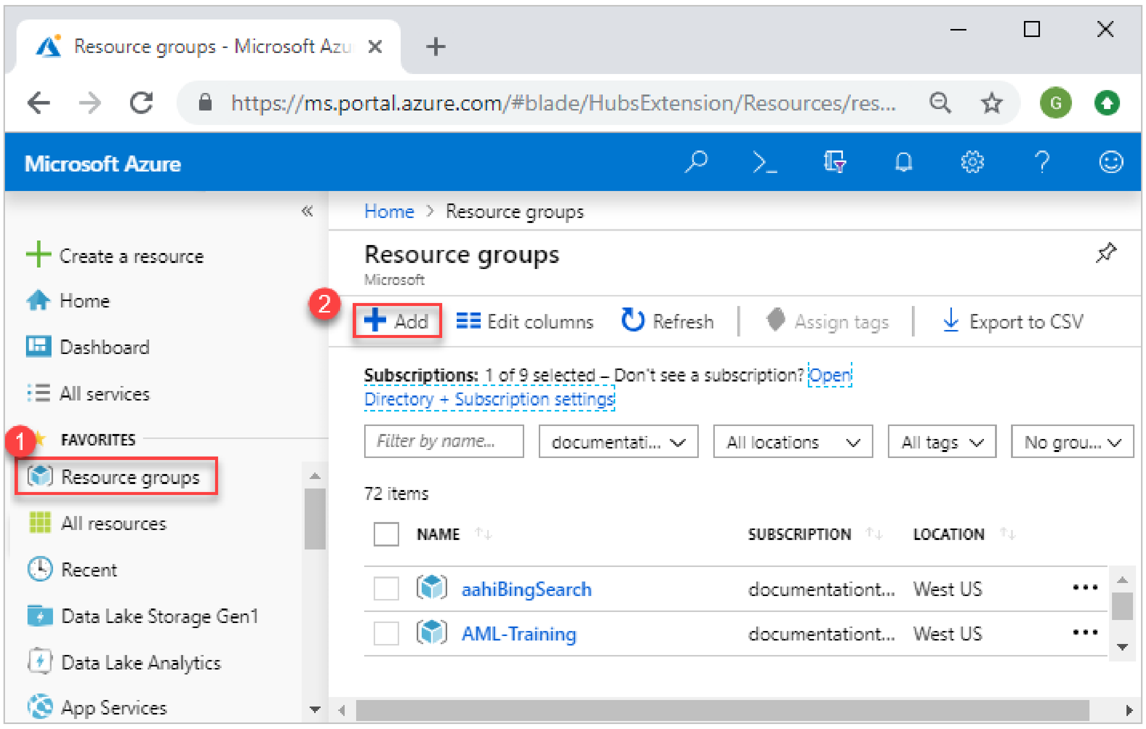
A resource group is a container that holds related resources for an Azure solution. The resource group can include all the resources for the solution, or only those resources that you want to manage as a group. You decide how you want to allocate resources to resource groups based on what makes the most sense for your organization. Generally, add resources that share the same lifecycle to the same resource group so you can easily deploy, update, and delete them as a group.

The resource group stores metadata about the resources. Therefore, when you specify a location for the resource group, you are specifying where that metadata is stored. For compliance reasons, you may need to ensure that your data is stored in a particular region.

The resource group stores metadata about the resources. When you specify a location for the resource group, you're specifying where that metadata is stored.

Create resource groups

1. Sign in to the Azure portal.
2. Select **Resource groups**



Subscriptions: 1 of 9 selected – Don't see a subscription? Open Directory + Subscription settings

72 items

NAME	SUBSCRIPTION	LOCATION
aahiBingSearch	documentationt...	West US
AML-Training	documentationt...	West US

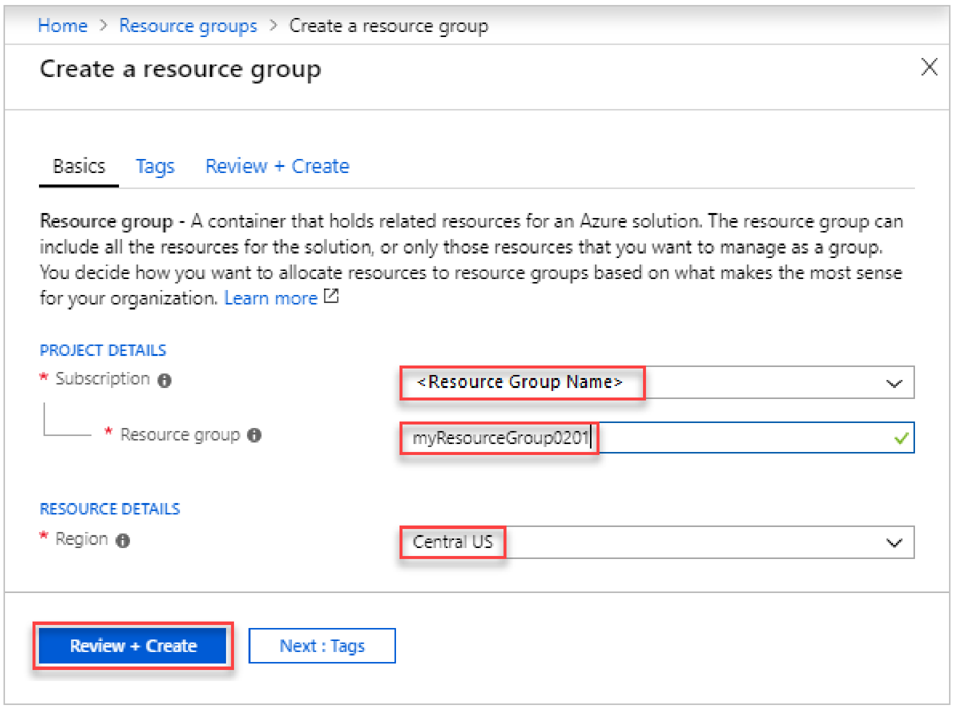
3. Select **Add**.

4. Enter the following values:

   - **Subscription**: Select your Azure subscription.

   - **Resource group**: Enter a new resource group name.

   - **Region**: Select an Azure location, such as **Central US**.



Resource group - A container that holds related resources for an Azure solution. The resource group can include all the resources for the solution, or only those resources that you want to manage as a group. You decide how you want to allocate resources to resource groups based on what makes the most sense for your organization. Learn more

PROJECT DETAILS

* Subscription: <Resource Group Name>

* Resource group: myResourceGroup0201

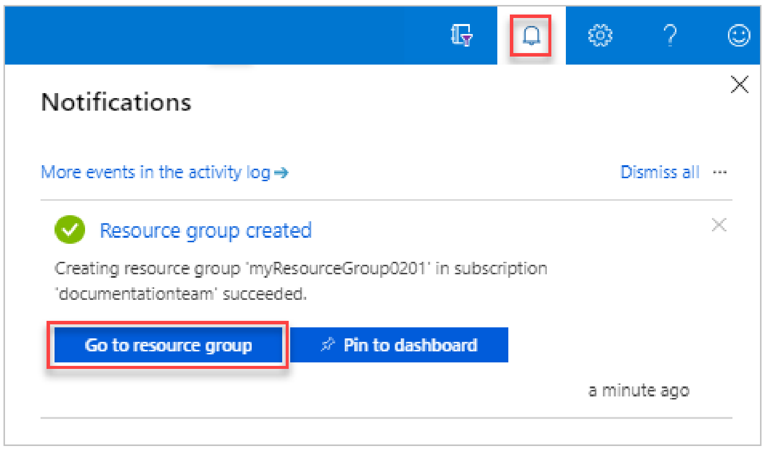
RESOURCE DETAILS

* Region: Central US

5. Select **Review + Create**

6. Select **Create**. It takes a few seconds to create a resource group.

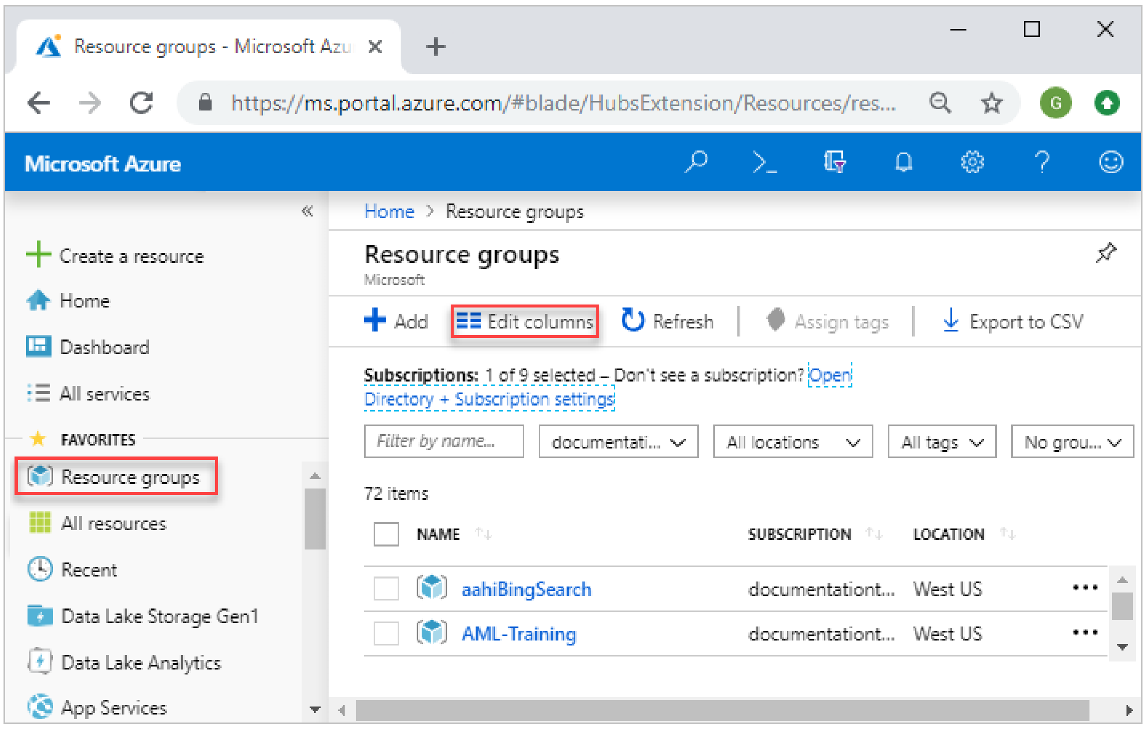
7. Select **Refresh** from the top menu to refresh the resource group list, and then select the newly created resource group to open it. Or select **Notification**(the bell icon) from the top, and then select **Go to resource group** to open the newly created resource group



Creating resource group 'myResourceGroup0201' in subscription 'documentationteam' succeeded.

List resource groups

1. Sign in to the Azure portal.
2. To list the resource groups, select **Resource groups**



Subscriptions: 1 of 9 selected – Don't see a subscription? Open Directory + Subscription settings

NAME	SUBSCRIPTION	LOCATION
aahiBingSearch	documentationt...	West US
AML-Training	documentationt...	West US

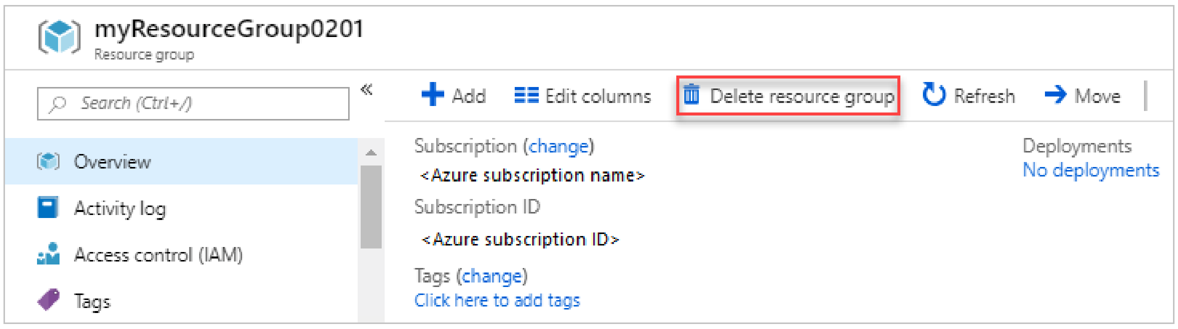
3. To customize the information displayed for the resource groups, select **Edit columns**. The following screenshot shows the addition columns you could add to the display:

Open resource groups

1. Sign in to the Azure portal.
2. Select **Resource groups**.
3. Select the resource group you want to open.

Delete resource groups

1. Open the resource group you want to delete. See Open resource groups.
2. Select **Delete resource group**.



For more information about how Azure Resource Manager orders the deletion of resources, see Azure Resource Manager resource group deletion.

Deploy resources to a resource group

After you have created a Resource Manager template, you can use the Azure portal to deploy your Azure resources. For creating a template, see Quickstart: Create and deploy Azure Resource Manager templates by using the Azure portal. For deploying a template using the portal, see Deploy resources with Resource Manager templates and Azure portal.

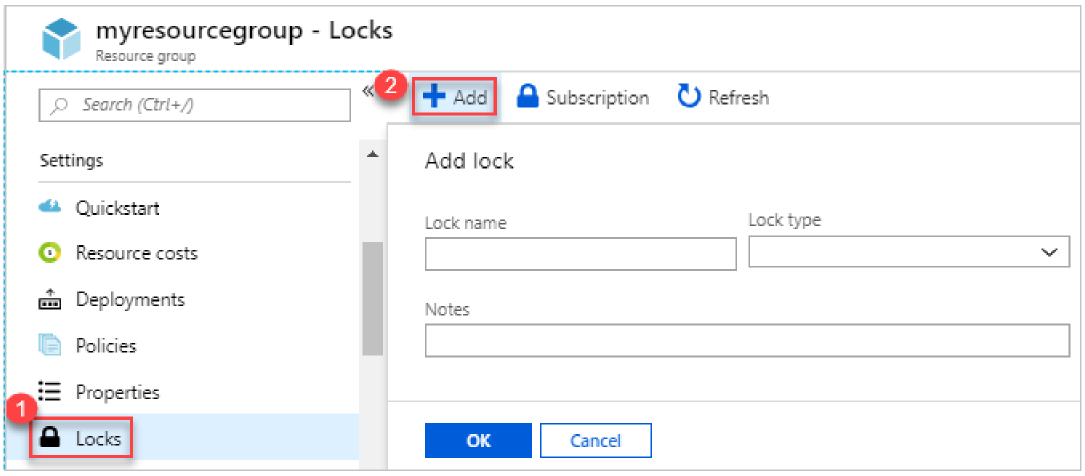
Move to another resource group or subscription

You can move the resources in the group to another resource group. For more information, see Move resources to new resource group or subscription.

Lock resource groups

Locking prevents other users in your organization from accidentally deleting or modifying critical resources, such as Azure subscription, resource group, or resource.

1. Open the resource group you want to delete. See Open resource groups.
2. In the left pane, select **Locks**.
3. To add a lock to the resource group, select **Add**.
4. Enter **Lock name**, **Lock type**, and **Notes**. The lock types include **Read-only**, and **Delete**.



For more information, see Lock resources to prevent unexpected changes.

Tag resource groups

You can apply tags to resource groups and resources to logically organize your assets. For information, see Using tags to organize your Azure resources.

Export resource groups to templates

For information about exporting templates, see Single and multi-resource export to template - Portal.

Manage access to resource groups

Role-based access control (RBAC) is the way that you manage access to resources in Azure. For more information, see Manage access using RBAC and the Azure portal.

Next steps

- To learn Azure Resource Manager, see Azure Resource Manager overview.
- To learn the Resource Manager template syntax, see Understand the structure and syntax of Azure Resource Manager templates.
- To learn how to develop templates, see the step-by-step tutorials.
- To view the Azure Resource Manager template schemas, see template reference.

Manage Azure Resource Manager resource groups by using Azure CLI

6/20/2019 • 3 minutes to read • Edit Online

Learn how to use Azure CLI with Azure Resource Manager to manage your Azure resource groups. For managing Azure resources, see Manage Azure resources by using Azure CLI.

Other articles about managing resource groups:

- Manage Azure resource groups by using the Azure portal
- Manage Azure resource groups by using Azure PowerShell

What is a resource group

A resource group is a container that holds related resources for an Azure solution. The resource group can include all the resources for the solution, or only those resources that you want to manage as a group. You decide how you want to allocate resources to resource groups based on what makes the most sense for your organization. Generally, add resources that share the same lifecycle to the same resource group so you can easily deploy, update, and delete them as a group.

The resource group stores metadata about the resources. Therefore, when you specify a location for the resource group, you are specifying where that metadata is stored. For compliance reasons, you may need to ensure that your data is stored in a particular region.

The resource group stores metadata about the resources. When you specify a location for the resource group, you're specifying where that metadata is stored.

Create resource groups

The following CLI script creates a resource group, and then shows the resource group.

```
echo "Enter the Resource Group name:" &&
read resourceGroupName &&
echo "Enter the location (i.e. centralus):" &&
read location &&
az group create --name $resourceGroupName --location $location
```

List resource groups

The following CLI script lists the resource groups under your subscription.

```
az group list
```

To get one resource group:

```
echo "Enter the Resource Group name:" &&
read resourceGroupName &&
az group show --name $resourceGroupName
```

Delete resource groups

The following CLI script deletes a resource group:

```
echo "Enter the Resource Group name:" &&
read resourceGroupName &&
az group delete --name $resourceGroupName
```

For more information about how Azure Resource Manager orders the deletion of resources, see Azure Resource Manager resource group deletion.

Deploy resources to an existing resource group

See Deploy resources to an existing resource group.

Deploy a resource group and resources

You can create a resource group and deploy resources to the group by using a Resource Manager template. For more information, see Create resource group and deploy resources.

Redeploy when deployment fails

This feature is also known as *Rollback on error*. For more information, see Redeploy when deployment fails.

Move to another resource group or subscription

You can move the resources in the group to another resource group. For more information, see Move resources.

Lock resource groups

Locking prevents other users in your organization from accidentally deleting or modifying critical resources, such as Azure subscription, resource group, or resource.

The following script locks a resource group so the resource group can't be deleted.

```
echo "Enter the Resource Group name:" &&
read resourceGroupName &&
az lock create --name LockGroup --lock-type CanNotDelete --resource-group $resourceGroupName
```

The following script gets all locks for a resource group:

```
echo "Enter the Resource Group name:" &&
read resourceGroupName &&
az lock list --resource-group $resourceGroupName
```

The following script deletes a lock:

```
echo "Enter the Resource Group name:" &&
read resourceGroupName &&
echo "Enter the lock name:" &&
read lockName &&
az lock delete --name $lockName --resource-group $resourceGroupName
```

For more information, see Lock resources with Azure Resource Manager.

Tag resource groups

You can apply tags to resource groups and resources to logically organize your assets. For information, see Using tags to organize your Azure resources.

Export resource groups to templates

After setting up your resource group successfully, you may want to view the Resource Manager template for the resource group. Exporting the template offers two benefits:

- Automate future deployments of the solution because the template contains all the complete infrastructure.
- Learn template syntax by looking at the JavaScript Object Notation (JSON) that represents your solution.

```
echo "Enter the Resource Group name:" &&
read resourceGroupName &&
az group export --name $resourceGroupName
```

The script displays the template on the console. Copy the JSON, and save as a file.

For more information, see Single and multi-resource export to template in Azure portal.

Manage access to resource groups

Role-based access control (RBAC) is the way that you manage access to resources in Azure. For more information, see Manage access using RBAC and Azure CLI.

Next steps

- To learn Azure Resource Manager, see Azure Resource Manager overview.
- To learn the Resource Manager template syntax, see Understand the structure and syntax of Azure Resource Manager templates.
- To learn how to develop templates, see the step-by-step tutorials.
- To view the Azure Resource Manager template schemas, see template reference.

Manage Azure Resource Manager resource groups by using Azure PowerShell

7/10/2019 • 4 minutes to read • Edit Online

Learn how to use Azure PowerShell with Azure Resource Manager to manage your Azure resource groups. For managing Azure resources, see Manage Azure resources by using Azure PowerShell.

Other articles about managing resource groups:

- Manage Azure resource groups by using the Azure portal
- Manage Azure resource groups by using Azure CLI

What is a resource group

A resource group is a container that holds related resources for an Azure solution. The resource group can include all the resources for the solution, or only those resources that you want to manage as a group. You decide how you want to allocate resources to resource groups based on what makes the most sense for your organization. Generally, add resources that share the same lifecycle to the same resource group so you can easily deploy, update, and delete them as a group.

The resource group stores metadata about the resources. Therefore, when you specify a location for the resource group, you're specifying where that metadata is stored. For compliance reasons, you may need to ensure that your data is stored in a particular region.

The resource group stores metadata about the resources. When you specify a location for the resource group, you're specifying where that metadata is stored.

Create resource groups

The following PowerShell script creates a resource group, and then shows the resource group.

```
$resourceGroupName = Read-Host -Prompt "Enter the Resource Group name"
$location = Read-Host -Prompt "Enter the location (i.e. centralus)"

New-AzResourceGroup -Name $resourceGroupName -Location $location

Get-AzResourceGroup -Name $resourceGroupName
```

List resource groups

The following PowerShell script lists the resource groups under your subscription.

```
Get-AzResourceGroup
```

To get one resource group:

```
$resourceGroupName = Read-Host -Prompt "Enter the Resource Group name"

Get-AzResourceGroup -Name $resourceGroupName
```

Delete resource groups

The following PowerShell script deletes a resource group:

```
$resourceGroupName = Read-Host -Prompt "Enter the Resource Group name"

Remove-AzResourceGroup -Name $resourceGroupName
```

For more information about how Azure Resource Manager orders the deletion of resources, see Azure Resource Manager resource group deletion.

Deploy resources to an existing resource group

See Deploy resources to an existing resource group.

To validate a resource group deployment, see Test-AzResourceGroupDeployment.

Deploy a resource group and resources

You can create a resource group and deploy resources to the group by using a Resource Manager template. For more information, see Create resource group and deploy resources.

Redeploy when deployment fails

This feature is also known as *Rollback on error*. For more information, see Redeploy when deployment fails.

Move to another resource group or subscription

You can move the resources in the group to another resource group. For more information, see Move resources to new resource group or subscription.

Lock resource groups

Locking prevents other users in your organization from accidentally deleting or modifying critical resources, such as Azure subscription, resource group, or resource.

The following script locks a resource group so the resource group can't be deleted.

```
$resourceGroupName = Read-Host -Prompt "Enter the Resource Group name"

New-AzResourceLock -LockName LockGroup -LockLevel CanNotDelete -ResourceGroupName $resourceGroupName
```

The following script gets all locks for a resource group:

```
$resourceGroupName = Read-Host -Prompt "Enter the Resource Group name"

Get-AzResourceLock -ResourceGroupName $resourceGroupName
```

For more information, see Lock resources with Azure Resource Manager.

Tag resource groups

You can apply tags to resource groups and resources to logically organize your assets. For information, see Using tags to organize your Azure resources.

Export resource groups to templates

After setting up your resource group, you can view a Resource Manager template for the resource group. Exporting the template offers two benefits:

- Automate future deployments of the solution because the template contains the complete infrastructure.
- Learn template syntax by looking at the JavaScript Object Notation (JSON) that represents your solution.

To export all resources in a resource group, use the Export-AzResourceGroup cmdlet and provide the resource group name.

```
$resourceGroupName = Read-Host -Prompt "Enter the Resource Group name"

Export-AzResourceGroup -ResourceGroupName $resourceGroupName
```

It saves the template as a local file.

Instead of exporting all resources in the resource group, you can select which resources to export.

To export one resource, pass that resource ID.

```
$resource = Get-AzResource `
  -ResourceGroupName <resource-group-name> `
  -ResourceName <resource-name> `
  -ResourceType <resource-type>
Export-AzResourceGroup `
  -ResourceGroupName <resource-group-name> `
  -Resource $resource.ResourceId
```

To export more than one resource, pass the resource IDs in an array.

```
Export-AzResourceGroup `
  -ResourceGroupName <resource-group-name> `
  -Resource @($resource1.ResourceId, $resource2.ResourceId)
```

When exporting the template, you can specify whether parameters are used in the template. By default, parameters for resource names are included but they don't have a default value. You must pass that parameter value during deployment.

```
"parameters": {
  "serverfarms_demoHostPlan_name": {
    "defaultValue": null,
    "type": "String"
  },
  "sites_webSite3bwt23ktvdo36_name": {
    "defaultValue": null,
    "type": "String"
  }
}
```

In the resource, the parameter is used for the name.

```
"resources": [
  {
    "type": "Microsoft.Web/serverfarms",
    "apiVersion": "2016-09-01",
    "name": "[parameters('serverfarms_demoHostPlan_name')]",
    ...
  }
]
```

If you use the `-IncludeParameterDefaultValue` parameter when exporting the template, the template parameter includes a default value that is set to the current value. You can either use that default value or overwrite the default value by passing in a different value.

```
"parameters": {
  "serverfarms_demoHostPlan_name": {
    "defaultValue": "demoHostPlan",
    "type": "String"
  },
  "sites_webSite3bwt23ktvdo36_name": {
    "defaultValue": "webSite3bwt23ktvdo36",
    "type": "String"
  }
}
```

If you use the `-SkipResourceNameParameterization` parameter when exporting the template, parameters for resource names aren't included in the template. Instead, the resource name is set directly on the resource to its current value. You can't customize the name during deployment.

```
"resources": [
  {
    "type": "Microsoft.Web/serverfarms",
    "apiVersion": "2016-09-01",
    "name": "demoHostPlan",
    ...
  }
]
```

For more information, see Single and multi-resource export to template in Azure portal.

Manage access to resource groups

Role-based access control (RBAC) is the way that you manage access to resources in Azure. For more information, see Manage access using RBAC and Azure PowerShell.

Next steps

- To learn Azure Resource Manager, see Azure Resource Manager overview.
- To learn the Resource Manager template syntax, see Understand the structure and syntax of Azure Resource Manager templates.
- To learn how to develop templates, see the step-by-step tutorials.
- To view the Azure Resource Manager template schemas, see template reference.

Manage Azure resources by using the Azure portal

6/18/2019 • 3 minutes to read • Edit Online

Learn how to use the Azure portal with Azure Resource Manager to manage your Azure resources. For managing resource groups, see Manage Azure resource groups by using the Azure portal.

Other articles about managing resources:

- Manage Azure resources by using Azure CLI
- Manage Azure resources by using Azure PowerShell

NOTE

This article provides steps for how to delete personal data from the device or service and can be used to support your obligations under the GDPR. If you're looking for general info about GDPR, see the GDPR section of the Service Trust portal.

Deploy resources to a resource group

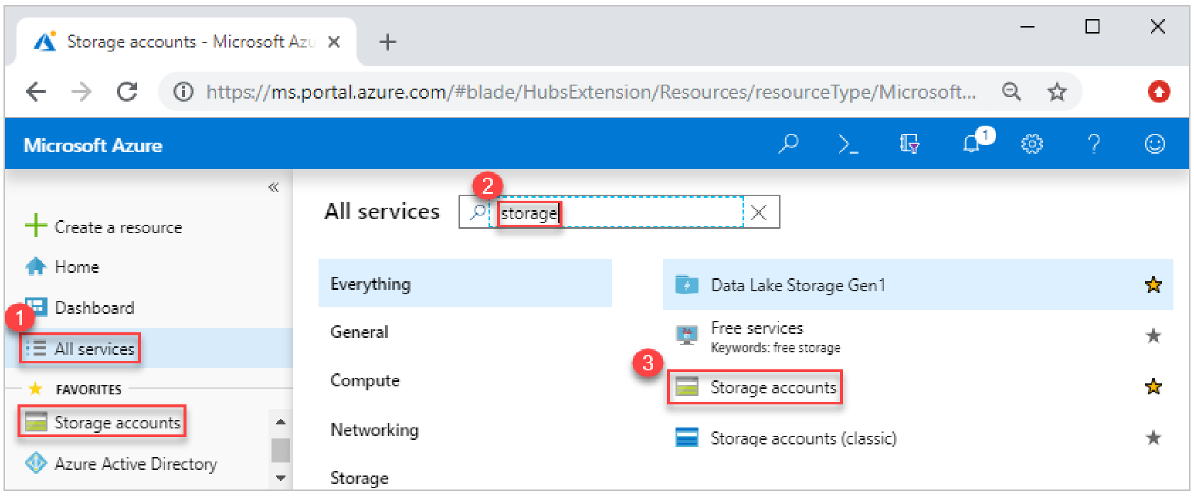
After you have created a Resource Manager template, you can use the Azure portal to deploy your Azure resources. For creating a template, see Quickstart: Create and deploy Azure Resource Manager templates by using the Azure portal. For deploying a template using the portal, see Deploy resources with Resource Manager templates and Azure portal.

Open resources

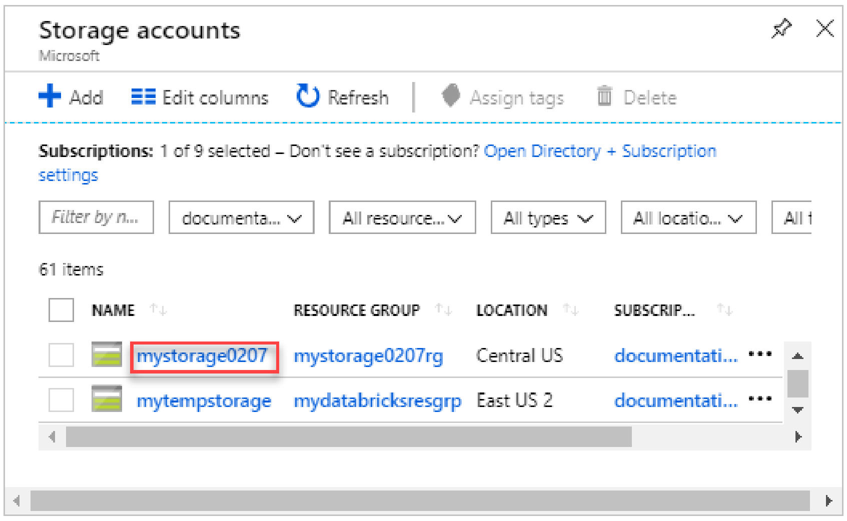
Azure resources are organized by Azure services and by resource groups. The following procedures shows how to open a storage account called **mystorage0207**. The virtual machine resides in a resource group called **mystorage0207rg**.

To open a resource by the service type:

1. Sign in to the Azure portal.
2. In the left pane, select the Azure service. In this case, **Storage accounts**. If you don't see the service listed, select **All services**, and then select the service type.



3. Select the resource you want to open.



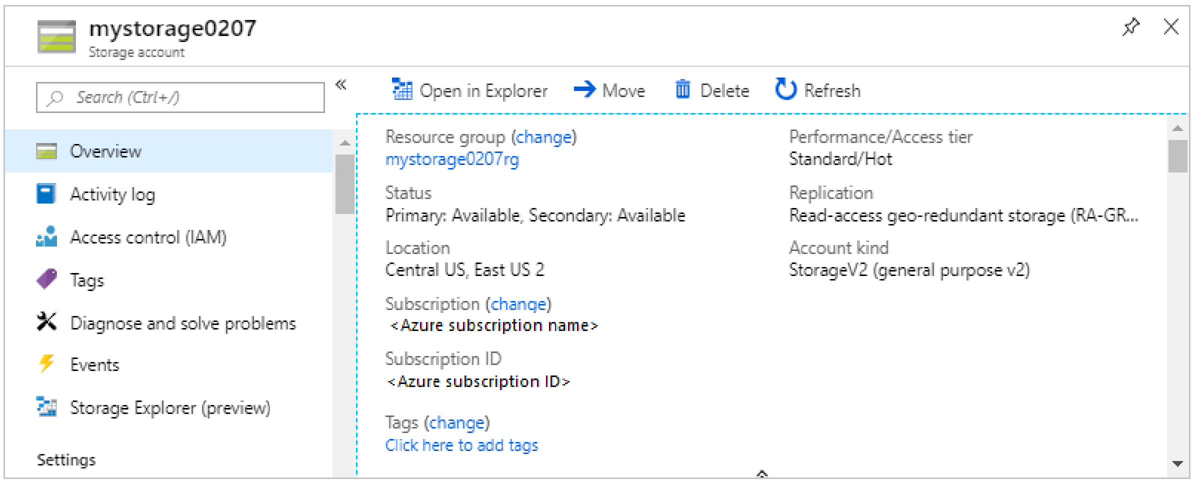
Subscriptions: 1 of 9 selected – Don't see a subscription? Open Directory + Subscription settings

Filter by n... | documenta... | All resource... | All types | All locatio... | All t

61 items

NAME	RESOURCE GROUP	LOCATION	SUBSCRIP...
mystorage0207	mystorage0207rg	Central US	documentati...
mytempstorage	mydatabricksresgrp	East US 2	documentati...

A storage account looks like:



Resource group (change)
mystorage0207rg

Status
Primary: Available, Secondary: Available

Location
Central US, East US 2

Subscription (change)
<Azure subscription name>

Subscription ID
<Azure subscription ID>

Tags (change)
Click here to add tags

Performance/Access tier
Standard/Hot

Replication
Read-access geo-redundant storage (RA-GR...

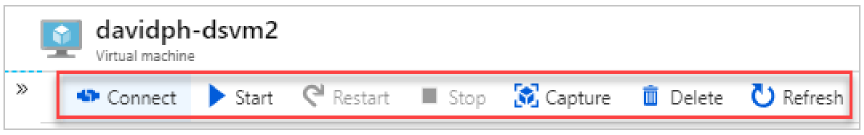
Account kind
StorageV2 (general purpose v2)

To open a resource by resource group:

1. Sign in to the Azure portal.
2. In the left pane, select **Resource groups** to list the resource within the group.
3. Select the resource you want to open.

Manage resources

When viewing a resource in the portal, you see the options for managing that particular resource.

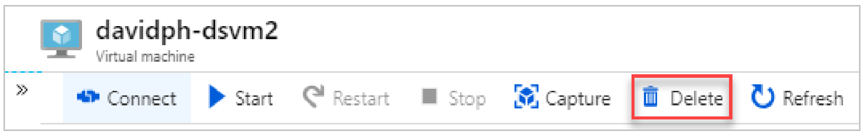


The screenshot shows the management options for an Azure virtual machine. You can perform operations such as starting, restarting, and stopping a virtual machine.

Delete resources

1. Open the resource in the portal. For the steps, see Open resources.

2. Select **Delete**. The following screenshot shows the management options for a virtual machine.



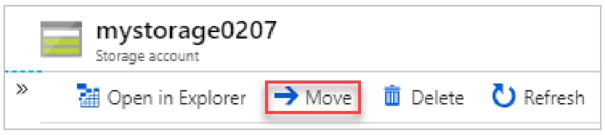
3. Type the name of the resource to confirm the deletion, and then select **Delete**.

For more information about how Azure Resource Manager orders the deletion of resources, see Azure Resource Manager resource group deletion.

Move resources

1. Open the resource in the portal. For the steps, see Open resources.

2. Select **Move**. The following screenshot shows the management options for a storage account.



3. Select **Move to another resource group** or **Moeve to another subscription** depending on your needs.

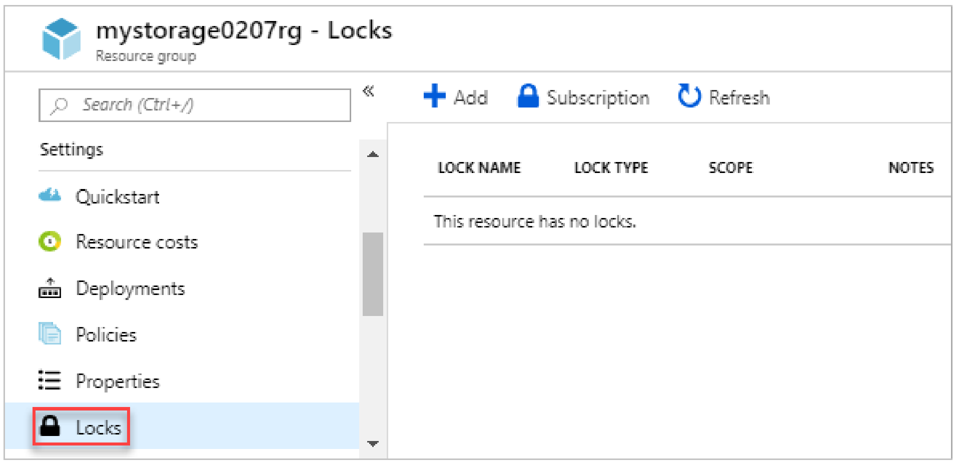
For more information, see Move resources to new resource group or subscription.

Lock resources

Locking prevents other users in your organization from accidentally deleting or modifying critical resources, such as Azure subscription, resource group, or resource.

1. Open the resource in the portal. For the steps, see Open resources.

2. Select **Locks**. The following screenshot shows the management options for a storage account.



This resource has no locks.

3. Select **Add**, and then specify the lock properties.

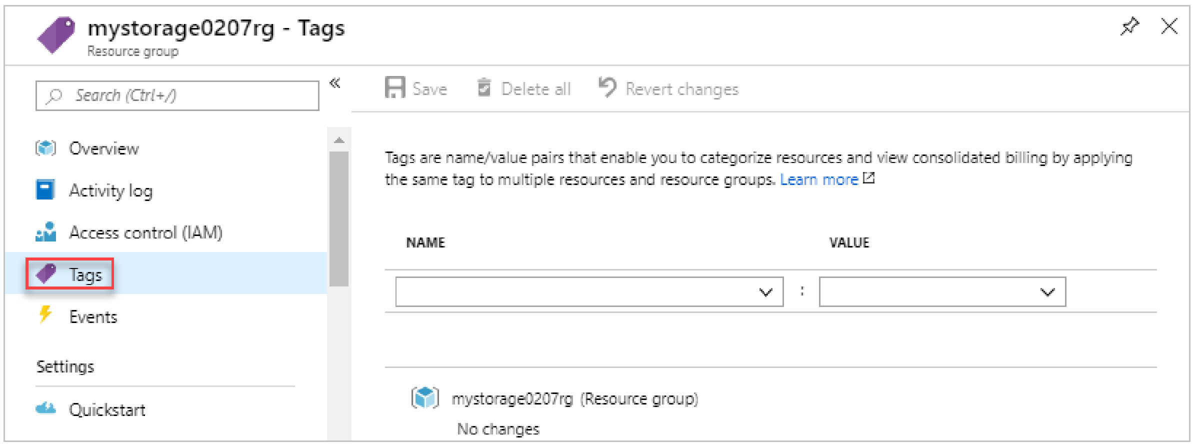
For more information, see Lock resources with Azure Resource Manager.

Tag resources

Tagging helps organizing your resource group and resources logically.

1. Open the resource in the portal. For the steps, see Open resources.

2. Select **Tags**. The following screenshot shows the management options for a storage account.



Tags are name/value pairs that enable you to categorize resources and view consolidated billing by applying the same tag to multiple resources and resource groups. Learn more

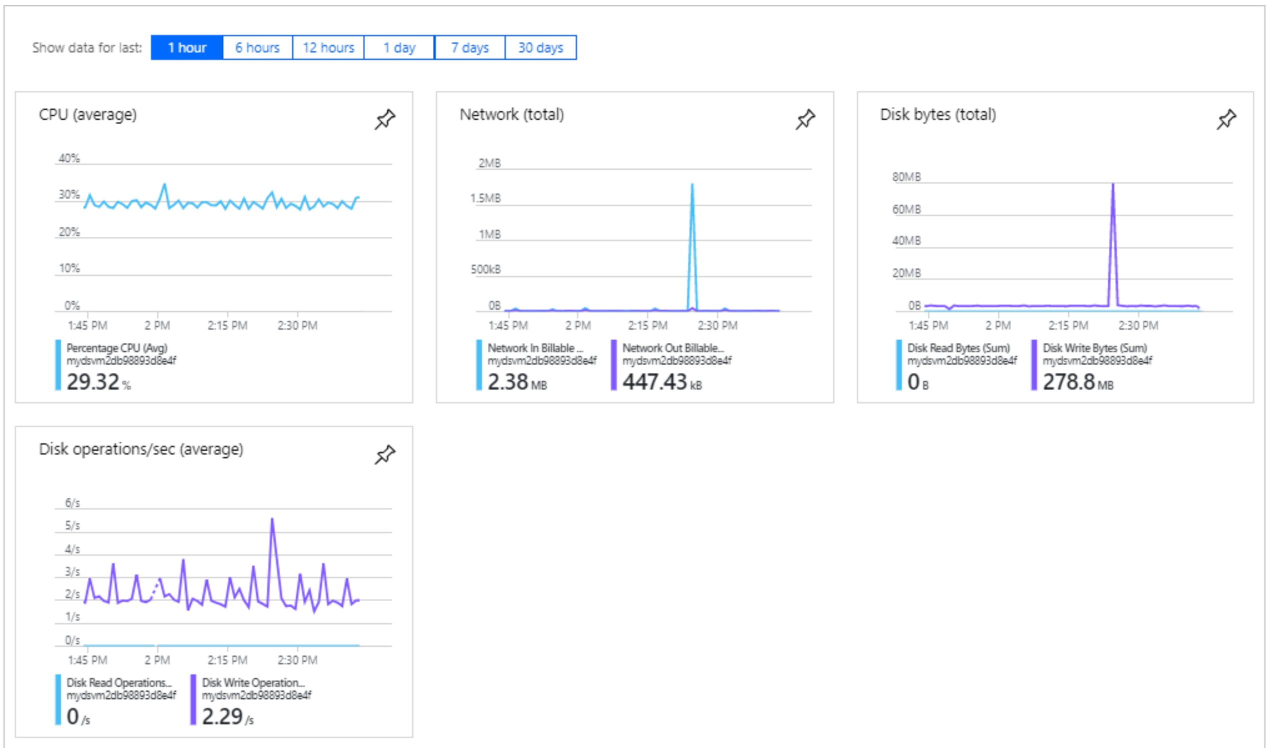
mystorage0207rg (Resource group)
No changes

3. Specify the tag properties, and then select **Save**.

For information, see Using tags to organize your Azure resources.

Monitor resources

When you open a resource, the portal presents default graphs and tables for monitoring that resource type. The following screenshot shows the graphs for a virtual machine:


Show data for last:
1 hour
6 hours
12 hours
1 day
7 days
30 days
CPU (average)
Network (total)
Disk bytes (total)
Disk operations/sec (average)
Percentage CPU (Avg) mydsvm2db98893d8e4f 29.32 %
Network In Billable ... mydsvm2db98893d8e4f 2.38 MB
Network Out Billable... mydsvm2db98893d8e4f 447.43 kB
Disk Read Bytes (Sum) mydsvm2db98893d8e4f 0 B
Disk Write Bytes (Sum) mydsvm2db98893d8e4f 278.8 MB
Disk Read Operations... mydsvm2db98893d8e4f 0 /s
Disk Write Operation... mydsvm2db98893d8e4f 2.29 /s

You can select the pin icon on the upper right corner of the graphs to pin the graph to the dashboard. To learn about working with dashboards, see Creating and sharing dashboards in the Azure portal.

Manage access to resources

Role-based access control (RBAC) is the way that you manage access to resources in Azure. For more information, see Manage access using RBAC and the Azure portal.

Next steps

- To learn Azure Resource Manager, see Azure Resource Manager overview.
- To learn the Resource Manager template syntax, see Understand the structure and syntax of Azure Resource Manager templates.

- To learn how to develop templates, see the step-by-step tutorials.
- To view the Azure Resource Manager template schemas, see template reference.

Manage Azure resources by using Azure CLI

6/18/2019 • 3 minutes to read • Edit Online

Learn how to use Azure CLI with Azure Resource Manager to manage your Azure resources. For managing resource groups, see Manage Azure resource groups by using Azure CLI.

Other articles about managing resources:

- Manage Azure resources by using the Azure portal
- Manage Azure resources by using Azure PowerShell

Deploy resources to an existing resource group

You can deploy Azure resources directly by using Azure PowerShell, or deploy a Resource Manager template to create Azure resources.

Deploy a resource

The following script creates a storage account.

```
echo "Enter the Resource Group name:" &&
read resourceGroupName &&
echo "Enter the location (i.e. centralus):" &&
read location &&
echo "Enter the storage account name:" &&
read storageAccountName &&
az storage account create --resource-group $resourceGroupName --name $storageAccountName --location $location
--sku Standard_LRS --kind StorageV2 &&
az storage account show --resource-group $resourceGroupName --name $storageAccountName
```

Deploy a template

The following script creates deploy a Quickstart template to create a storage account. For more information, see Quickstart: Create Azure Resource Manager templates by using Visual Studio Code.

```
echo "Enter the Resource Group name:" &&
read resourceGroupName &&
echo "Enter the location (i.e. centralus):" &&
read location &&
az group deployment create --resource-group $resourceGroupName --template-uri
"https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-storage-account-
create/azuredeploy.json"
```

For more information, see Deploy resources with Resource Manager templates and Azure CLI.

Deploy a resource group and resources

You can create a resource group and deploy resources to the group. For more information, see Create resource group and deploy resources.

Deploy resources to multiple subscriptions or resource groups

Typically, you deploy all the resources in your template to a single resource group. However, there are scenarios where you want to deploy a set of resources together but place them in different resource groups or subscriptions. For more information, see Deploy Azure resources to multiple subscriptions or resource groups.

Delete resources

The following script shows how to delete a storage account.

```
echo "Enter the Resource Group name:" &&
read resourceGroupName &&
echo "Enter the storage account name:" &&
read storageAccountName &&
az storage account delete --resource-group $resourceGroupName --name $storageAccountName
```

For more information about how Azure Resource Manager orders the deletion of resources, see Azure Resource Manager resource group deletion.

Move resources

The following script shows how to remove a storage account from one resource group to another resource group.

```
echo "Enter the source Resource Group name:" &&
read srcResourceGroupName &&
echo "Enter the destination Resource Group name:" &&
read destResourceGroupName &&
echo "Enter the storage account name:" &&
read storageAccountName &&
storageAccount=$(az resource show --resource-group $srcResourceGroupName --name $storageAccountName --
resource-type Microsoft.Storage/storageAccounts --query id --output tsv) &&
az resource move --destination-group $destResourceGroupName --ids $storageAccount
```

For more information, see Move resources to new resource group or subscription.

Lock resources

Locking prevents other users in your organization from accidentally deleting or modifying critical resources, such as Azure subscription, resource group, or resource.

The following script locks a storage account so the account can't be deleted.

```
echo "Enter the Resource Group name:" &&
read resourceGroupName &&
echo "Enter the storage account name:" &&
read storageAccountName &&
az lock create --name LockSite --lock-type CanNotDelete --resource-group $resourceGroupName --resource-name
$storageAccountName --resource-type Microsoft.Storage/storageAccounts
```

The following script gets all locks for a storage account:

```
echo "Enter the Resource Group name:" &&
read resourceGroupName &&
echo "Enter the storage account name:" &&
read storageAccountName &&
az lock list --resource-group $resourceGroupName --resource-name $storageAccountName --resource-type
Microsoft.Storage/storageAccounts --parent ""
```

The following script deletes a lock of a storage account:

```
echo "Enter the Resource Group name:" &&
read resourceGroupName &&
echo "Enter the storage account name:" &&
read storageAccountName &&
lockId=$(az lock show --name LockSite --resource-group $resourceGroupName --resource-type
Microsoft.Storage/storageAccounts --resource-name $storageAccountName --output tsv --query id)&&
az lock delete --ids $lockId
```

For more information, see Lock resources with Azure Resource Manager.

Tag resources

Tagging helps organizing your resource group and resources logically. For information, see Using tags to organize your Azure resources.

Manage access to resources

Role-based access control (RBAC) is the way that you manage access to resources in Azure. For more information, see Manage access using RBAC and Azure CLI.

Next steps

- To learn Azure Resource Manager, see Azure Resource Manager overview.
- To learn the Resource Manager template syntax, see Understand the structure and syntax of Azure Resource Manager templates.
- To learn how to develop templates, see the step-by-step tutorials.
- To view the Azure Resource Manager template schemas, see template reference.

Manage Azure resources by using Azure PowerShell

6/18/2019 • 3 minutes to read • Edit Online

Learn how to use Azure PowerShell with Azure Resource Manager to manage your Azure resources. For managing resource groups, see Manage Azure resource groups by using Azure PowerShell.

Other articles about managing resources:

- Manage Azure resources by using the Azure portal
- Manage Azure resources by using Azure CLI

Deploy resources to an existing resource group

You can deploy Azure resources directly by using Azure PowerShell, or deploy a Resource Manager template to create Azure resources.

Deploy a resource

The following script creates a storage account.

```
$resourceGroupName = Read-Host -Prompt "Enter the Resource Group name"
$location = Read-Host -Prompt "Enter the location (i.e. centralus)"
$storageAccountName = Read-Host -Prompt "Enter the storage account name"

# Create the storage account.
$storageAccount = New-AzStorageAccount -ResourceGroupName $resourceGroupName `
  -Name $storageAccountName `
  -Location $location `
  -SkuName "Standard_LRS"

# Retrieve the context.
$ctx = $storageAccount.Context
```

Deploy a template

The following script creates deploy a Quickstart template to create a storage account. For more information, see Quickstart: Create Azure Resource Manager templates by using Visual Studio Code.

```
$resourceGroupName = Read-Host -Prompt "Enter the Resource Group name"
$location = Read-Host -Prompt "Enter the location (i.e. centralus)"
$templateUri = "https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-storage-account-create/azuredeploy.json"
New-AzResourceGroupDeployment -ResourceGroupName $resourceGroupName -TemplateUri $templateUri -Location $location
```

For more information, see Deploy resources with Resource Manager templates and Azure PowerShell.

Deploy a resource group and resources

You can create a resource group and deploy resources to the group. For more information, see Create resource group and deploy resources.

Deploy resources to multiple subscriptions or resource groups

Typically, you deploy all the resources in your template to a single resource group. However, there are scenarios where you want to deploy a set of resources together but place them in different resource groups or subscriptions. For more information, see Deploy Azure resources to multiple subscriptions or resource groups.

Delete resources

The following script shows how to delete a storage account.

```
$resourceGroupName = Read-Host -Prompt "Enter the Resource Group name"
$storageAccountName = Read-Host -Prompt "Enter the storage account name"

Remove-AzStorageAccount -ResourceGroupName $resourceGroupName -AccountName $storageAccountName
```

For more information about how Azure Resource Manager orders the deletion of resources, see Azure Resource Manager resource group deletion.

Move resources

The following script shows how to remove a storage account from one resource group to another resource group.

```
$srcResourceGroupName = Read-Host -Prompt "Enter the source Resource Group name"
$destResourceGroupName = Read-Host -Prompt "Enter the destination Resource Group name"
$storageAccountName = Read-Host -Prompt "Enter the storage account name"

$storageAccount = Get-AzResource -ResourceGroupName $srcResourceGroupName -ResourceName $storageAccountName
Move-AzResource -DestinationResourceGroupName $destResourceGroupName -ResourceId $storageAccount.ResourceId
```

For more information, see Move resources to new resource group or subscription.

Lock resources

Locking prevents other users in your organization from accidentally deleting or modifying critical resources, such as Azure subscription, resource group, or resource.

The following script locks a storage account so the account can't be deleted.

```
$resourceGroupName = Read-Host -Prompt "Enter the Resource Group name"
$storageAccountName = Read-Host -Prompt "Enter the storage account name"

New-AzResourceLock -LockName LockStorage -LockLevel CanNotDelete -ResourceGroupName $resourceGroupName -
ResourceName $storageAccountName -ResourceType Microsoft.Storage/storageAccounts
```

The following script gets all locks for a storage account:

```
$resourceGroupName = Read-Host -Prompt "Enter the Resource Group name"
$storageAccountName = Read-Host -Prompt "Enter the storage account name"

Get-AzResourceLock -ResourceGroupName $resourceGroupName -ResourceName $storageAccountName -ResourceType
Microsoft.Storage/storageAccounts
```

The following script deletes a lock of a storage account:

```
$resourceGroupName = Read-Host -Prompt "Enter the Resource Group name"
$storageAccountName = Read-Host -Prompt "Enter the storage account name"

$lockId = (Get-AzResourceLock -ResourceGroupName $resourceGroupName -ResourceName $storageAccountName -
ResourceType Microsoft.Storage/storageAccounts).LockId
Remove-AzResourceLock -LockId $lockId
```

For more information, see Lock resources with Azure Resource Manager.

Tag resources

Tagging helps organizing your resource group and resources logically. For information, see Using tags to organize your Azure resources.

Manage access to resources

Role-based access control (RBAC) is the way that you manage access to resources in Azure. For more information, see Manage access using RBAC and Azure PowerShell.

Next steps

- To learn Azure Resource Manager, see Azure Resource Manager overview.
- To learn the Resource Manager template syntax, see Understand the structure and syntax of Azure Resource Manager templates.
- To learn how to develop templates, see the step-by-step tutorials.
- To view the Azure Resource Manager template schemas, see template reference.

Lock resources to prevent unexpected changes

6/17/2019 • 6 minutes to read • Edit Online

As an administrator, you may need to lock a subscription, resource group, or resource to prevent other users in your organization from accidentally deleting or modifying critical resources. You can set the lock level to **CanNotDelete** or **ReadOnly**. In the portal, the locks are called **Delete** and **Read-only** respectively.

- **CanNotDelete** means authorized users can still read and modify a resource, but they can't delete the resource.
- **ReadOnly** means authorized users can read a resource, but they can't delete or update the resource. Applying this lock is similar to restricting all authorized users to the permissions granted by the **Reader** role.

> **NOTE**
> This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

How locks are applied

When you apply a lock at a parent scope, all resources within that scope inherit the same lock. Even resources you add later inherit the lock from the parent. The most restrictive lock in the inheritance takes precedence.

Unlike role-based access control, you use management locks to apply a restriction across all users and roles. To learn about setting permissions for users and roles, see Azure Role-based Access Control.

Resource Manager locks apply only to operations that happen in the management plane, which consists of operations sent to `https://management.azure.com`. The locks don't restrict how resources perform their own functions. Resource changes are restricted, but resource operations aren't restricted. For example, a ReadOnly lock on a SQL Database prevents you from deleting or modifying the database. It doesn't prevent you from creating, updating, or deleting data in the database. Data transactions are permitted because those operations aren't sent to `https://management.azure.com`.

Applying **ReadOnly** can lead to unexpected results because some operations that don't seem to modify the resource actually require actions that are blocked by the lock. The **ReadOnly** lock can be applied to the resource or to the resource group containing the resource. Some common examples of the operations that are blocked by a **ReadOnly** lock are:

- A **ReadOnly** lock on a storage account prevents all users from listing the keys. The list keys operation is handled through a POST request because the returned keys are available for write operations.
- A **ReadOnly** lock on an App Service resource prevents Visual Studio Server Explorer from displaying files for the resource because that interaction requires write access.
- A **ReadOnly** lock on a resource group that contains a virtual machine prevents all users from starting or restarting the virtual machine. These operations require a POST request.

Who can create or delete locks

To create or delete management locks, you must have access to `Microsoft.Authorization/*` or

`Microsoft.Authorization/locks/*` actions. Of the built-in roles, only **Owner** and **User Access Administrator** are granted those actions.

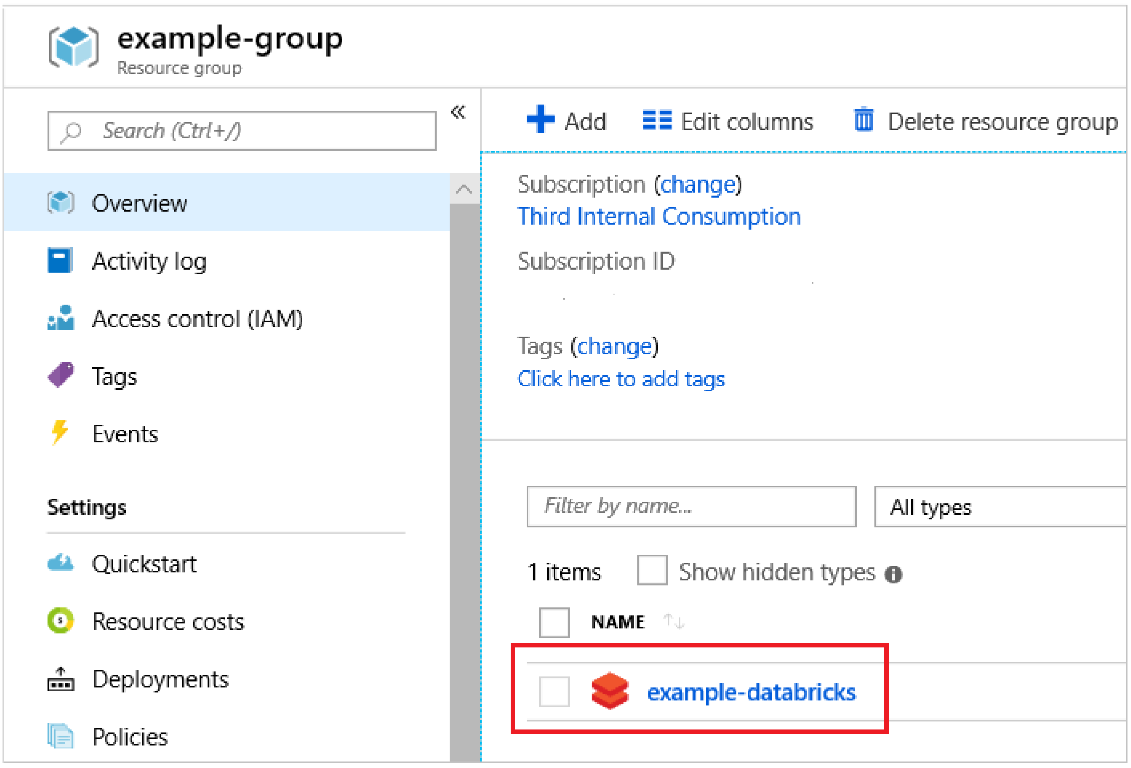
Managed Applications and locks

Some Azure services, such as Azure Databricks, use managed applications to implement the service. In that case, the service creates two resource groups. One resource group contains an overview of the service and isn't locked. The other resource group contains the infrastructure for the service and is locked.

If you try to delete the infrastructure resource group, you get an error stating that the resource group is locked. If you try to delete the lock for the infrastructure resource group, you get an error stating that the lock can't be deleted because it's owned by a system application.

Instead, delete the service, which also deletes the infrastructure resource group.

For managed applications, select the service you deployed.



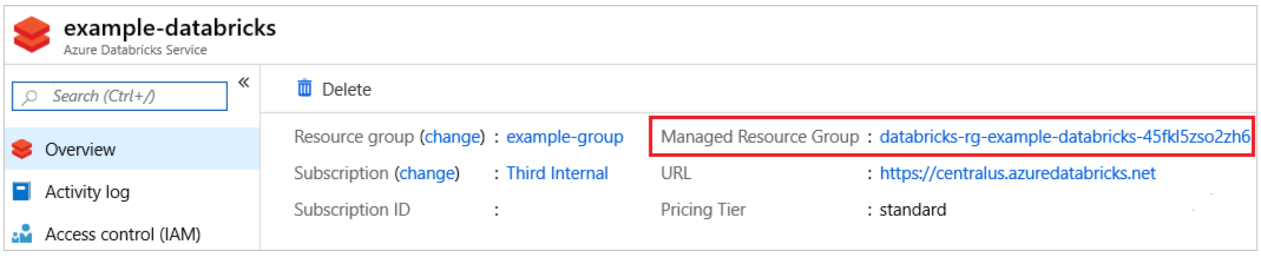
Subscription (change)
Third Internal Consumption
Subscription ID

Tags (change)
Click here to add tags

Filter by name...
All types
1 items
Show hidden types

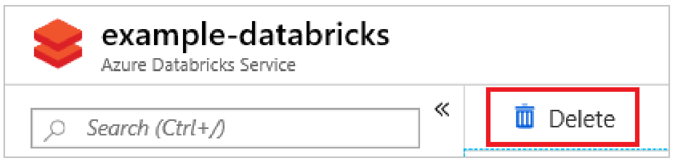
NAME
example-databricks

Notice the service includes a link for a **Managed Resource Group**. That resource group holds the infrastructure and is locked. It can't be directly deleted.



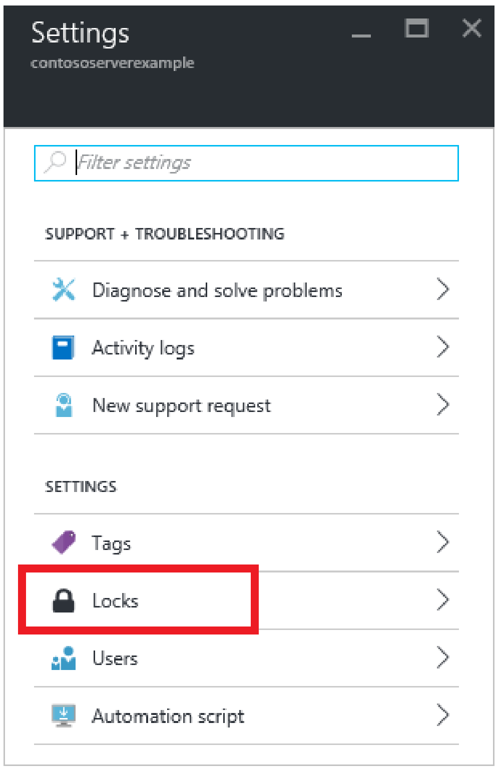
Resource group (change)	: example-group	Managed Resource Group	: databricks-rg-example-databricks-45fkl5zso2zh6
Subscription (change)	: Third Internal	URL	: https://centralus.azuredatabricks.net
Subscription ID	:	Pricing Tier	: standard

To delete everything for the service, including the locked infrastructure resource group, select **Delete** for the service.

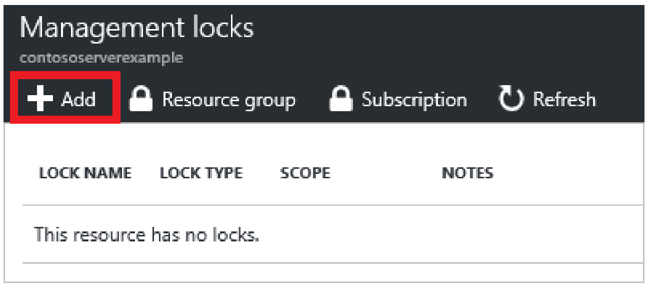


Portal

1. In the Settings blade for the resource, resource group, or subscription that you wish to lock, select **Locks**.

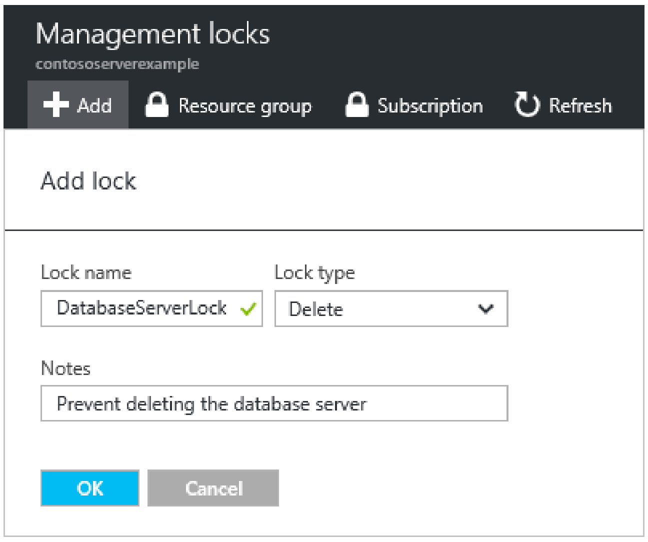


2. To add a lock, select **Add**. If you want to create a lock at a parent level, select the parent. The currently selected resource inherits the lock from the parent. For example, you could lock the resource group to apply a lock to all its resources.



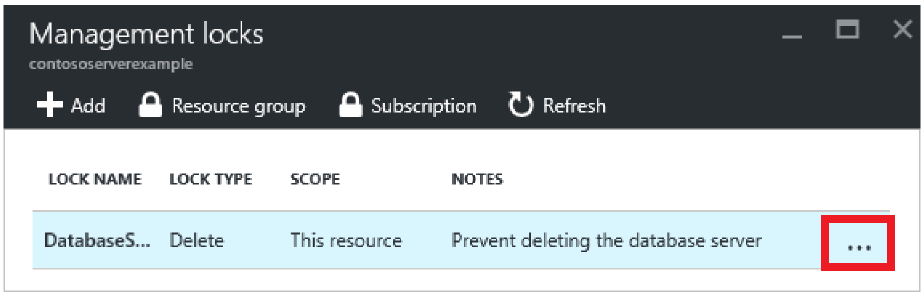
LOCK NAME	LOCK TYPE	SCOPE	NOTES
This resource has no locks.			

3. Give the lock a name and lock level. Optionally, you can add notes that describe the lock.



Lock name: DatabaseServerLock
Lock type: Delete
Notes: Prevent deleting the database server

4. To delete the lock, select the ellipsis and **Delete** from the available options.



LOCK NAME	LOCK TYPE	SCOPE	NOTES
DatabaseS...	Delete	This resource	Prevent deleting the database server

Template

When using a Resource Manager template to deploy a lock, you use different values for the name and type depending on the scope of the lock.

When applying a lock to a **resource**, use the following formats:

- name - `{resourceName}/Microsoft.Authorization/{lockName}`
- type - `{resourceProviderNamespace}/{resourceType}/providers/locks`

When applying a lock to a **resource group** or **subscription**, use the following formats:

- name - `{lockName}`
- type - `Microsoft.Authorization/locks`

The following example shows a template that creates an app service plan, a web site, and a lock on the web site. The resource type of the lock is the resource type of the resource to lock and **/providers/locks**. The name of the lock is created by concatenating the resource name with **/Microsoft.Authorization/** and the name of the lock.

```json
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "hostingPlanName": {
            "type": "string"
        }
    },
    "variables": {
        "siteName": "[concat('ExampleSite', uniqueString(resourceGroup().id))]"
    },
    "resources": [
        {
            "apiVersion": "2016-09-01",
            "type": "Microsoft.Web/serverfarms",
            "name": "[parameters('hostingPlanName')]",
            "location": "[resourceGroup().location]",
            "sku": {
                "tier": "Free",
                "name": "f1",
                "capacity": 0
            },
            "properties": {
                "targetWorkerCount": 1
            }
        },
        {
            "apiVersion": "2016-08-01",
            "name": "[variables('siteName')]",
            "type": "Microsoft.Web/sites",
            "location": "[resourceGroup().location]",
            "dependsOn": [
                "[resourceId('Microsoft.Web/serverfarms', parameters('hostingPlanName'))]"
            ],
            "properties": {
                "serverFarmId": "[parameters('hostingPlanName')]"
            }
        },
        {
            "type": "Microsoft.Web/sites/providers/locks",
            "apiVersion": "2016-09-01",
            "name": "[concat(variables('siteName'), '/Microsoft.Authorization/siteLock')]",
            "dependsOn": [
                "[resourceId('Microsoft.Web/sites', variables('siteName'))]"
            ],
            "properties": {
                "level": "CanNotDelete",
                "notes": "Site should not be deleted."
            }
        }
    ]
}
```

For an example of setting a lock on a resource group, see Create a resource group and lock it.

PowerShell

You lock deployed resources with Azure PowerShell by using the New-AzResourceLock command.

To lock a resource, provide the name of the resource, its resource type, and its resource group name.

```
New-AzResourceLock -LockLevel CanNotDelete -LockName LockSite -ResourceName examplesite -ResourceType
Microsoft.Web/sites -ResourceGroupName exampleresourcegroup
```

To lock a resource group, provide the name of the resource group.

```
New-AzResourceLock -LockName LockGroup -LockLevel CanNotDelete -ResourceGroupName exampleresourcegroup
```

To get information about a lock, use Get-AzResourceLock. To get all the locks in your subscription, use:

```
Get-AzResourceLock
```

To get all locks for a resource, use:

```
Get-AzResourceLock -ResourceName examplesite -ResourceType Microsoft.Web/sites -ResourceGroupName
exampleresourcegroup
```

To get all locks for a resource group, use:

```
Get-AzResourceLock -ResourceGroupName exampleresourcegroup
```

To delete a lock, use:

```
$lockId = (Get-AzResourceLock -ResourceGroupName exampleresourcegroup -ResourceName examplesite -ResourceType
Microsoft.Web/sites).LockId
Remove-AzResourceLock -LockId $lockId
```

Azure CLI

You lock deployed resources with Azure CLI by using the az lock create command.

To lock a resource, provide the name of the resource, its resource type, and its resource group name.

```
az lock create --name LockSite --lock-type CanNotDelete --resource-group exampleresourcegroup --resource-name
examplesite --resource-type Microsoft.Web/sites
```

To lock a resource group, provide the name of the resource group.

```
az lock create --name LockGroup --lock-type CanNotDelete --resource-group exampleresourcegroup
```

To get information about a lock, use az lock list. To get all the locks in your subscription, use:

```
az lock list
```

To get all locks for a resource, use:

```
az lock list --resource-group exampleresourcegroup --resource-name examplesite --namespace Microsoft.Web --
resource-type sites --parent ""
```

To get all locks for a resource group, use:

```
az lock list --resource-group exampleresourcegroup
```

To delete a lock, use:

```
lockid=$(az lock show --name LockSite --resource-group exampleresourcegroup --resource-type
Microsoft.Web/sites --resource-name examplesite --output tsv --query id)
az lock delete --ids $lockid
```

REST API

You can lock deployed resources with the REST API for management locks. The REST API enables you to create and delete locks, and retrieve information about existing locks.

To create a lock, run:

```
PUT https://management.azure.com/{scope}/providers/Microsoft.Authorization/locks/{lock-name}?api-version=
{api-version}
```

The scope could be a subscription, resource group, or resource. The lock-name is whatever you want to call the lock. For api-version, use **2016-09-01**.

In the request, include a JSON object that specifies the properties for the lock.

```
{
  "properties": {
    "level": "CanNotDelete",
    "notes": "Optional text notes."
  }
}
```

Next steps

- To learn about logically organizing your resources, see Using tags to organize your resources
- You can apply restrictions and conventions across your subscription with customized policies. For more information, see What is Azure Policy?.
- For guidance on how enterprises can use Resource Manager to effectively manage subscriptions, see Azure enterprise scaffold - prescriptive subscription governance.

Use Resource Manager authentication API to access subscriptions

6/18/2019 • 14 minutes to read • Edit Online

If you're a software developer who needs to create an app that manages a customer's Azure resources, this article shows you how to authenticate with the Azure Resource Manager APIs and gain access to resources in other subscriptions.

Your app can access the Resource Manager APIs in couple of ways:

1. **User + app access**: for apps that access resources for a signed-in user. This approach works for apps, such as web apps and command-line tools, that deal with only "interactive management" of Azure resources.
2. **App-only access**: for apps that run daemon services and scheduled jobs. The app's identity is granted direct access to the resources. This approach works for apps that need long-term headless (unattended) access to Azure.

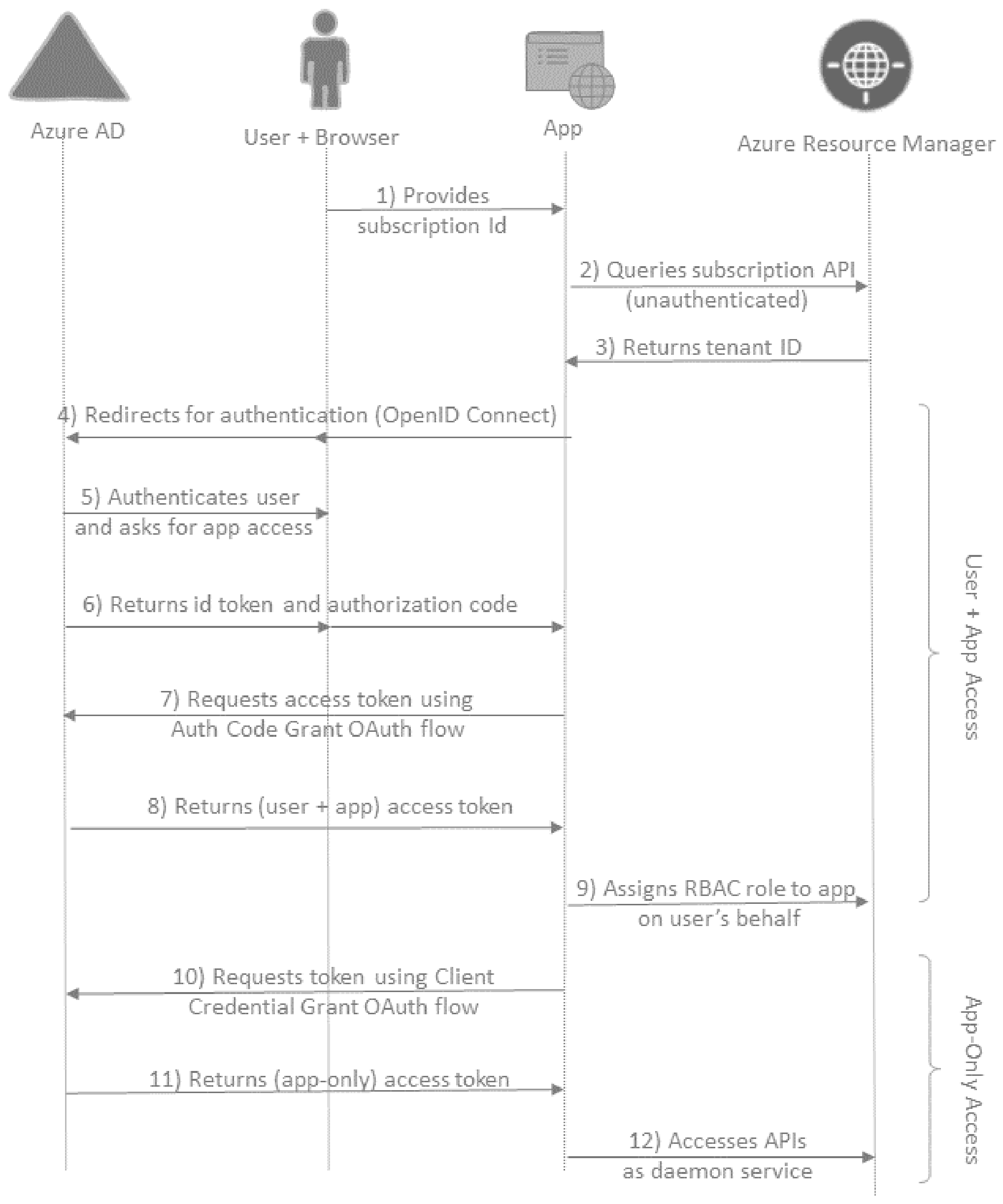
This article provides step-by-step instructions to create an app that employs both these authorization methods. It shows how to do each step with REST API or C#. The complete ASP.NET MVC application is available at https://github.com/dushyantgill/VipSwapper/tree/master/CloudSense.

What the web app does

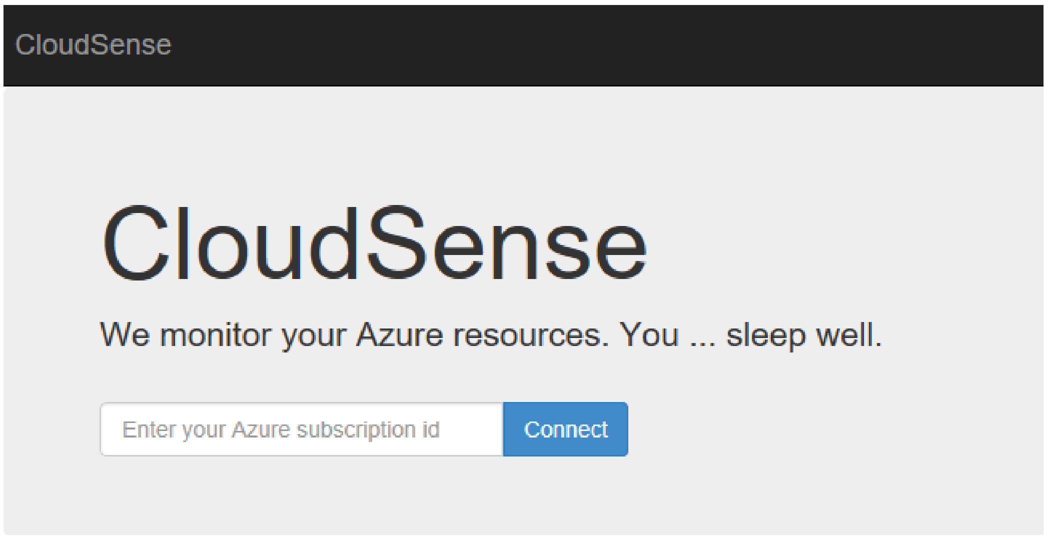
The web app:

1. Signs-in an Azure user.
2. Asks user to grant the web app access to Resource Manager.
3. Gets user + app access token for accessing Resource Manager.
4. Uses token (from step 3) to assign the app's service principal to a role in the subscription. This step gives the app long-term access to the subscription.
5. Gets app-only access token.
6. Uses token (from step 5) to manage resources in the subscription through Resource Manager.

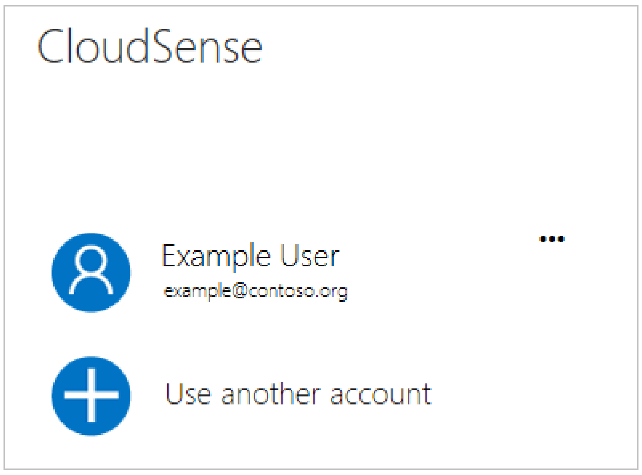
Here's the flow of the web application.


Azure AD
User + Browser
App
Azure Resource Manager
1) Provides subscription Id
2) Queries subscription API (unauthenticated)
3) Returns tenant ID
4) Redirects for authentication (OpenID Connect)
5) Authenticates user and asks for app access
6) Returns id token and authorization code
7) Requests access token using Auth Code Grant OAuth flow
8) Returns (user + app) access token
9) Assigns RBAC role to app on user's behalf
10) Requests token using Client Credential Grant OAuth flow
11) Returns (app-only) access token
12) Accesses APIs as daemon service
User + App Access
App-Only Access

As a user, you provide the subscription ID for the subscription you want to use:


CloudSense
CloudSense
We monitor your Azure resources. You ... sleep well.
Enter your Azure subscription id
Connect

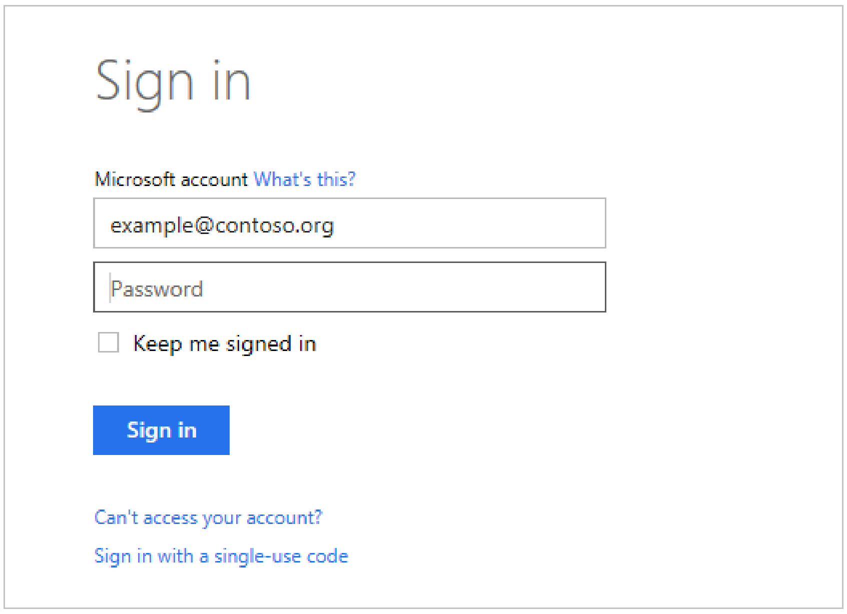
Select the account to use for logging in.


CloudSense

Example User
example@contoso.org

Use another account

Provide your credentials.


Sign in

Microsoft account What's this?

example@contoso.org

Password

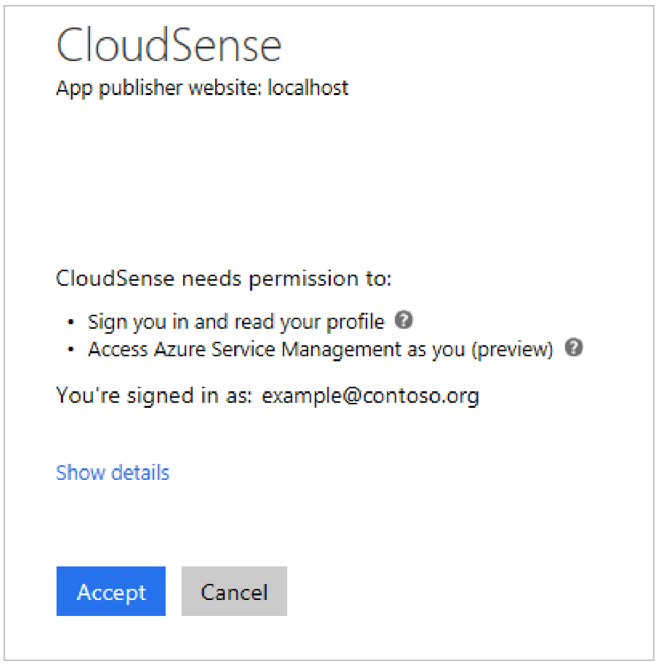
Keep me signed in

Sign in

Can't access your account?

Sign in with a single-use code

Grant the app access to your Azure subscriptions:


CloudSense
App publisher website: localhost

CloudSense needs permission to:

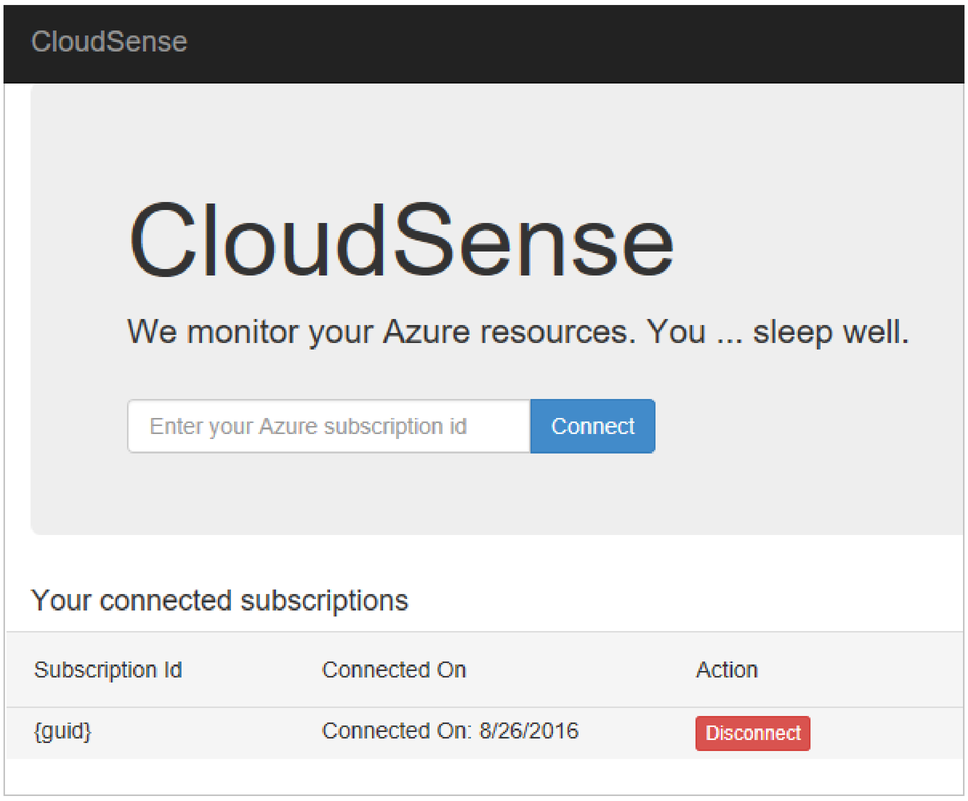
- Sign you in and read your profile
- Access Azure Service Management as you (preview)

You're signed in as: example@contoso.org

Show details

Accept Cancel

Manage your connected subscriptions:


CloudSense
CloudSense
We monitor your Azure resources. You ... sleep well.
Enter your Azure subscription id
Connect
Your connected subscriptions

Subscription Id	Connected On	Action
{guid}	Connected On: 8/26/2016	Disconnect

Register application

Before you start coding, register your web app with Azure Active Directory (AD). The app registration creates a central identity for your app in Azure AD. It holds basic information about your application like OAuth Client ID, Reply URLs, and credentials that your application uses to authenticate and access Azure Resource Manager APIs. The app registration also records the various delegated permissions that your application needs when accessing Microsoft APIs for the user.

To register your app, see Quickstart: Register an application with the Microsoft identity platform. Give your app a name, and select **Accounts in any organizational directory** for the supported account types. For redirect URL, provide a domain associated with your Azure Active Directory.

To sign in as the AD application, you need the application ID and a secret. The application ID is displayed in the overview for the application. To create a secret and request API permissions, see Quickstart: Configure a client application to access web APIs. Provide a new client secret. For API permissions, select **Azure Service Management**. Select **Delegated permissions** and **user_impersonation**.

Optional configuration - certificate credential

Azure AD also supports certificate credentials for applications: you create a self-signed cert, keep the private key, and add the public key to your Azure AD application registration. For authentication, your application sends a small payload to Azure AD signed using your private key, and Azure AD validates the signature using the public key that you registered.

For information about creating an AD app with a certificate, see Use Azure PowerShell to create a service principal to access resources or Use Azure CLI to create a service principal to access resources.

Get tenant ID from subscription ID

To request a token that can be used to call Resource Manager, your application needs to know the tenant ID of the Azure AD tenant that hosts the Azure subscription. Most likely, your users know their subscription IDs, but they might not know their tenant IDs for Azure Active Directory. To get the user's tenant ID, ask the user for the subscription ID. Provide that subscription ID when sending a request about the subscription:

```
https://management.azure.com/subscriptions/{subscription-id}?api-version=2015-01-01
```

The request fails because the user hasn't logged in yet, but you can retrieve the tenant ID from the response. In that exception, retrieve the tenant ID from the response header value for **WWW-Authenticate**. You see this implementation in the GetDirectoryForSubscription method.

Get user + app access token

Your application redirects the user to Azure AD with an OAuth 2.0 Authorize Request - to authenticate the user's credentials and get back an authorization code. Your application uses the authorization code to get an access token for Resource Manager. The ConnectSubscription method creates the authorization request.

This article shows the REST API requests to authenticate the user. You can also use helper libraries to authenticate in your code. For more information about these libraries, see Azure Active Directory Authentication Libraries. For guidance on integrating identity management in an application, see Azure Active Directory developer's guide.

Auth request (OAuth 2.0)

Issue an Open ID Connect/OAuth2.0 Authorize Request to the Azure AD Authorize endpoint:

```
https://login.microsoftonline.com/{tenant-id}/OAuth2/Authorize
```

The query string parameters that are available for this request are described in the request an authorization code article.

The following example shows how to request OAuth2.0 authorization:

```
https://login.microsoftonline.com/{tenant-id}/OAuth2/Authorize?client_id=a0448380-c346-4f9f-b897-
c18733de9394&response_mode=query&response_type=code&redirect_uri=http%3a%2f%2fwww.vipswapper.com%2fcloudsense%2
fAccount%2fSignIn&resource=https%3a%2f%2fgraph.windows.net%2f&domain_hint=live.com
```

Azure AD authenticates the user, and, if necessary, asks the user to grant permission to the app. It returns the authorization code to the Reply URL of your application. Depending on the requested response_mode, Azure AD either sends back the data in query string or as post data.

```
code=AAABAAAAiL****FDMZBUwZ8eCAA&session_state=2d16bbce-d5d1-443f-acdf-75f6b0ce8850
```

Auth request (Open ID Connect)

If you not only wish to access Azure Resource Manager for the user, but also allow the user to sign in to your application using their Azure AD account, issue an Open ID Connect Authorize Request. With Open ID Connect, your application also receives an id_token from Azure AD that your app can use to sign in the user.

The query string parameters that are available for this request are described in the Send the sign-in request article.

An example Open ID Connect request is:

```
 https://login.microsoftonline.com/{tenant-id}/OAuth2/Authorize?client_id=a0448380-c346-4f9f-b897-
c18733de9394&response_mode=form_post&response_type=code+id_token&redirect_uri=http%3a%2f%2fwww.vipswapper.com%2
fcloudsense%2fAccount%2fSignIn&resource=https%3a%2f%2fgraph.windows.net%2f&scope=openid+profile&nonce=63567Dc4M
DAw&domain_hint=live.com&state=M_12tMyKaM8
```

Azure AD authenticates the user, and, if necessary, asks the user to grant permission to the app. It returns the authorization code to the Reply URL of your application. Depending on the requested response_mode, Azure AD either sends back the data in query string or as post data.

An example Open ID Connect response is:

```
code=AAABAAAAiL*****I4rDWd7zXsH6WUjlkIEQxIAA&id_token=eyJ0eXAiOiJKV1Q*****T3GrzzSFxg&state=M_12tMyKaM8&session_
state=2d16bbce-d5d1-443f-acdf-75f6b0ce8850
```

Token request (OAuth2.0 Code Grant Flow)

Now that your application has received the authorization code from Azure AD, it's time to get the access token for Azure Resource Manager. Post an OAuth2.0 Code Grant Token Request to the Azure AD Token endpoint:

```
https://login.microsoftonline.com/{tenant-id}/OAuth2/Token
```

The query string parameters that are available for this request are described in the use the authorization code article.

The following example shows a request for code grant token with password credential:

```
POST https://login.microsoftonline.com/7fe877e6-a150-4992-bbfe-f517e304dfa0/oauth2/token HTTP/1.1

Content-Type: application/x-www-form-urlencoded
Content-Length: 1012

grant_type=authorization_code&code=AAABAAAAiL9Kn2Z*****L1nVMH3Z5ESiAA&redirect_uri=http%3A%2F%2Flocalhost%3A620
80%2FAccount%2FSignIn&client_id=a0448380-c346-4f9f-b897-c18733de9394&client_secret=olna84E8*****goScOg%3D
```

When working with certificate credentials, create a JSON Web Token (JWT) and sign (RSA SHA256) using the private key of your application's certificate credential. Building this token is shown in the [client credential flow](../active-directory/develop/v1-oauth2-client-creds-grant-flow.md#second-case-access-token-request-with a-certificate). For reference, see the Active Directory Auth Library (.NET) code to sign Client Assertion JWT tokens.

See the Open ID Connect spec for details on client authentication.

The following example shows a request for code grant token with certificate credential:

```
POST https://login.microsoftonline.com/7fe877e6-a150-4992-bbfe-f517e304dfa0/oauth2/token HTTP/1.1

Content-Type: application/x-www-form-urlencoded
Content-Length: 1012

grant_type=authorization_code&code=AAABAAAAiL9Kn2Z*****L1nVMH3Z5ESiAA&redirect_uri=http%3A%2F%2Flocalhost%3A620
80%2FAccount%2FSignIn&client_id=a0448380-c346-4f9f-b897-
c18733de9394&client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-
bearer&client_assertion=eyJhbG*****Y9cYo8nEjMyA
```

An example response for code grant token:

```
HTTP/1.1 200 OK

{"token_type":"Bearer","expires_in":"3599","expires_on":"1432039858","not_before":"1432035958","resource":"http
s://management.core.windows.net/","access_token":"eyJ0eXAiOiJKV1Q****M7Cw6JWtfY2lGc5A","refresh_token":"AAABAAA
AiL9Kn2Z****55j-sjnyYgAA","scope":"user_impersonation","id_token":"eyJ0eXAiOiJKV*****-drP1J3P-
HnHi9Rr46kGZnukEBH4dsg"}
```

Handle code grant token response

A successful token response contains the (user + app) access token for Azure Resource Manager. Your application uses this access token to access Resource Manager for the user. The lifetime of access tokens issued by Azure AD is one hour. It's unlikely that your web application needs to renew the (user + app) access token. If it needs to renew the access token, use the refresh token that your application receives in the token response. Post an OAuth2.0 Token Request to the Azure AD Token endpoint:

```
https://login.microsoftonline.com/{tenant-id}/OAuth2/Token
```

The parameters to use with the refresh request are described in refreshing the access token.

The following example shows how to use the refresh token:

```
POST https://login.microsoftonline.com/7fe877e6-a150-4992-bbfe-f517e304dfa0/oauth2/token HTTP/1.1

Content-Type: application/x-www-form-urlencoded
Content-Length: 1012

grant_type=refresh_token&refresh_token=AAABAAAAiL9Kn2Z****55j-sjnyYgAA&client_id=a0448380-c346-4f9f-b897-
c18733de9394&client_secret=olna84E8*****goScOg%3D
```

Although refresh tokens can be used to get new access tokens for Azure Resource Manager, they aren't suitable for offline access by your application. The refresh tokens lifetime is limited, and refresh tokens are bound to the user. If the user leaves the organization, the application using the refresh token loses access. This approach isn't suitable for applications that are used by teams to manage their Azure resources.

Check if user can assign access to subscription

Your application now has a token to access Azure Resource Manager for the user. The next step is to connect your app to the subscription. After connecting, your app can manage those subscriptions even when the user isn't present (long-term offline access).

For each subscription to connect, call the Resource Manager list permissions API to determine whether the user has access management rights for the subscription.

The UserCanManagerAccessForSubscription method of the ASP.NET MVC sample app implements this call.

The following example shows how to request a user's permissions on a subscription. 83cfe939-2402-4581-b761-4f59b0a041e4 is the ID of the subscription.

```
GET https://management.azure.com/subscriptions/83cfe939-2402-4581-b761-
4f59b0a041e4/providers/microsoft.authorization/permissions?api-version=2015-07-01 HTTP/1.1

Authorization: Bearer eyJ0eXAiOiJKV1QiLC***lwO1mM7Cw6JWtfY2lGc5A
```

An example of the response to get user's permissions on subscription is:

```
HTTP/1.1 200 OK

{"value":[{"actions":["*"],"notActions":
["Microsoft.Authorization/*/Write","Microsoft.Authorization/*/Delete"]},{"actions":["*/read"],"notActions":
[]}]}
```

The permissions API returns multiple permissions. Each permission consists of allowed actions (**actions**) and disallowed actions (**notactions**). If an action is present in the allowed actions of any permission and not present in the disallowed actions of that permission, the user is allowed to do that action. **microsoft.authorization/roleassignments/write** is the action that grants access management rights. Your application must parse the permissions result to look for a regex match on this action string in the **actions** and **notactions** of each permission.

Get app-only access token

Now, you know if the user can assign access to the Azure subscription. The next steps are:

1. Assign the appropriate RBAC role to your application's identity on the subscription.
2. Validate the access assignment by querying for the application's permission on the subscription or by accessing Resource Manager using app-only token.
3. Record the connection in your applications "connected subscriptions" data structure - persisting the ID of the subscription.

Let's look closer at the first step. To assign the appropriate RBAC role to the application's identity, you must determine:

- The object ID of your application's identity in the user's Azure Active Directory
- The identifier of the RBAC role that your application requires on the subscription

When your application authenticates a user from an Azure AD, it creates a service principal object for your application in that Azure AD. Azure allows RBAC roles to be assigned to service principals to grant direct access to corresponding applications on Azure resources. This action is exactly what you wish to do. Query the Azure AD Graph API to determine the identifier of the service principal of your application in the signed-in user's Azure AD.

You only have an access token for Azure Resource Manager - you need a new access token to call the Azure AD Graph API. Every application in Azure AD has permission to query its own service principal object, so an app-only access token is sufficient.

Get app-only access token for Azure AD Graph API

To authenticate your app and get a token to Azure AD Graph API, issue a Client Credential Grant OAuth2.0 flow token request to Azure AD token endpoint (**https://login.microsoftonline.com/{directory_domain_name}/OAuth2/Token**).

The GetObjectIdOfServicePrincipalInOrganization method of the ASP.net MVC sample application gets an app-only access token for Graph API using the Active Directory Authentication Library for .NET.

The query string parameters that are available for this request are described in the Request an Access Token article.

An example request for client credential grant token:

```
POST https://login.microsoftonline.com/62e173e9-301e-423e-bcd4-29121ec1aa24/oauth2/token HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 187</pre>
<pre>grant_type=client_credentials&client_id=a0448380-c346-4f9f-b897-
c18733de9394&resource=https%3A%2F%2Fgraph.windows.net%2F &client_secret=olna8C*****Og%3D
```

An example response for client credential grant token:

```
HTTP/1.1 200 OK

{"token_type":"Bearer","expires_in":"3599","expires_on":"1432039862","not_before":"1432035962","resource":"http
s://graph.windows.net/","access_token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Ik1uQ19WWmNBVGZNNXBPWWlKSE1
iYTlnb0VLWSIsImtpZCI6Ik1uQ19WWmNBVGZNNXBPWWlKSE1iYTlnb0VLWSJ9.eyJhdWQiOiJodHRwczovL2dyYXBoLndpbmRv****G5gUTV-
kKorR-pg"}
```

Get ObjectId of application service principal in user Azure AD

Now, use the app-only access token to query the Azure AD Graph Service Principals API to determine the Object ID of the application's service principal in the directory.

The GetObjectIdOfServicePrincipalInOrganization method of the ASP.net MVC sample application implements this call.

The following example shows how to request an application's service principal. a0448380-c346-4f9f-b897-c18733de9394 is the client ID of the application.

```
GET https://graph.windows.net/62e173e9-301e-423e-bcd4-29121ec1aa24/servicePrincipals?api-
version=1.5&$filter=appId%20eq%20'a0448380-c346-4f9f-b897-c18733de9394' HTTP/1.1

Authorization: Bearer eyJ0eXAiOiJK*****-kKorR-pg
```

The following example shows a response to the request for an application's service principal

```
HTTP/1.1 200 OK

{"odata.metadata":"https://graph.windows.net/62e173e9-301e-423e-bcd4-
29121ec1aa24/$metadata#directoryObjects/Microsoft.DirectoryServices.ServicePrincipal","value":
[{"odata.type":"Microsoft.DirectoryServices.ServicePrincipal","objectType":"ServicePrincipal","objectId":"9b501
8d4-6951-42ed-8a92-
f11ec283ccec","deletionTimestamp":null,"accountEnabled":true,"appDisplayName":"CloudSense","appId":"a0448380-
c346-4f9f-b897-c18733de9394","appOwnerTenantId":"62e173e9-301e-423e-bcd4-
29121ec1aa24","appRoleAssignmentRequired":false,"appRoles":
[],"displayName":"CloudSense","errorUrl":null,"homepage":"http://www.vipswapper.com/cloudsense","keyCredentials
":[],"logoutUrl":null,"oauth2Permissions":[{"adminConsentDescription":"Allow the application to access
CloudSense on behalf of the signed-in user.","adminConsentDisplayName":"Access CloudSense","id":"b7b7338e-683a-
4796-b95e-60c10380de1c","isEnabled":true,"type":"User","userConsentDescription":"Allow the application to
access CloudSense on your behalf.","userConsentDisplayName":"Access
CloudSense","value":"user_impersonation"}],"passwordCredentials":
[],"preferredTokenSigningKeyThumbprint":null,"publisherName":"vipswapper"quot;,"replyUrls":
["http://www.vipswapper.com/cloudsense","http://www.vipswapper.com","http://vipswapper.com","http://vipswapper.
azurewebsites.net","http://localhost:62080"],"samlMetadataUrl":null,"servicePrincipalNames":
["http://www.vipswapper.com/cloudsense","a0448380-c346-4f9f-b897-c18733de9394"],"tags":
["WindowsAzureActiveDirectoryIntegratedApp"]}]}
```

Get Azure RBAC role identifier

To assign the appropriate RBAC role to your service principal, you must determine the identifier of the Azure RBAC role.

The right RBAC role for your application:

- If your application only monitors the subscription, without making any changes, it requires only reader permissions on the subscription. Assign the **Reader** role.
- If your application manages Azure the subscription, creating/modifying/deleting entities, it requires one of the contributor permissions.
  - To manage a particular type of resource, assign the resource-specific contributor roles (Virtual Machine Contributor, Virtual Network Contributor, Storage Account Contributor, etc.)
  - To manage any resource type, assign the **Contributor** role.

The role assignment for your application is visible to users, so select the least-required privilege.

Call the Resource Manager role definition API to list all Azure RBAC roles and then iterate over the result to find the role definition by name.

The GetRoleId method of the ASP.net MVC sample app implements this call.

The following request example shows how to get Azure RBAC role identifier. 09cbd307-aa71-4aca-b346-5f253e6e3ebb is the ID of the subscription.

```
GET https://management.azure.com/subscriptions/09cbd307-aa71-4aca-b346-
5f253e6e3ebb/providers/Microsoft.Authorization/roleDefinitions?api-version=2015-07-01 HTTP/1.1

Authorization: Bearer eyJ0eXAiOiJKV*****fY2lGc5
```

The response is in the following format:

```
HTTP/1.1 200 OK

{"value":[{"properties":{"roleName":"API Management Service
Contributor","type":"BuiltInRole","description":"Lets you manage API Management services, but not access to
them.","scope":"/","permissions":[{"actions":
["Microsoft.ApiManagement/Services/*","Microsoft.Authorization/*/read","Microsoft.Resources/subscriptions/resou
rces/read","Microsoft.Resources/subscriptions/resourceGroups/read","Microsoft.Resources/subscriptions/resourceG
roups/resources/read","Microsoft.Resources/subscriptions/resourceGroups/deployments/*","Microsoft.Insights/aler
tRules/*","Microsoft.Support/*"],"notActions":[]}]},"id":"/subscriptions/09cbd307-aa71-4aca-b346-
5f253e6e3ebb/providers/Microsoft.Authorization/roleDefinitions/312a565d-c81f-4fd8-895a-
4e21e48d571c","type":"Microsoft.Authorization/roleDefinitions","name":"312a565d-c81f-4fd8-895a-4e21e48d571c"},
{"properties":{"roleName":"Application Insights Component Contributor","type":"BuiltInRole","description":"Lets
you manage Application Insights components, but not access to them.","scope":"/","permissions":[{"actions":
["Microsoft.Insights/components/*","Microsoft.Insights/webtests/*","Microsoft.Authorization/*/read","Microsoft.
Resources/subscriptions/resources/read","Microsoft.Resources/subscriptions/resourceGroups/read","Microsoft.Reso
urces/subscriptions/resourceGroups/resources/read","Microsoft.Resources/subscriptions/resourceGroups/deployment
s/*","Microsoft.Insights/alertRules/*","Microsoft.Support/*"],"notActions":[]}]},"id":"/subscriptions/09cbd307-
aa71-4aca-b346-5f253e6e3ebb/providers/Microsoft.Authorization/roleDefinitions/ae349356-3a1b-4a5e-921d-
050484c6347e","type":"Microsoft.Authorization/roleDefinitions","name":"ae349356-3a1b-4a5e-921d-050484c6347e"}]}
```

You don't need to call this API on an ongoing basis. Once you've determined the well-known GUID of the role definition, you can construct the role definition ID as:

```
/subscriptions/{subscription_id}/providers/Microsoft.Authorization/roleDefinitions/{well-known-role-guid}
```

Here are the identifiers of commonly used built-in roles:

ROLE	GUID
Reader	acdd72a7-3385-48ef-bd42-f606fba81ae7
Contributor	b24988ac-6180-42a0-ab88-20f7382dd24c
Virtual Machine Contributor	d73bb868-a0df-4d4d-bd69-98a00b01fccb
Virtual Network Contributor	b34d265f-36f7-4a0d-a4d4-e158ca92e90f
Storage Account Contributor	86e8f5dc-a6e9-4c67-9d15-de283e8eac25
Website Contributor	de139f84-1756-47ae-9be6-808fbbe84772
Web Plan Contributor	2cc479cb-7b4d-49a8-b449-8c00fd0f0a4b
SQL Server Contributor	6d8ee4ec-f05a-4a1d-8b00-a9b17e38b437
SQL DB Contributor	9b7fa17d-e63e-47b0-bb0a-15c516ac86ec

Assign RBAC role to application

You have everything you need to assign the appropriate RBAC role to your service principal by using the Resource

Manager create role assignment API.

The GrantRoleToServicePrincipalOnSubscription method of the ASP.net MVC sample app implements this call.

An example request to assign RBAC role to application:

```
PUT https://management.azure.com/subscriptions/09cbd307-aa71-4aca-b346-
5f253e6e3ebb/providers/microsoft.authorization/roleassignments/4f87261d-2816-465d-8311-70a27558df4c?api-
version=2015-07-01 HTTP/1.1

Authorization: Bearer eyJ0eXAiOiJKV1QiL*****FlwO1mM7Cw6JWtfY2lGc5
Content-Type: application/json
Content-Length: 230

{"properties": {"roleDefinitionId":"/subscriptions/09cbd307-aa71-4aca-b346-
5f253e6e3ebb/providers/Microsoft.Authorization/roleDefinitions/b24988ac-6180-42a0-ab88-
20f7382dd24c","principalId":"c3097b31-7309-4c59-b4e3-770f8406bad2"}}
```

In the request, the following values are used:

GUID	DESCRIPTION
09cbd307-aa71-4aca-b346-5f253e6e3ebb	the ID of the subscription
c3097b31-7309-4c59-b4e3-770f8406bad2	the object ID of the service principal of the application
b24988ac-6180-42a0-ab88-20f7382dd24c	the ID of the contributor role
4f87261d-2816-465d-8311-70a27558df4c	a new guid created for the new role assignment

The response is in the following format:

```
HTTP/1.1 201 Created

{"properties":{"roleDefinitionId":"/subscriptions/09cbd307-aa71-4aca-b346-
5f253e6e3ebb/providers/Microsoft.Authorization/roleDefinitions/b24988ac-6180-42a0-ab88-
20f7382dd24c","principalId":"c3097b31-7309-4c59-b4e3-770f8406bad2","scope":"/subscriptions/09cbd307-aa71-4aca-
b346-5f253e6e3ebb"},"id":"/subscriptions/09cbd307-aa71-4aca-b346-
5f253e6e3ebb/providers/Microsoft.Authorization/roleAssignments/4f87261d-2816-465d-8311-
70a27558df4c","type":"Microsoft.Authorization/roleAssignments","name":"4f87261d-2816-465d-8311-70a27558df4c"}
```

Get app-only access token for Azure Resource Manager

To validate that app can access the subscription, do a test task on the subscription using an app-only token.

To get an app-only access token, follow instructions from section Get app-only access token for Azure AD Graph API, with a different value for the resource parameter:

```
https://management.core.windows.net/
```

The ServicePrincipalHasReadAccessToSubscription method of the ASP.NET MVC sample application gets an app-only access token for Azure Resource Manager using the Active Directory Authentication Library for .net.

Get Application's Permissions on Subscription

To check that your application can access an Azure subscription, you may also call the Resource Manager Permissions API. This approach is similar to how you determined whether the user has Access Management rights for the subscription. However, this time, call the permissions API with the app-only access token that you received in the previous step.

The ServicePrincipalHasReadAccessToSubscription method of the ASP.NET MVC sample app implements this call.

Manage connected subscriptions

When the appropriate RBAC role is assigned to your application's service principal on the subscription, your application can keep monitoring/managing it using app-only access tokens for Azure Resource Manager.

If a subscription owner removes your application's role assignment using the portal or command-line tools, your application is no longer able to access that subscription. In that case, you should notify the user that the connection with the subscription was severed from outside the application and give them an option to "repair" the connection. "Repair" would re-create the role assignment that was deleted offline.

Just as you enabled the user to connect subscriptions to your application, you must allow the user to disconnect subscriptions too. From an access management point of view, disconnect means removing the role assignment that the application's service principal has on the subscription. Optionally, any state in the application for the subscription might be removed too. Only users with access management permission on the subscription can disconnect the subscription.

The RevokeRoleFromServicePrincipalOnSubscription method of the ASP.net MVC sample app implements this call.

That's it - users can now easily connect and manage their Azure subscriptions with your application.

Programmatically create Azure Enterprise subscriptions (preview)

7/31/2019 • 7 minutes to read • Edit Online

As an Azure customer on Enterprise Agreement (EA), you can create EA (MS-AZR-0017P) and EA Dev/Test (MS-AZR-0148P) subscriptions programmatically. In this article, you learn how to create subscriptions programmatically using Azure Resource Manager.

When you create an Azure subscription from this API, that subscription is governed by the agreement under which you obtained Microsoft Azure services from Microsoft or an authorized reseller. To learn more, see Microsoft Azure Legal Information.

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

Prerequisites

You must have an Owner role on the Enrollment Account you wish to create subscriptions under. There are two ways to get these roles:

- Your Enrollment Administrator can make you an Account Owner (sign in required) which makes you an Owner of the Enrollment Account. Follow the instructions in the invitation email you receive to manually create an initial subscription. Confirm account ownership and manually create an initial EA subscription before proceeding to the next step. Just adding the account to the enrollment isn't enough.
- An existing Owner of the Enrollment Account can grant you access. Similarly, if you want to use a service principal to create the EA subscription, you must grant that service principal the ability to create subscriptions.

Find accounts you have access to

After you're added to an Azure EA enrollment as an Account Owner, Azure uses the account-to-enrollment relationship to determine where to bill the subscription charges. All subscriptions created under the account are billed towards the EA enrollment that the account is in. To create subscriptions, you must pass in values about the enrollment account and the user principals to own the subscription.

To run the following commands, you must be logged in to the Account Owner's *home directory*, which is the directory that subscriptions are created in by default.

- REST
- PowerShell
- Azure CLI

Request to list all enrollment accounts you have access to:

```
GET https://management.azure.com/providers/Microsoft.Billing/enrollmentAccounts?api-version=2018-03-01-preview
```

Azure responds with a list of all enrollment accounts you have access to:

```
{
  "value": [
    {
      "id": "/providers/Microsoft.Billing/enrollmentAccounts/747ddfe5-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
      "name": "747ddfe5-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
      "type": "Microsoft.Billing/enrollmentAccounts",
      "properties": {
        "principalName": "SignUpEngineering@contoso.com"
      }
    },
    {
      "id": "/providers/Microsoft.Billing/enrollmentAccounts/4cd2fcf6-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
      "name": "4cd2fcf6-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
      "type": "Microsoft.Billing/enrollmentAccounts",
      "properties": {
        "principalName": "BillingPlatformTeam@contoso.com"
      }
    }
  ]
}
```

Use the `principalName` property to identify the account that you want subscriptions to be billed to. Copy the `name` of that account. For example, if you wanted to create subscriptions under the SignUpEngineering@contoso.com enrollment account, you'd copy `747ddfe5-xxxx-xxxx-xxxx-xxxxxxxxxxxx`. This is the object ID of the enrollment account. Paste this value somewhere so that you can use it in the next step as `enrollmentAccountObjectId`.

Create subscriptions under a specific enrollment account

The following example creates a subscription named *Dev Team Subscription* in the enrollment account selected in the previous step. The subscription offer is *MS-AZR-0017P* (regular Microsoft Enterprise Agreement). It also optionally adds two users as RBAC Owners for the subscription.

- REST
- PowerShell
- Azure CLI

Make the following request, replacing `<enrollmentAccountObjectId>` with the `name` copied from the first step (`747ddfe5-xxxx-xxxx-xxxx-xxxxxxxxxxxx`). If you'd like to specify owners, learn how to get user object IDs.

```
POST
https://management.azure.com/providers/Microsoft.Billing/enrollmentAccounts/<enrollmentAccountObjectId>/provid
ers/Microsoft.Subscription/createSubscription?api-version=2018-03-01-preview

{
  "displayName": "Dev Team Subscription",
  "offerType": "MS-AZR-0017P",
  "owners": [
    {
      "objectId": "<userObjectId>"
    },
    {
      "objectId": "<servicePrincipalObjectId>"
    }
  ]
}
```

ELEMENT NAME	REQUIRED	TYPE	DESCRIPTION
`displayName`	No	String	The display name of the subscription. If not specified, it's set to the name of the offer, like "Microsoft Azure Enterprise."
`offerType`	Yes	String	The offer of the subscription. The two options for EA are MS-AZR-0017P (production use) and MS-AZR-0148P (dev/test, needs to be turned on using the EA portal).
`owners`	No	String	The Object ID of any user that you'd like to add as an RBAC Owner on the subscription when it's created.

In the response, you get back a `subscriptionOperation` object for monitoring. When the subscription creation is finished, the `subscriptionOperation` object would return a `subscriptionLink` object, which has the subscription ID.

Limitations of Azure Enterprise subscription creation API

- Only Azure Enterprise subscriptions can be created using this API.
- There's a limit of 200 subscriptions per enrollment account. After that, subscriptions can only be created through the Account Center. If you want to create more subscriptions through the API, create another enrollment account.
- Users who aren't Account Owners, but were added to an enrollment account via RBAC, can't create subscriptions using the Account Center.
- You can't select the tenant for the subscription to be created in. The subscription is always created in the home tenant of the Account Owner. To move the subscription to a different tenant, see change subscription tenant.

Next steps

- For an example on creating subscriptions using .NET, see sample code on GitHub.
- Now that you've created a subscription, you can grant that ability to other users and service principals. For

more information, see Grant access to create Azure Enterprise subscriptions (preview).

- To learn more about managing large numbers of subscriptions using management groups, see Organize your resources with Azure management groups

Grant access to create Azure Enterprise subscriptions (preview)

6/18/2019 • 5 minutes to read • Edit Online

As an Azure customer on Enterprise Agreement (EA), you can give another user or service principal permission to create subscriptions billed to your account. In this article, you learn how to use Role-Based Access Control (RBAC) to share the ability to create subscriptions, and how to audit subscription creations. You must have the Owner role on the account you wish to share.

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

Grant access

To create subscriptions under an enrollment account, users must have the RBAC Owner role on that account. You can grant a user or a group of users the RBAC Owner role on an enrollment account by following these steps:

1. Get the object ID of the enrollment account you want to grant access to

   To grant others the RBAC Owner role on an enrollment account, you must either be the Account Owner or an RBAC Owner of the account.

   - REST
   - PowerShell
   - Azure CLI

   Request to list all enrollment accounts you have access to:

   ```
   GET https://management.azure.com/providers/Microsoft.Billing/enrollmentAccounts?api-version=2018-03-01-
   preview
   ```

   Azure responds with a list of all enrollment accounts you have access to:

```json
{
  "value": [
    {
      "id": "/providers/Microsoft.Billing/enrollmentAccounts/747ddfe5-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
      "name": "747ddfe5-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
      "type": "Microsoft.Billing/enrollmentAccounts",
      "properties": {
        "principalName": "SignUpEngineering@contoso.com"
      }
    },
    {
      "id": "/providers/Microsoft.Billing/enrollmentAccounts/4cd2fcf6-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
      "name": "4cd2fcf6-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
      "type": "Microsoft.Billing/enrollmentAccounts",
      "properties": {
        "principalName": "BillingPlatformTeam@contoso.com"
      }
    }
  ]
}
```

Use the `principalName` property to identify the account that you want to grant RBAC Owner access to. Copy the `name` of that account. For example, if you wanted to grant RBAC Owner access to the SignUpEngineering@contoso.com enrollment account, you'd copy `747ddfe5-xxxx-xxxx-xxxx-xxxxxxxxxxxx`. This is the object ID of the enrollment account. Paste this value somewhere so that you can use it in the next step as `enrollmentAccountObjectId`.

Use the `principalName` property to identify the account that you want to grant RBAC Owner access to. Copy the `name` of that account. For example, if you wanted to grant RBAC Owner access to the SignUpEngineering@contoso.com enrollment account, you'd copy `747ddfe5-xxxx-xxxx-xxxx-xxxxxxxxxxxx`. This is the object ID of the enrollment account. Paste this value somewhere so that you can use it in the next step as `enrollmentAccountObjectId`.

2. Get object ID of the user or group you want to give the RBAC Owner role to

   a. In the Azure portal, search on **Azure Active Directory**.
   b. If you want to grant a user access, click on **Users** in the menu on the left. If you want to grant access to a group, click **Groups**.
   c. Select the User or Group you want to give the RBAC Owner role to.
   d. If you selected a User, you'll find the object ID in the Profile page. If you selected a Group, the object ID will be in the Overview page. Copy the **ObjectID** by clicking the icon to the right of the text box. Paste this somewhere so that you can use it in the next step as `userObjectId`.

3. Grant the user or group the RBAC Owner role on the enrollment account

   Using the values you collected in the first two steps, grant the user or group the RBAC Owner role on the enrollment account.

   - REST
   - PowerShell
   - Azure CLI

   Run the following command, replacing `<enrollmentAccountObjectId>` with the `name` you copied in the first step (`747ddfe5-xxxx-xxxx-xxxx-xxxxxxxxxxxx`). Replace `<userObjectId>` with the object ID you copied from the second step.

```
PUT
https://management.azure.com/providers/Microsoft.Billing/enrollmentAccounts/<enrollmentAccountObjectId>
/providers/Microsoft.Authorization/roleAssignments/<roleAssignmentGuid>?api-version=2015-07-01

{
  "properties": {
    "roleDefinitionId":
"/providers/Microsoft.Billing/enrollmentAccounts/providers/Microsoft.Authorization/roleDefinitions/<own
erRoleDefinitionId>",
    "principalId": "<userObjectId>"
  }
}
```

When the Owner role is successfully assigned at the enrollment account scope, Azure responds with information of the role assignment:

```
{
  "properties": {
    "roleDefinitionId":
"/providers/Microsoft.Billing/enrollmentAccounts/providers/Microsoft.Authorization/roleDefinitions/<own
erRoleDefinitionId>",
    "principalId": "<userObjectId>",
    "scope": "/providers/Microsoft.Billing/enrollmentAccounts/747ddfe5-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
    "createdOn": "2018-03-05T08:36:26.4014813Z",
    "updatedOn": "2018-03-05T08:36:26.4014813Z",
    "createdBy": "<assignerObjectId>",
    "updatedBy": "<assignerObjectId>"
  },
  "id":
"/providers/Microsoft.Billing/enrollmentAccounts/providers/Microsoft.Authorization/roleDefinitions/<own
erRoleDefinitionId>",
  "type": "Microsoft.Authorization/roleAssignments",
  "name": "<roleAssignmentGuid>"
}
```

Audit who created subscriptions using activity logs

To track the subscriptions created via this API, use the Tenant Activity Log API. It's currently not possible to use PowerShell, CLI, or Azure portal to track subscription creation.

1. As a tenant admin of the Azure AD tenant, elevate access then assign a Reader role to the auditing user over the scope `/providers/microsoft.insights/eventtypes/management`.

2. As the auditing user, call the Tenant Activity Log API to see subscription creation activities. Example:

```
GET "/providers/Microsoft.Insights/eventtypes/management/values?api-version=2015-04-
01&$filter=eventTimestamp ge '{greaterThanTimeStamp}' and eventTimestamp le '{lessThanTimestamp}' and
eventChannels eq 'Operation' and resourceProvider eq 'Microsoft.Subscription'"
```

To conveniently call this API from the command line, try ARMClient.

Next steps

- Now that the user or service principal has permission to create a subscription, you can use that identity to programmatically create Azure Enterprise subscriptions.
- For an example on creating subscriptions using .NET, see sample code on GitHub.
- To learn more about Azure Resource Manager and its APIs, see Azure Resource Manager overview.
- To learn more about managing large numbers of subscriptions using management groups, see Organize your

resources with Azure management groups

- To see a comprehensive best practice guidance for large organizations on subscription governance, see Azure enterprise scaffold - prescriptive subscription governance

Authenticate requests across tenants

6/18/2019 • 2 minutes to read • Edit Online

When creating a multi-tenant application, you may need to handle authentication requests for resources that are in different tenants. A common scenario is when a virtual machine in one tenant must join a virtual network in another tenant. Azure Resource Manager provides a header value for storing auxiliary tokens to authenticate the requests to different tenants.

Header values for authentication

The request has the following authentication header values:

HEADER NAME	DESCRIPTION	EXAMPLE VALUE
Authorization	Primary token	Bearer <primary-token>
x-ms-authorization-auxiliary	Auxiliary tokens	Bearer <auxiliary-token1>; EncryptedBearer <auxiliary-token2>; Bearer <auxiliary-token3>

The auxiliary header can hold up to three auxiliary tokens.

In the code of your multi-tenant app, get the authentication token for other tenants and store them in the auxiliary headers. All the tokens must be from the same user or application. The user or application must have been invited as a guest to the other tenants.

Processing the request

When your app sends a request to Resource Manager, the request is run under the identity from the primary token. The primary token must be valid and unexpired. This token must be from a tenant that can manage the subscription.

When the request references a resource from different tenant, Resource Manager checks the auxiliary tokens to determine if the request can be processed. All auxiliary tokens in the header must be valid and unexpired. If any token is expired, Resource Manager returns a 401 response code. The response includes the client ID and tenant ID from the token that isn't valid. If the auxiliary header contains a valid token for the tenant, the cross tenant request is processed.

Next steps

- To learn about sending authentication requests with the Azure Resource Manager APIs, see Use Resource Manager authentication API to access subscriptions.
- For more information about tokens, see Azure Active Directory access tokens.

View activity logs to monitor actions on resources

5/14/2019 • 5 minutes to read • Edit Online

Through activity logs, you can determine:

- what operations were taken on the resources in your subscription
- who started the operation
- when the operation occurred
- the status of the operation
- the values of other properties that might help you research the operation

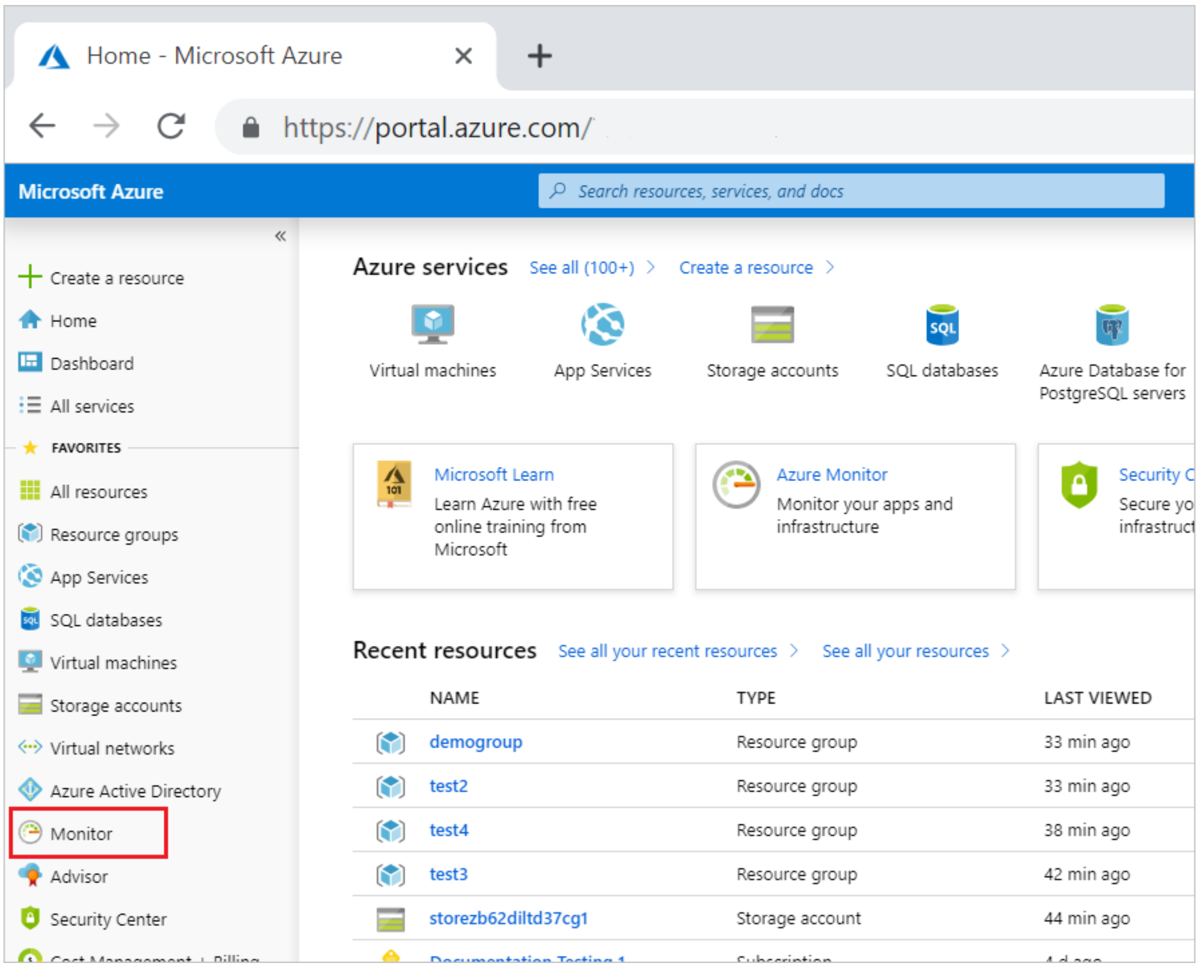
The activity log contains all write operations (PUT, POST, DELETE) for your resources. It doesn't include read operations (GET). For a list of resource actions, see Azure Resource Manager Resource Provider operations. You can use the activity logs to find an error when troubleshooting or to monitor how a user in your organization modified a resource.

Activity logs are kept for 90 days. You can query for any range of dates, as long as the starting date isn't more than 90 days in the past.

You can retrieve information from the activity logs through the portal, PowerShell, Azure CLI, Insights REST API, or Insights .NET Library.

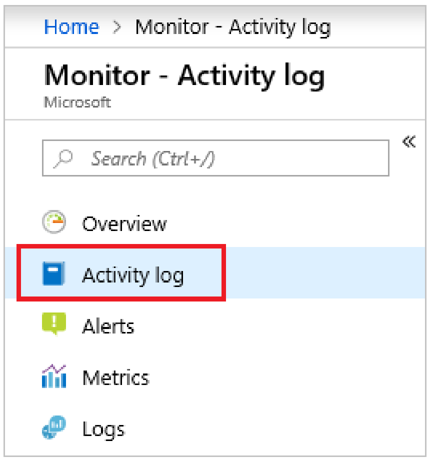
Azure portal

1. To view the activity logs through the portal, select **Monitor**.

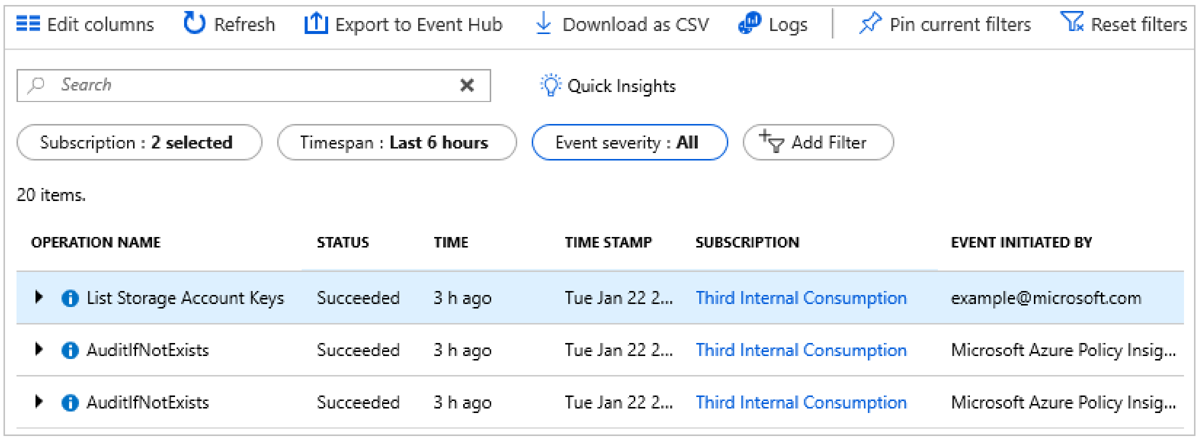


NAME	TYPE	LAST VIEWED
demogroup	Resource group	33 min ago
test2	Resource group	33 min ago
test4	Resource group	38 min ago
test3	Resource group	42 min ago
storezb62diltd37cg1	Storage account	44 min ago

2. Select **Activity Log**.



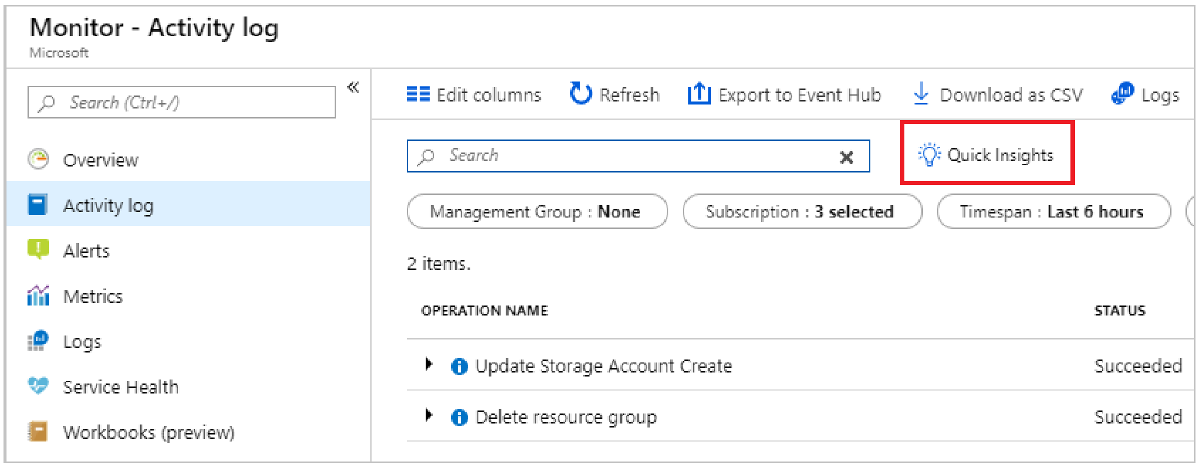
3. You see a summary of recent operations. A default set of filters is applied to the operations. Notice the information on the summary includes who started the action and when it happened.



20 items.

OPERATION NAME	STATUS	TIME	TIME STAMP	SUBSCRIPTION	EVENT INITIATED BY
List Storage Account Keys	Succeeded	3 h ago	Tue Jan 22 2...	Third Internal Consumption	example@microsoft.com
AuditIfNotExists	Succeeded	3 h ago	Tue Jan 22 2...	Third Internal Consumption	Microsoft Azure Policy Insig...
AuditIfNotExists	Succeeded	3 h ago	Tue Jan 22 2...	Third Internal Consumption	Microsoft Azure Policy Insig...

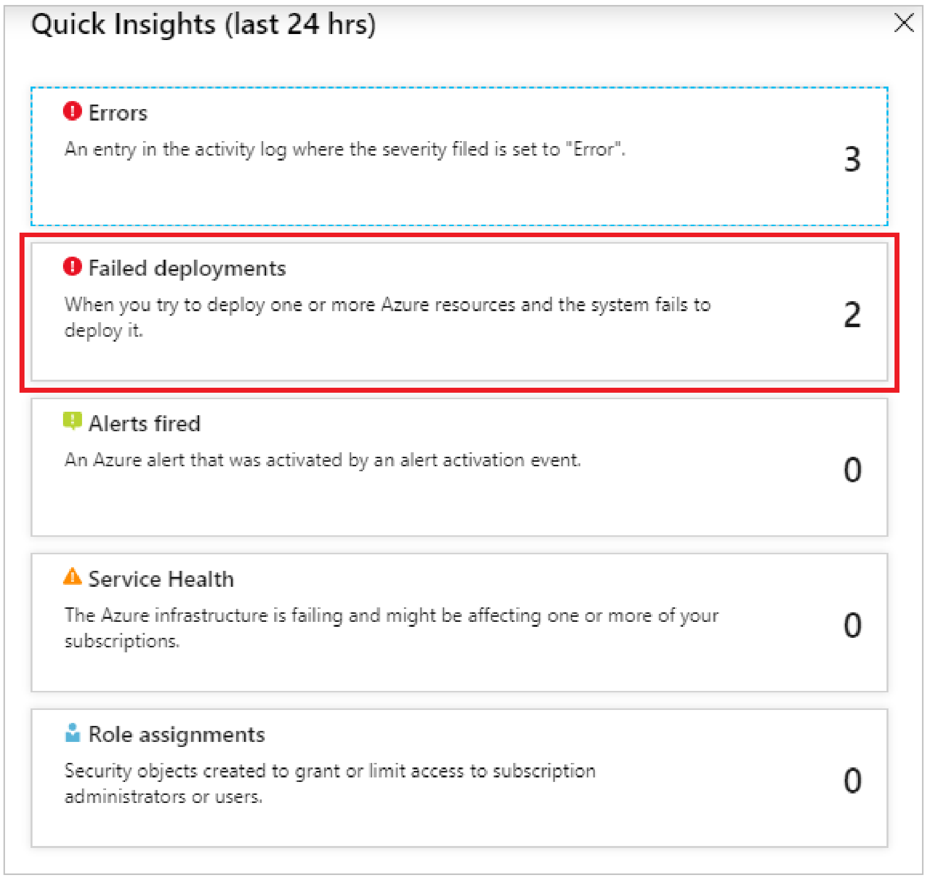
4. To quickly run a pre-defined set of filters, select **Quick Insights**.



2 items.

OPERATION NAME	STATUS
Update Storage Account Create	Succeeded
Delete resource group	Succeeded

5. Select one of the options. For example, select **Failed deployments** to see errors from deployments.


Quick Insights (last 24 hrs)

Errors
An entry in the activity log where the severity filed is set to "Error". 3

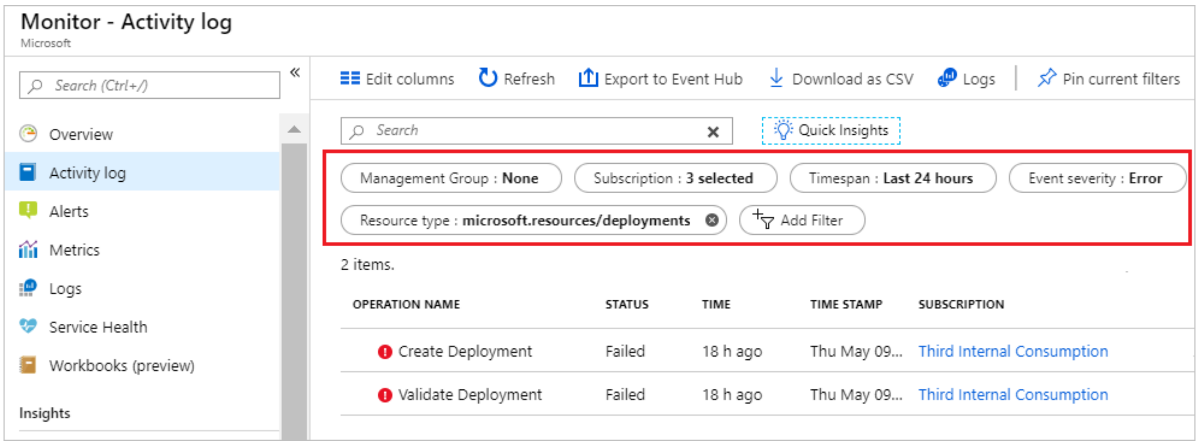
Failed deployments
When you try to deploy one or more Azure resources and the system fails to deploy it. 2

Alerts fired
An Azure alert that was activated by an alert activation event. 0

Service Health
The Azure infrastructure is failing and might be affecting one or more of your subscriptions. 0

Role assignments
Security objects created to grant or limit access to subscription administrators or users. 0

6. Notice the filters have been changed to focus on deployment errors in the last 24 hours. Only operations that match the filters are displayed.

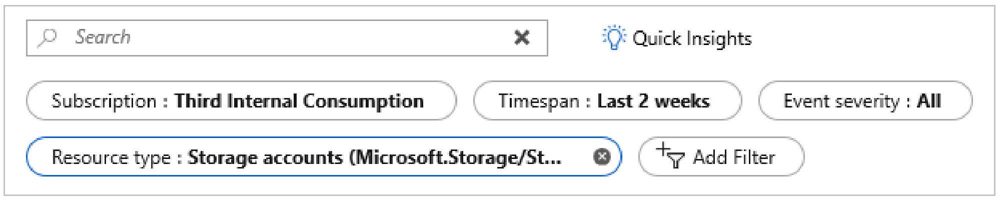

Monitor - Activity log
Microsoft

Management Group : None | Subscription : 3 selected | Timespan : Last 24 hours | Event severity : Error
Resource type : microsoft.resources/deployments

2 items.

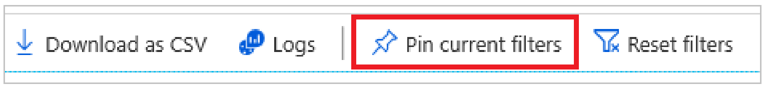
OPERATION NAME	STATUS	TIME	TIME STAMP	SUBSCRIPTION
Create Deployment	Failed	18 h ago	Thu May 09...	Third Internal Consumption
Validate Deployment	Failed	18 h ago	Thu May 09...	Third Internal Consumption

7. To focus on specific operations, change the filters or apply new ones. For example, the following image shows a new value for the **Timespan** and **Resource type** is set to storage accounts.

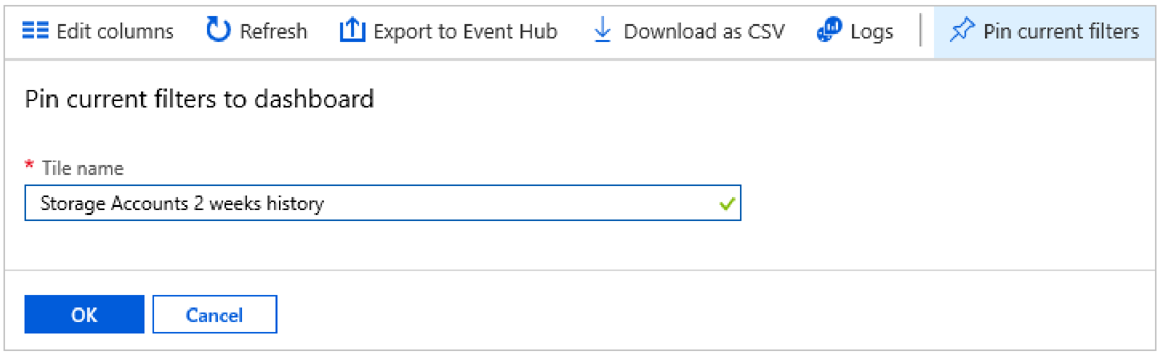


Subscription : Third Internal Consumption | Timespan : Last 2 weeks | Event severity : All
Resource type : Storage accounts (Microsoft.Storage/St...

8. If you need to run the query again later, select **Pin current filters**.



9. Give the filter a name.


Edit columns | Refresh | Export to Event Hub | Download as CSV | Logs | Pin current filters

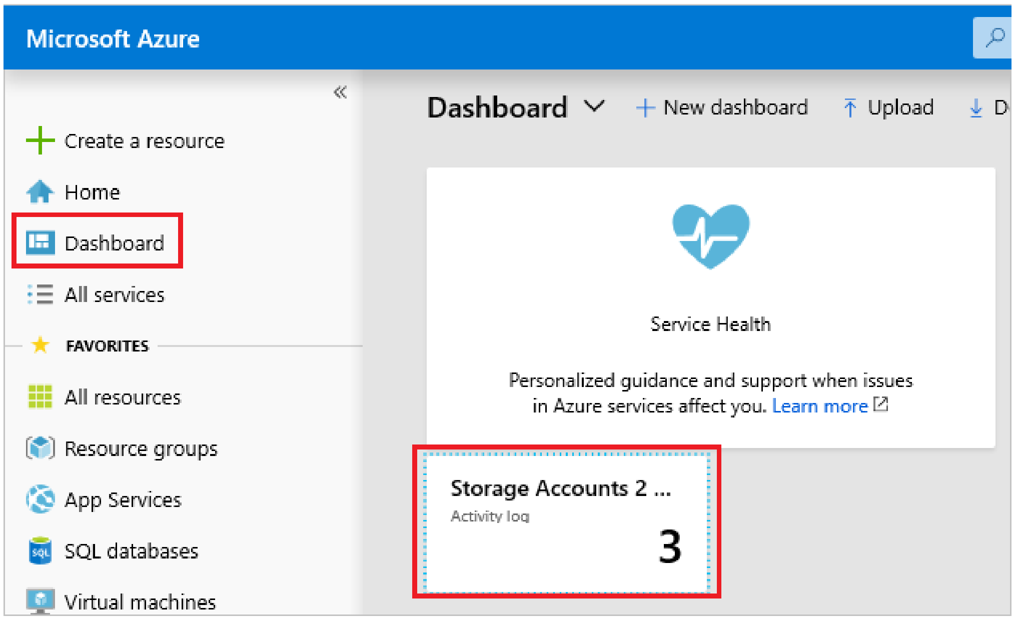
Pin current filters to dashboard

* Tile name

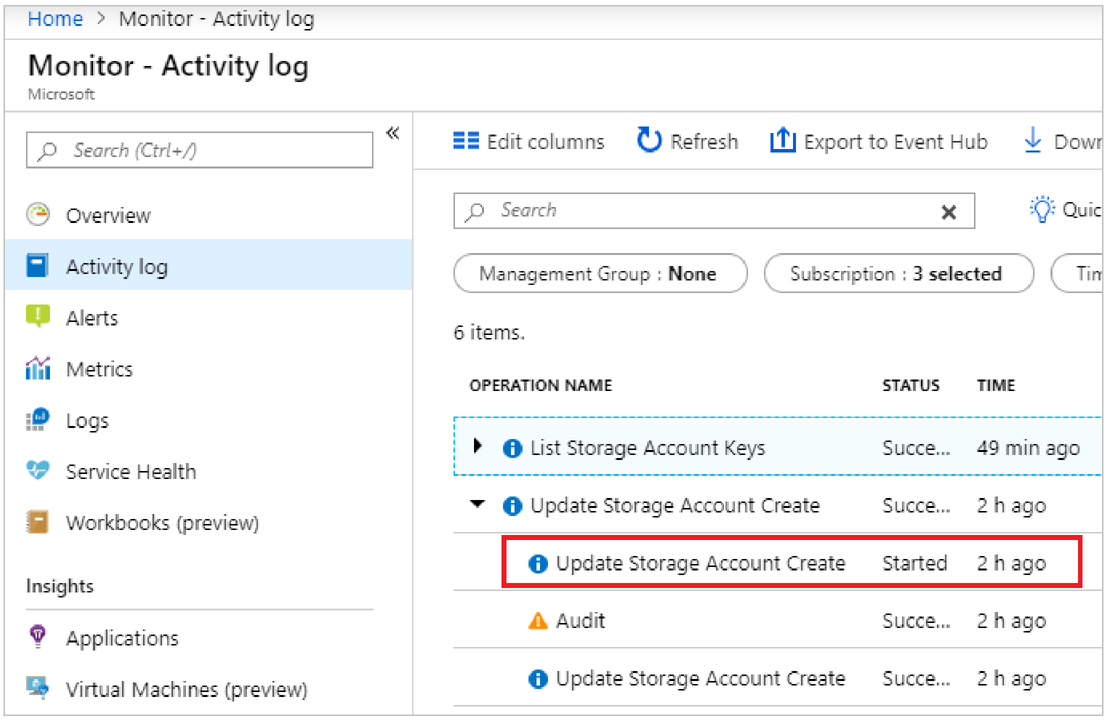
Storage Accounts 2 weeks history

OK | Cancel

10. The filter is available in the dashboard.



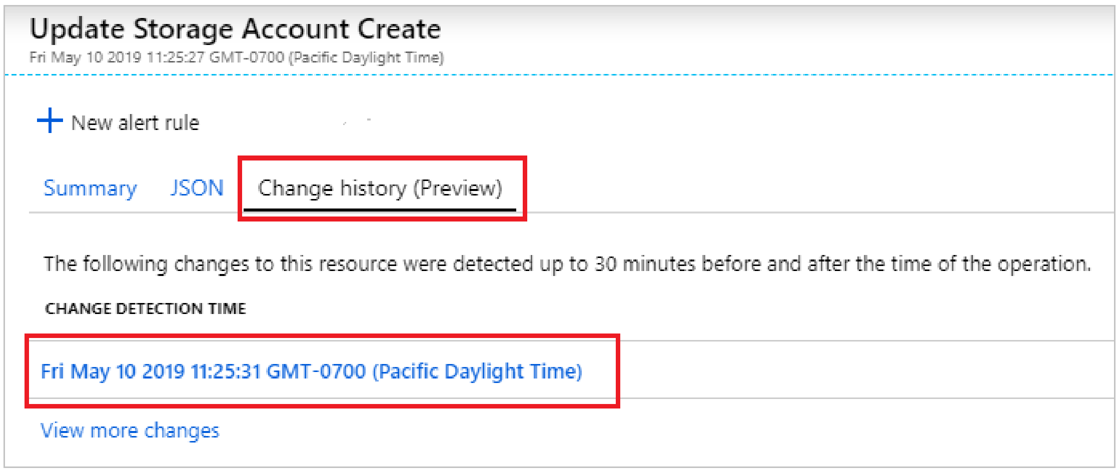
11. From the portal, you can view changes to a resource. Go back to the default view in Monitor, and select an operation that involved changing a resource.



6 items.

OPERATION NAME	STATUS	TIME
List Storage Account Keys	Succe...	49 min ago
Update Storage Account Create	Succe...	2 h ago
Update Storage Account Create	Started	2 h ago
Audit	Succe...	2 h ago
Update Storage Account Create	Succe...	2 h ago

12. Select **Change history (Preview)** and pick one of the available operations.



Update Storage Account Create

Fri May 10 2019 11:25:27 GMT-0700 (Pacific Daylight Time)

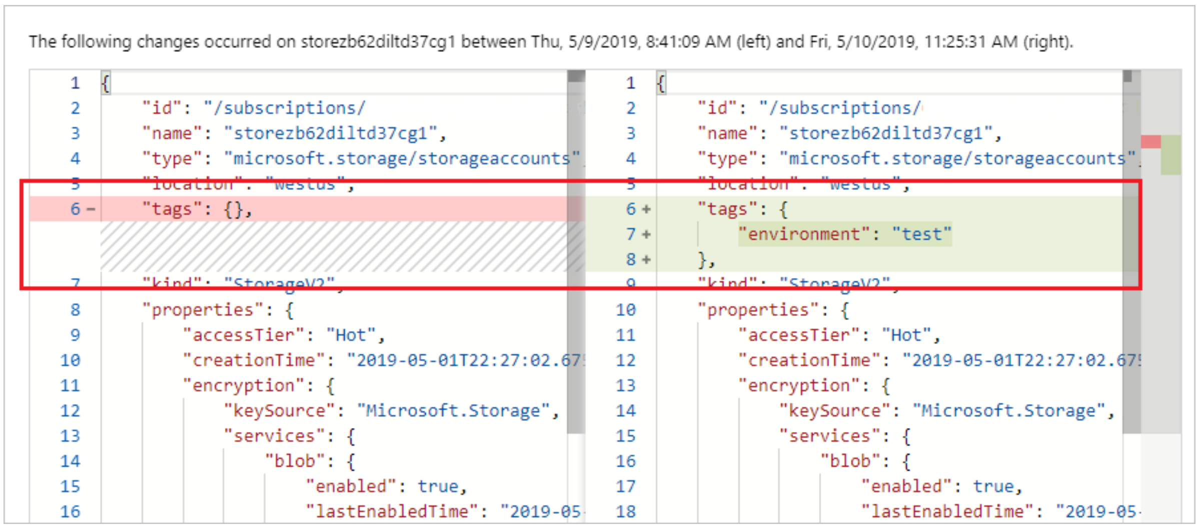
The following changes to this resource were detected up to 30 minutes before and after the time of the operation.

CHANGE DETECTION TIME

Fri May 10 2019 11:25:31 GMT-0700 (Pacific Daylight Time)

View more changes

13. The changes in the resource are displayed.



The following changes occurred on storezb62diltd37cg1 between Thu, 5/9/2019, 8:41:09 AM (left) and Fri, 5/10/2019, 11:25:31 AM (right).

```
{
    "id": "/subscriptions/
    "name": "storezb62diltd37cg1",
    "type": "microsoft.storage/storageaccounts"
    "location": "westus",
    "tags": {},
    "kind": "StorageV2",
    "properties": {
        "accessTier": "Hot",
        "creationTime": "2019-05-01T22:27:02.67
        "encryption": {
            "keySource": "Microsoft.Storage",
            "services": {
                "blob": {
                    "enabled": true,
                    "lastEnabledTime": "2019-05
```

```
{
    "id": "/subscriptions/
    "name": "storezb62diltd37cg1",
    "type": "microsoft.storage/storageaccounts"
    "location": "westus",
    "tags": {
        "environment": "test"
    },
    "kind": "StorageV2",
    "properties": {
        "accessTier": "Hot",
        "creationTime": "2019-05-01T22:27:02.67
        "encryption": {
            "keySource": "Microsoft.Storage",
            "services": {
                "blob": {
                    "enabled": true,
                    "lastEnabledTime": "2019-05
```

To learn more about change history, see Get resource changes.

PowerShell

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

To retrieve log entries, run the **Get-AzLog** command. You provide additional parameters to filter the list of entries. If you don't specify a start and end time, entries for the last seven days are returned.

```
Get-AzLog -ResourceGroup ExampleGroup
```

The following example shows how to use the activity log to research operations taken during a specified time. The start and end dates are specified in a date format.

```
Get-AzLog -ResourceGroup ExampleGroup -StartTime 2019-05-05T06:00 -EndTime 2019-05-09T06:00
```

Or, you can use date functions to specify the date range, such as the last 14 days.

```
Get-AzLog -ResourceGroup ExampleGroup -StartTime (Get-Date).AddDays(-14)
```

You can look up the actions taken by a particular user.

```
Get-AzLog -ResourceGroup ExampleGroup -StartTime (Get-Date).AddDays(-14) -Caller someone@contoso.com
```

You can filter for failed operations.

```
Get-AzLog -ResourceGroup ExampleGroup -Status Failed
```

You can focus on one error by looking at the status message for that entry.

```
(Get-AzLog -ResourceGroup ExampleGroup -Status Failed).Properties.Content.statusMessage | ConvertFrom-Json
```

You can select specific values to limit the data that is returned.

```
Get-AzLog -ResourceGroupName ExampleGroup | Format-table EventTimeStamp, Caller, @{n='Operation'; e=
{$_.OperationName.value}}, @{n='Status'; e={$_.Status.value}}, @{n='SubStatus'; e=
{$_.SubStatus.LocalizedValue}}
```

Depending on the start time you specify, the previous commands can return a long list of operations for the resource group. You can filter the results for what you are looking for by providing search criteria. For example, you can filter by the type of operation.

```
Get-AzLog -ResourceGroup ExampleGroup | Where-Object {$_.OperationName.value -eq
"Microsoft.Resources/deployments/write"}
```

You can use Resource Graph to see the change history for a resource. For more information, see Get resource changes.

Azure CLI

To retrieve log entries, run the az monitor activity-log list command with an offset to indicate the time span.

```
az monitor activity-log list --resource-group ExampleGroup --offset 7d
```

The following example shows how to use the activity log to research operations taken during a specified time. The start and end dates are specified in a date format.

```
az monitor activity-log list -g ExampleGroup --start-time 2019-05-01 --end-time 2019-05-15
```

You can look up the actions taken by a particular user, even for a resource group that no longer exists.

```
az monitor activity-log list -g ExampleGroup --caller someone@contoso.com --offset 5d
```

You can filter for failed operations.

```
az monitor activity-log list -g ExampleGroup --status Failed --offset 1d
```

You can focus on one error by looking at the status message for that entry.

```
az monitor activity-log list -g ExampleGroup --status Failed --offset 1d --query [].properties.statusMessage
```

You can select specific values to limit the data that is returned.

```
az monitor activity-log list -g ExampleGroup --offset 1d --query '[].{Operation: operationName.value, Status:
status.value, SubStatus: subStatus.localizedValue}'
```

Depending on the start time you specify, the previous commands can return a long list of operations for the resource group. You can filter the results for what you are looking for by providing search criteria. For example, you can filter by the type of operation.

```
az monitor activity-log list -g ExampleGroup --offset 1d --query "[?
operationName.value=='Microsoft.Storage/storageAccounts/write']"
```

You can use Resource Graph to see the change history for a resource. For more information, see Get resource changes.

REST API

The REST operations for working with the activity log are part of the Insights REST API. To retrieve activity log events, see List the management events in a subscription.

Next steps

- Azure Activity logs can be used with Power BI to gain greater insights about the actions in your subscription. See View and analyze Azure Activity Logs in Power BI and more.
- To learn about setting security policies, see Azure Role-based Access Control.
- To learn about the commands for viewing deployment operations, see View deployment operations.
- To learn how to prevent deletions on a resource for all users, see Lock resources with Azure Resource Manager.
- To see the list of operations available for each Microsoft Azure Resource Manager provider, see Azure Resource Manager Resource Provider operations

View deployment history with Azure Resource Manager

5/14/2019 • 4 minutes to read • Edit Online

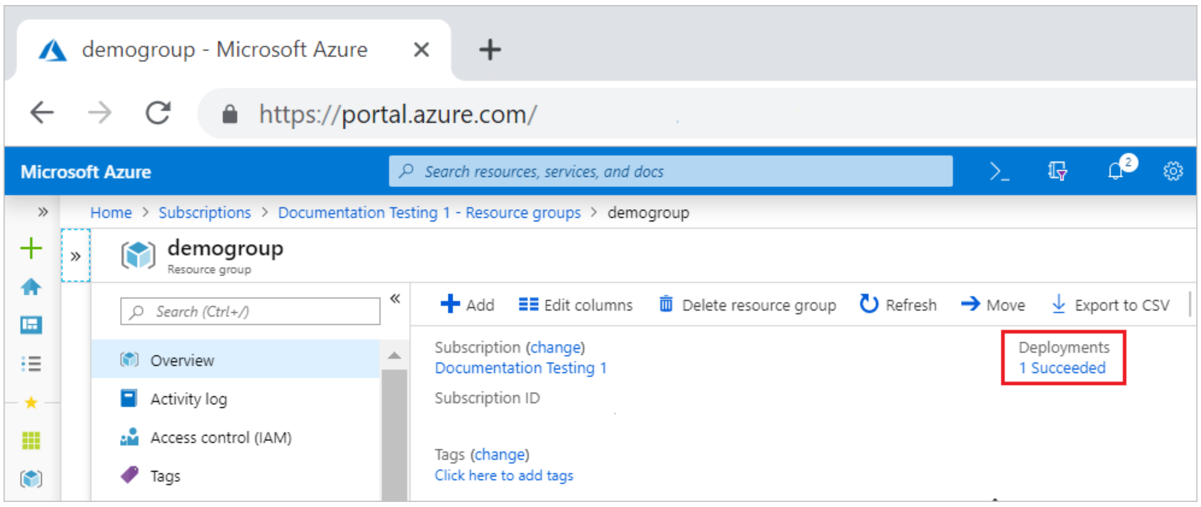
Azure Resource Manager enables you to view your deployment history and examine specific operations in past deployments. You can see the resources that were deployed, and get information about any errors.

For help with resolving particular deployment errors, see Resolve common errors when deploying resources to Azure with Azure Resource Manager.

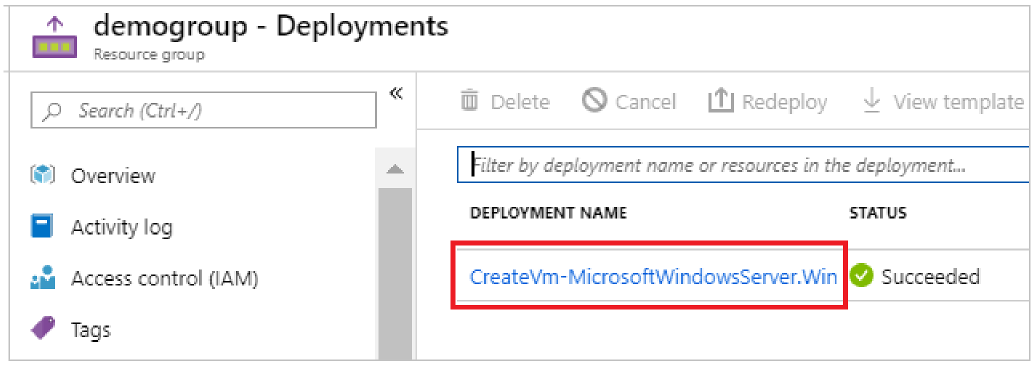
Portal

To get details about a deployment from the deployment history.

1. Select the resource group you want to examine.
2. Select the link under **Deployments**.

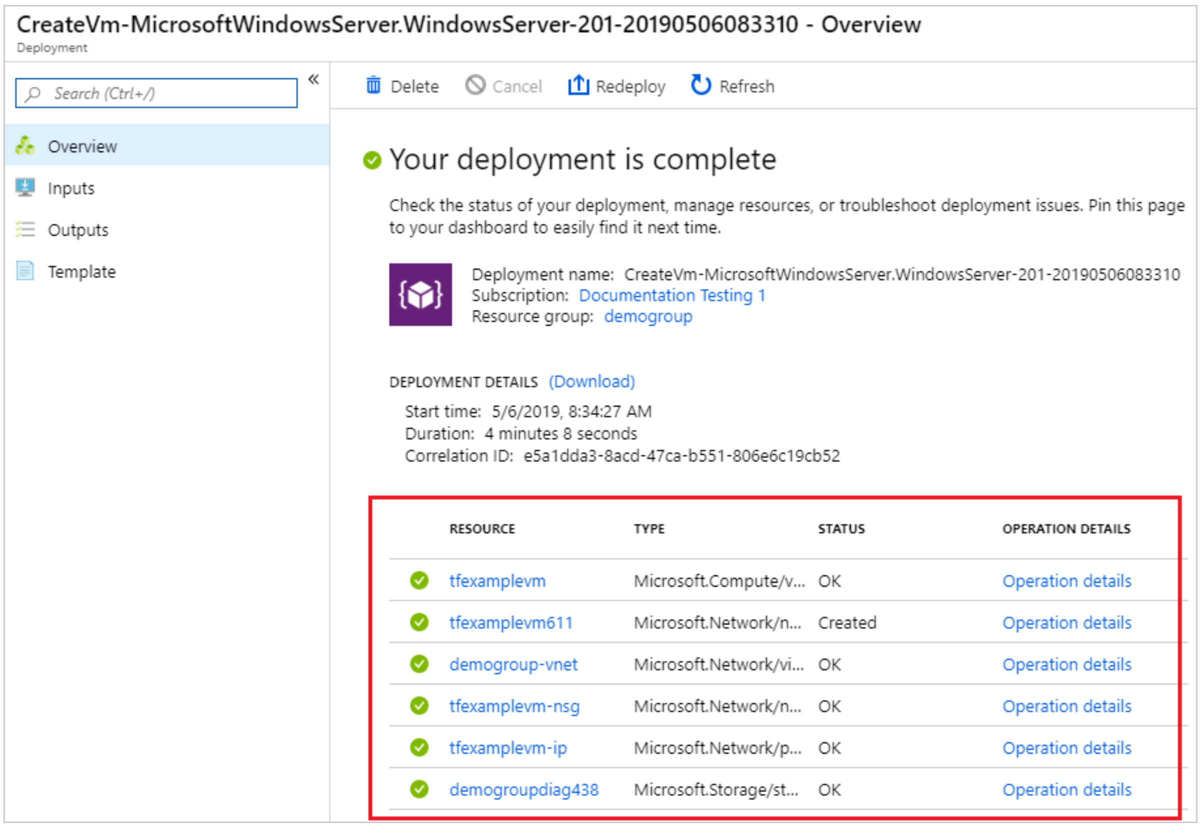


3. Select one of the deployments from the deployment history.



DEPLOYMENT NAME	STATUS
CreateVm-MicrosoftWindowsServer.Win	Succeeded

4. A summary of the deployment is displayed, including a list of the resources that were deployed.


CreateVm-MicrosoftWindowsServer.WindowsServer-201-20190506083310 - Overview
Deployment

Your deployment is complete

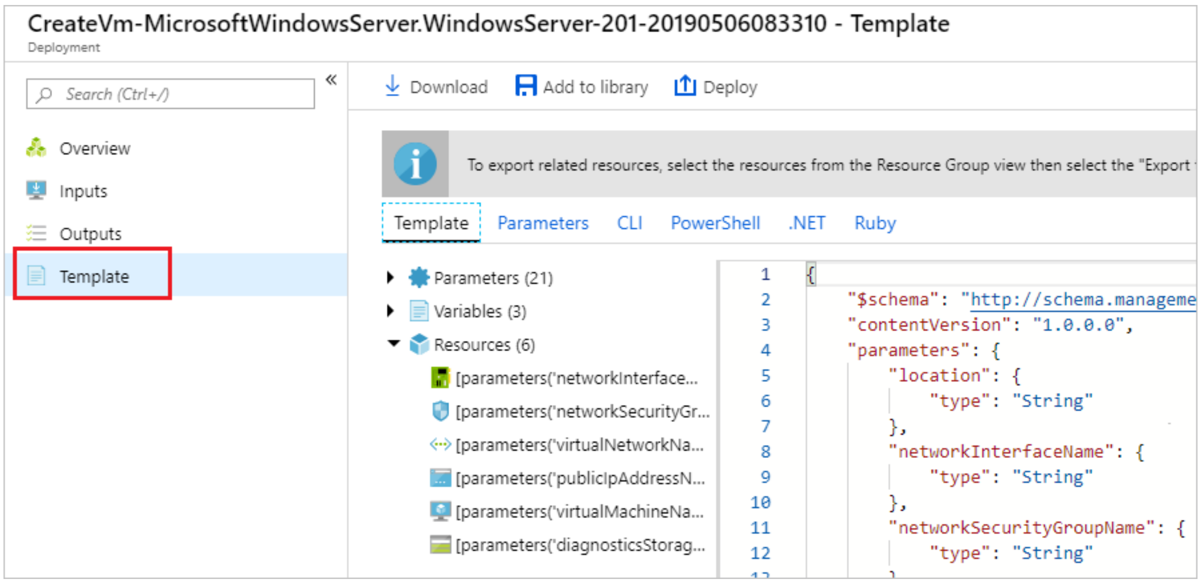
Check the status of your deployment, manage resources, or troubleshoot deployment issues. Pin this page to your dashboard to easily find it next time.

Deployment name: CreateVm-MicrosoftWindowsServer.WindowsServer-201-20190506083310
Subscription: Documentation Testing 1
Resource group: demogroup

DEPLOYMENT DETAILS (Download)
Start time: 5/6/2019, 8:34:27 AM
Duration: 4 minutes 8 seconds
Correlation ID: e5a1dda3-8acd-47ca-b551-806e6c19cb52

RESOURCE	TYPE	STATUS	OPERATION DETAILS
tfexamplevm	Microsoft.Compute/v...	OK	Operation details
tfexamplevm611	Microsoft.Network/n...	Created	Operation details
demogroup-vnet	Microsoft.Network/vi...	OK	Operation details
tfexamplevm-nsg	Microsoft.Network/n...	OK	Operation details
tfexamplevm-ip	Microsoft.Network/p...	OK	Operation details
demogroupdiag438	Microsoft.Storage/st...	OK	Operation details

5. To view the template used for the deployment, select **Template**. You can download the template to reuse it.

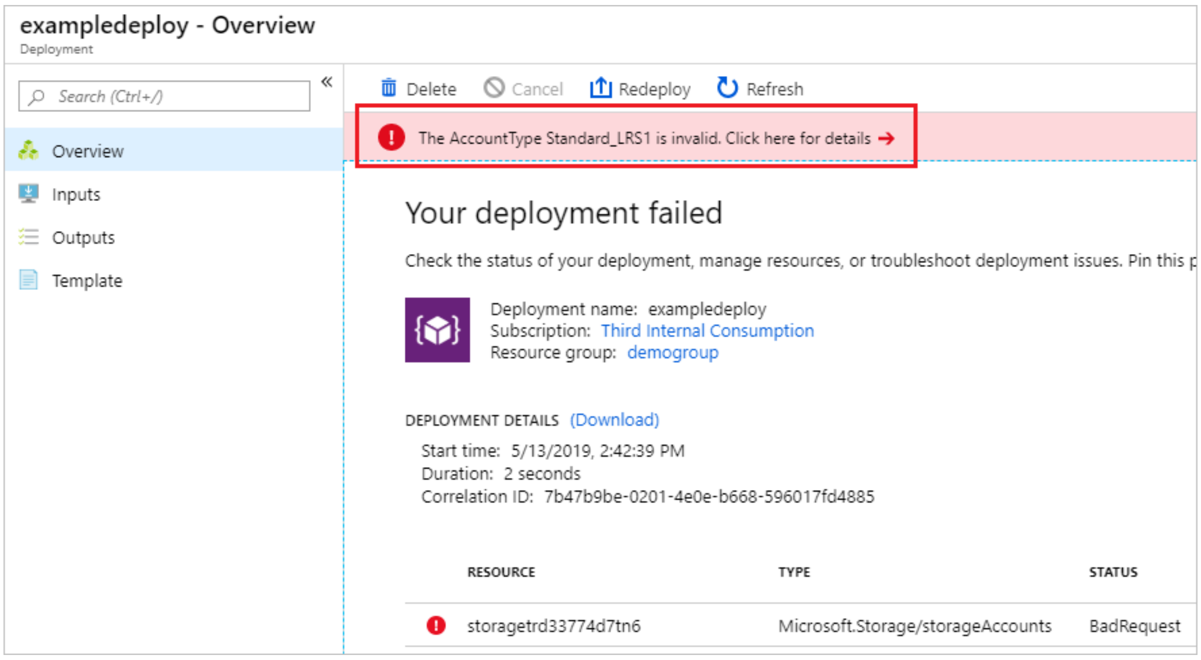

CreateVm-MicrosoftWindowsServer.WindowsServer-201-20190506083310 - Template
Deployment

To export related resources, select the resources from the Resource Group view then select the "Export

Parameters (21)
Variables (3)
Resources (6)
[parameters('networkInterface...
[parameters('networkSecurityGr...
[parameters('virtualNetworkNa...
[parameters('publicIpAddressN...
[parameters('virtualMachineNa...
[parameters('diagnosticsStorag...

```
{
    "$schema": "http://schema.manageme
    "contentVersion": "1.0.0.0",
    "parameters": {
        "location": {
            "type": "String"
        },
        "networkInterfaceName": {
            "type": "String"
        },
        "networkSecurityGroupName": {
            "type": "String"
```

6. If your deployment failed, you see an error message. Select the error message for more details.


exampledeploy - Overview
Deployment

The AccountType Standard_LRS1 is invalid. Click here for details →

Your deployment failed

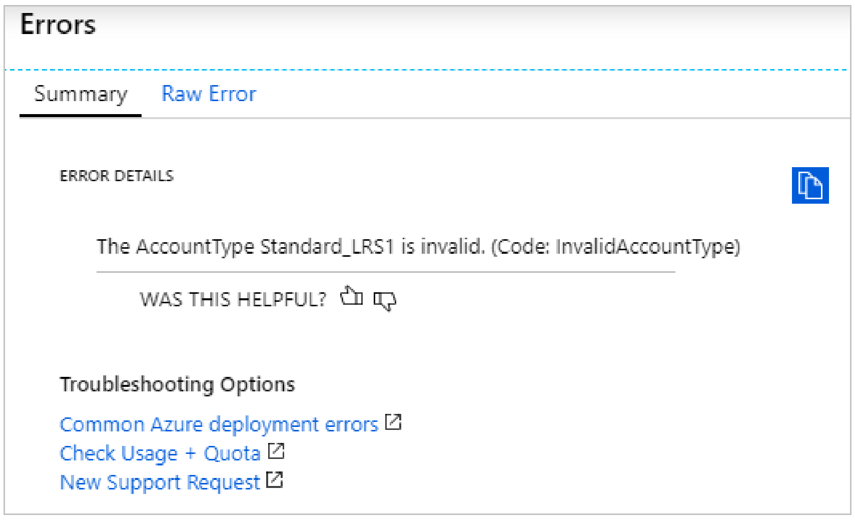
Check the status of your deployment, manage resources, or troubleshoot deployment issues. Pin this p

Deployment name: exampledeploy
Subscription: Third Internal Consumption
Resource group: demogroup

DEPLOYMENT DETAILS (Download)
Start time: 5/13/2019, 2:42:39 PM
Duration: 2 seconds
Correlation ID: 7b47b9be-0201-4e0e-b668-596017fd4885

RESOURCE	TYPE	STATUS
storagetrd33774d7tn6	Microsoft.Storage/storageAccounts	BadRequest

7. The detailed error message is displayed.


Errors

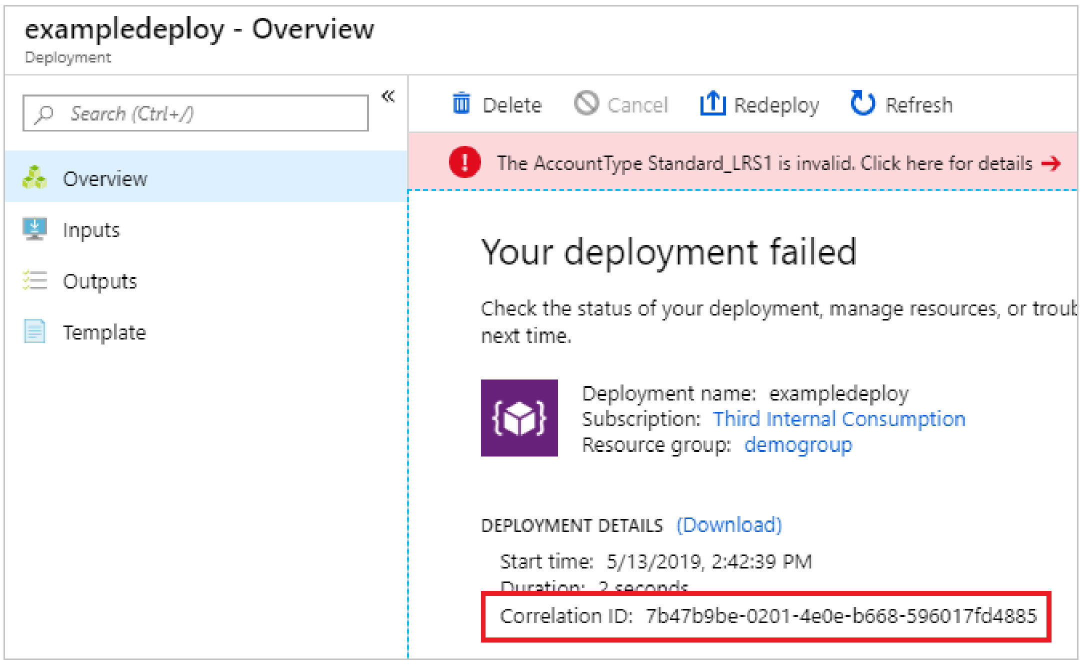
ERROR DETAILS

The AccountType Standard_LRS1 is invalid. (Code: InvalidAccountType)

WAS THIS HELPFUL?

Troubleshooting Options
Common Azure deployment errors
Check Usage + Quota
New Support Request

8. The correlation ID is used to track related events, and can be helpful when working with technical support to troubleshoot a deployment.


exampledeploy - Overview
Deployment

The AccountType Standard_LRS1 is invalid. Click here for details →

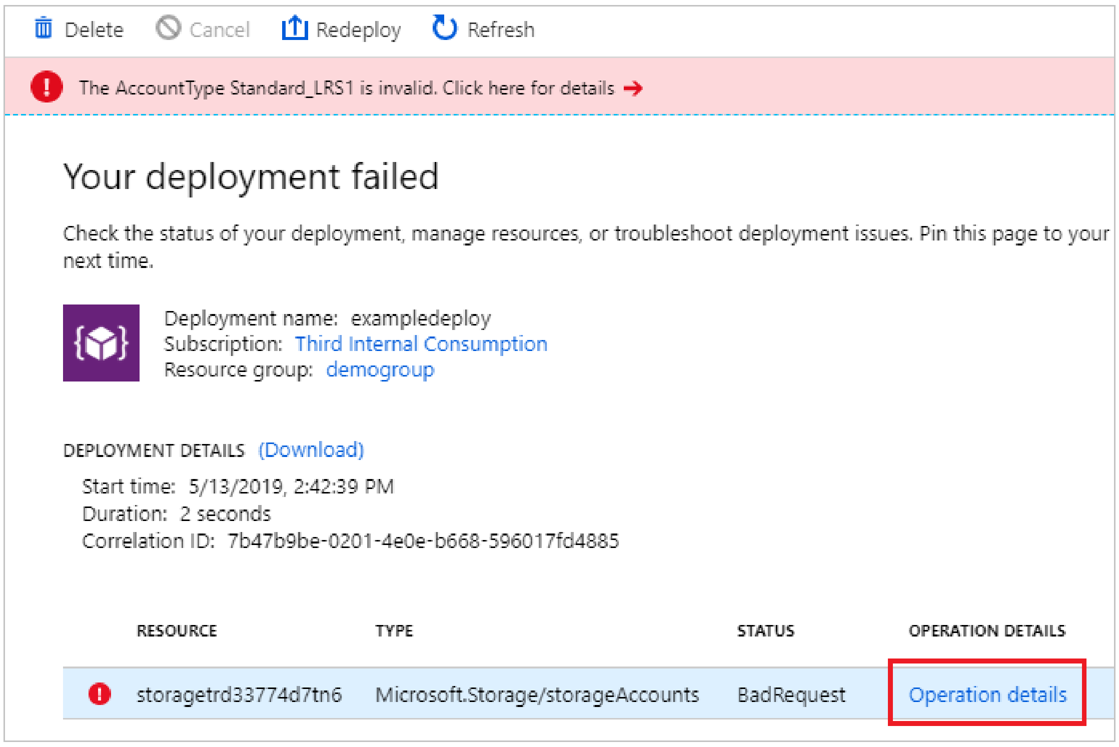
Your deployment failed

Check the status of your deployment, manage resources, or troub
next time.

Deployment name: exampledeploy
Subscription: Third Internal Consumption
Resource group: demogroup

DEPLOYMENT DETAILS (Download)
Start time: 5/13/2019, 2:42:39 PM
Duration: 2 seconds
Correlation ID: 7b47b9be-0201-4e0e-b668-596017fd4885

9. To learn more about the step that failed, select **Operation details**.



The AccountType Standard_LRS1 is invalid. Click here for details →

Your deployment failed

Check the status of your deployment, manage resources, or troubleshoot deployment issues. Pin this page to your next time.

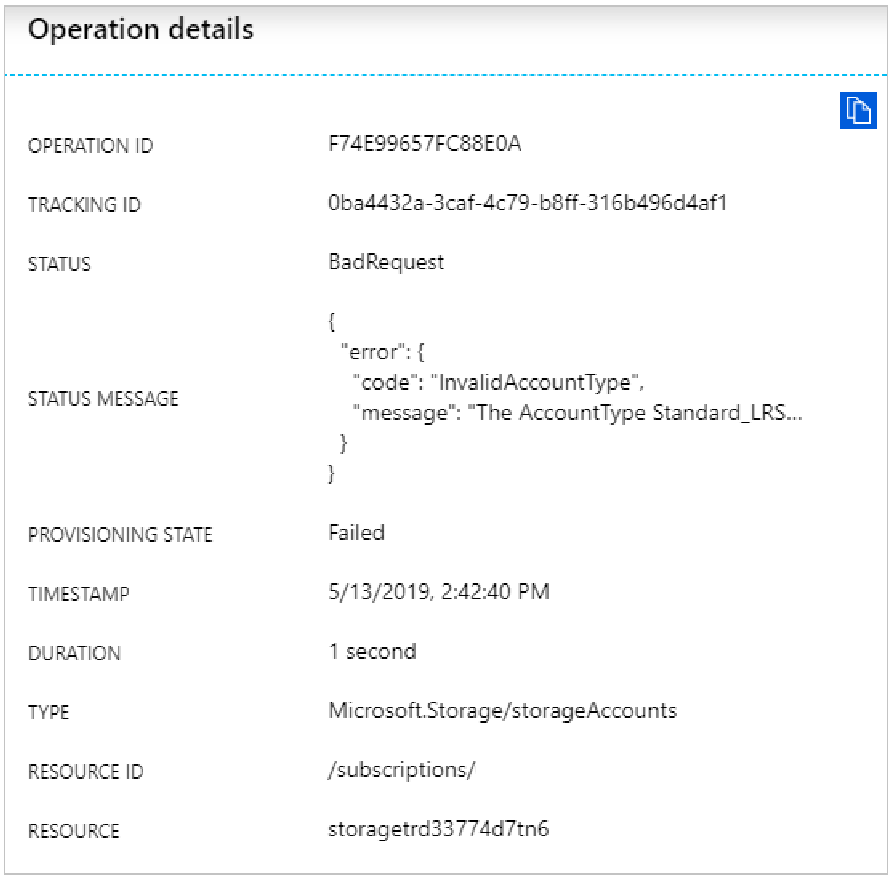
Deployment name: exampledeploy
Subscription: Third Internal Consumption
Resource group: demogroup

DEPLOYMENT DETAILS (Download)

Start time: 5/13/2019, 2:42:39 PM
Duration: 2 seconds
Correlation ID: 7b47b9be-0201-4e0e-b668-596017fd4885

RESOURCE	TYPE	STATUS	OPERATION DETAILS
storagetrd33774d7tn6	Microsoft.Storage/storageAccounts	BadRequest	Operation details

10. You see the details for that step of the deployment.



Operation details

OPERATION ID	F74E99657FC88E0A
TRACKING ID	0ba4432a-3caf-4c79-b8ff-316b496d4af1
STATUS	BadRequest
STATUS MESSAGE	{ "error": { "code": "InvalidAccountType", "message": "The AccountType Standard_LRS... } }
PROVISIONING STATE	Failed
TIMESTAMP	5/13/2019, 2:42:40 PM
DURATION	1 second
TYPE	Microsoft.Storage/storageAccounts
RESOURCE ID	/subscriptions/
RESOURCE	storagetrd33774d7tn6

PowerShell

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

To get the overall status of a deployment, use the **Get-AzResourceGroupDeployment** command.

```
Get-AzResourceGroupDeployment -ResourceGroupName ExampleGroup
```

Or, you can filter the results for only those deployments that have failed.

```
Get-AzResourceGroupDeployment -ResourceGroupName ExampleGroup | Where-Object ProvisioningState -eq Failed
```

The correlation ID is used to track related events, and can be helpful when working with technical support to troubleshoot a deployment. To get the correlation ID, use:

```
(Get-AzResourceGroupDeployment -ResourceGroupName ExampleGroup -DeploymentName azuredeploy).CorrelationId
```

Each deployment includes multiple operations. Each operation represents a step in the deployment process. To discover what went wrong with a deployment, you usually need to see details about the deployment operations. You can see the status of the operations with **Get-AzResourceGroupDeploymentOperation**.

```
Get-AzResourceGroupDeploymentOperation -ResourceGroupName ExampleGroup -DeploymentName azuredeploy
```

Which returns multiple operations with each one in the following format:

```
Id             :
/subscriptions/{guid}/resourceGroups/ExampleGroup/providers/Microsoft.Resources/deployments/Microsoft.Template
/operations/A3EB2DA598E0A780
OperationId    : A3EB2DA598E0A780
Properties     : @{provisioningOperation=Create; provisioningState=Succeeded; timestamp=2019-05-
13T21:42:40.7151512Z;
                duration=PT23.0227078S; trackingId=11d376e8-5d6d-4da8-847e-6f23c6443fbf;
                serviceRequestId=0196828d-8559-4bf6-b6b8-8b9057cb0e23; statusCode=OK; targetResource=}
PropertiesText : {duration:PT23.0227078S, provisioningOperation:Create, provisioningState:Succeeded,
                serviceRequestId:0196828d-8559-4bf6-b6b8-8b9057cb0e23...}
```

To get more details about failed operations, retrieve the properties for operations with **Failed** state.

```
(Get-AzResourceGroupDeploymentOperation -DeploymentName azuredeploy -ResourceGroupName
ExampleGroup).Properties | Where-Object ProvisioningState -eq Failed
```

Which returns all the failed operations with each one in the following format:

```
provisioningOperation : Create
provisioningState     : Failed
timestamp             : 2019-05-13T21:42:40.7151512Z
duration              : PT3.1449887S
trackingId            : f4ed72f8-4203-43dc-958a-15d041e8c233
serviceRequestId      : a426f689-5d5a-448d-a2f0-9784d14c900a
statusCode            : BadRequest
statusMessage         : @{error=}
targetResource        : @{id=/subscriptions/{guid}/resourceGroups/ExampleGroup/providers/
                         Microsoft.Network/publicIPAddresses/myPublicIP;
                         resourceType=Microsoft.Network/publicIPAddresses; resourceName=myPublicIP}
```

Note the serviceRequestId and the trackingId for the operation. The serviceRequestId can be helpful when working with technical support to troubleshoot a deployment. You'll use the trackingId in the next step to focus on a particular operation.

To get the status message of a particular failed operation, use the following command:

```
((Get-AzResourceGroupDeploymentOperation -DeploymentName azuredeploy -ResourceGroupName
ExampleGroup).Properties | Where-Object trackingId -eq f4ed72f8-4203-43dc-958a-
15d041e8c233).StatusMessage.error
```

Which returns:

```
code           message                                                                        details
----           -------                                                                        -------
DnsRecordInUse DNS record dns.westus.cloudapp.azure.com is already used by another public IP. {}
```

Every deployment operation in Azure includes request and response content. During deployment, you can use **DeploymentDebugLogLevel** parameter to specify that the request and/or response are logged.

You get that information from the log, and save it locally by using the following PowerShell commands:

```
(Get-AzResourceGroupDeploymentOperation -DeploymentName "TestDeployment" -ResourceGroupName "Test-
RG").Properties.request | ConvertTo-Json |  Out-File -FilePath <PathToFile>

(Get-AzResourceGroupDeploymentOperation -DeploymentName "TestDeployment" -ResourceGroupName "Test-
RG").Properties.response | ConvertTo-Json |  Out-File -FilePath <PathToFile>
```

Azure CLI

To get the overall status of a deployment, use the **azure group deployment show** command.

```
az group deployment show -g ExampleGroup -n ExampleDeployment
```

The correlation ID is used to track related events, and can be helpful when working with technical support to troubleshoot a deployment.

```
az group deployment show -g ExampleGroup -n ExampleDeployment --query properties.correlationId
```

To see the operations for a deployment, use:

```
az group deployment operation list -g ExampleGroup -n ExampleDeployment
```

REST

To get information about a deployment, use the Get information about a template deployment operation.

```
GET https://management.azure.com/subscriptions/{subscription-id}/resourcegroups/{resource-group-
name}/providers/microsoft.resources/deployments/{deployment-name}?api-version={api-version}
```

In the response, note in particular the **provisioningState**, **correlationId**, and **error** elements. The **correlationId** is used to track related events, and can be helpful when working with technical support to troubleshoot a deployment.

```
{
 ...
 "properties": {
   "provisioningState":"Failed",
   "correlationId":"d5062e45-6e9f-4fd3-a0a0-6b2c56b15757",
   ...
   "error":{
     "code":"DeploymentFailed","message":"At least one resource deployment operation failed. Please list
deployment operations for details. Please see https://aka.ms/arm-debug for usage details.",
     "details":[{"code":"Conflict","message":"{\r\n  \"error\": {\r\n    \"message\": \"Conflict\",\r\n
\"code\": \"Conflict\"\r\n  }\r\n}"}]
   }
 }
}
```

To get information about deployments, use List all template deployment operations.

```
GET https://management.azure.com/subscriptions/{subscription-id}/resourcegroups/{resource-group-
name}/providers/microsoft.resources/deployments/{deployment-name}/operations?$skiptoken={skiptoken}&api-
version={api-version}
```

The response includes request and/or response information based on what you specified in the **debugSetting** property during deployment.

```
{
 ...
 "properties":
 {
   ...
   "request":{
     "content":{
       "location":"West US",
       "properties":{
         "accountType": "Standard_LRS"
       }
     }
   },
   "response":{
     "content":{
       "error":{
         "message":"Conflict","code":"Conflict"
       }
     }
   }
 }
}
```

Next steps

- For help with resolving particular deployment errors, see Resolve common errors when deploying resources to Azure with Azure Resource Manager.
- To learn about using the activity logs to monitor other types of actions, see View activity logs to manage Azure resources.
- To validate your deployment before executing it, see Deploy a resource group with Azure Resource Manager template.

Troubleshoot common Azure deployment errors with Azure Resource Manager

6/18/2019 • 9 minutes to read • Edit Online

This article describes some common Azure deployment errors, and provides information to resolve the errors. If you can't find the error code for your deployment error, see Find error code.

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

Error codes

ERROR CODE	MITIGATION	MORE INFORMATION
AccountNameInvalid	Follow naming restrictions for storage accounts.	Resolve storage account name
AccountPropertyCannotBeSet	Check available storage account properties.	storageAccounts
AllocationFailed	The cluster or region doesn't have resources available or can't support the requested VM size. Retry the request at a later time, or request a different VM size.	Provisioning and allocation issues for Linux, Provisioning and allocation issues for Windows and Troubleshoot allocation failures
AnotherOperationInProgress	Wait for concurrent operation to complete.	
AuthorizationFailed	Your account or service principal doesn't have sufficient access to complete the deployment. Check the role your account belongs to, and its access for the deployment scope. You may receive this error when a required resource provider isn't registered.	Azure Role-Based Access Control Resolve registration
BadRequest	You sent deployment values that don't match what is expected by Resource Manager. Check the inner status message for help with troubleshooting.	Template reference and Supported locations

ERROR CODE	MITIGATION	MORE INFORMATION
Conflict	You're requesting an operation that isn't allowed in the resource's current state. For example, disk resizing is allowed only when creating a VM or when the VM is deallocated.	
DeploymentActive	Wait for concurrent deployment to this resource group to complete.	
DeploymentFailed	The DeploymentFailed error is a general error that doesn't provide the details you need to solve the error. Look in the error details for an error code that provides more information.	Find error code
DeploymentQuotaExceeded	If you reach the limit of 800 deployments per resource group, delete deployments from the history that are no longer needed. You can delete entries from the history with az group deployment delete for Azure CLI, or Remove-AzResourceGroupDeployment in PowerShell. Deleting an entry from the deployment history doesn't affect the deploy resources.	
DnsRecordInUse	The DNS record name must be unique. Either provide a different name, or modify the existing record.	
ImageNotFound	Check VM image settings.	
InUseSubnetCannotBeDeleted	You may get this error when trying to update a resource, but the request is processed by deleting and creating the resource. Make sure to specify all unchanged values.	Update resource
InvalidAuthenticationTokenTenant	Get access token for the appropriate tenant. You can only get the token from the tenant that your account belongs to.	
InvalidContentLink	You have most likely attempted to link to a nested template that isn't available. Double check the URI you provided for the nested template. If the template exists in a storage account, make sure the URI is accessible. You may need to pass a SAS token.	Linked templates

ERROR CODE	MITIGATION	MORE INFORMATION
InvalidParameter	One of the values you provided for a resource doesn't match the expected value. This error can result from many different conditions. For example, a password may be insufficient, or a blob name may be incorrect. Check the error message to determine which value needs to be corrected.	
InvalidRequestContent	Your deployment values either include values that aren't expected or are missing required values. Confirm the values for your resource type.	Template reference
InvalidRequestFormat	Enable debug logging when executing the deployment, and verify the contents of the request.	Debug logging
InvalidResourceNamespace	Check the resource namespace you specified in the **type** property.	Template reference
InvalidResourceReference	The resource either doesn't yet exist or is incorrectly referenced. Check whether you need to add a dependency. Verify that your use of the **reference** function includes the required parameters for your scenario.	Resolve dependencies
InvalidResourceType	Check the resource type you specified in the **type** property.	Template reference
InvalidSubscriptionRegistrationState	Register your subscription with the resource provider.	Resolve registration
InvalidTemplate	Check your template syntax for errors.	Resolve invalid template
InvalidTemplateCircularDependency	Remove unnecessary dependencies.	Resolve circular dependencies
LinkedAuthorizationFailed	Check if your account belongs to the same tenant as the resource group you're deploying to.	
LinkedInvalidPropertyId	The resource ID for a resource isn't resolving correctly. Check that you provide all required values for the resource ID, including subscription ID, resource group name, resource type, parent resource name (if needed), and resource name.	
LocationRequired	Provide a location for your resource.	Set location
MismatchingResourceSegments	Make sure nested resource has correct number of segments in name and type.	Resolve resource segments

ERROR CODE	MITIGATION	MORE INFORMATION
MissingRegistrationForLocation	Check resource provider registration status, and supported locations.	Resolve registration
MissingSubscriptionRegistration	Register your subscription with the resource provider.	Resolve registration
NoRegisteredProviderFound	Check resource provider registration status.	Resolve registration
NotFound	You may be attempting to deploy a dependent resource in parallel with a parent resource. Check if you need to add a dependency.	Resolve dependencies
OperationNotAllowed	The deployment is attempting an operation that exceeds the quota for the subscription, resource group, or region. If possible, revise your deployment to stay within the quotas. Otherwise, consider requesting a change to your quotas.	Resolve quotas
ParentResourceNotFound	Make sure a parent resource exists before creating the child resources.	Resolve parent resource
PasswordTooLong	You may have selected a password with too many characters, or may have converted your password value to a secure string before passing it as a parameter. If the template includes a **secure string** parameter, you don't need to convert the value to a secure string. Provide the password value as text.	
PrivateIPAddressInReservedRange	The specified IP address includes an address range required by Azure. Change IP address to avoid reserved range.	IP addresses
PrivateIPAddressNotInSubnet	The specified IP address is outside of the subnet range. Change IP address to fall within subnet range.	IP addresses
PropertyChangeNotAllowed	Some properties cannot be changed on a deployed resource. When updating a resource, limit your changes to permitted properties.	Update resource
RequestDisallowedByPolicy	Your subscription includes a resource policy that prevents an action you are trying to perform during deployment. Find the policy that blocks the action. If possible, modify your deployment to meet the limitations from the policy.	Resolve policies

ERROR CODE	MITIGATION	MORE INFORMATION
ReservedResourceName	Provide a resource name that doesn't include a reserved name.	Reserved resource names
ResourceGroupBeingDeleted	Wait for deletion to complete.	
ResourceGroupNotFound	Check the name of the target resource group for the deployment. It must already exist in your subscription. Check your subscription context.	Azure CLI PowerShell
ResourceNotFound	Your deployment references a resource that can't be resolved. Verify that your use of the **reference** function includes the parameters required for your scenario.	Resolve references
ResourceQuotaExceeded	The deployment is trying to create resources that exceed the quota for the subscription, resource group, or region. If possible, revise your infrastructure to stay within the quotas. Otherwise, consider requesting a change to your quotas.	Resolve quotas
SkuNotAvailable	Select SKU (such as VM size) that is available for the location you've selected.	Resolve SKU
StorageAccountAlreadyExists	Provide a unique name for the storage account.	Resolve storage account name
StorageAccountAlreadyTaken	Provide a unique name for the storage account.	Resolve storage account name
StorageAccountNotFound	Check the subscription, resource group, and name of the storage account you're trying to use.	
SubnetsNotInSameVnet	A virtual machine can only have one virtual network. When deploying several NICs, make sure they belong to the same virtual network.	Multiple NICs
TemplateResourceCircularDependency	Remove unnecessary dependencies.	Resolve circular dependencies
TooManyTargetResourceGroups	Reduce number of resource groups for a single deployment.	Cross resource group deployment

Find error code

There are two types of errors you can receive:

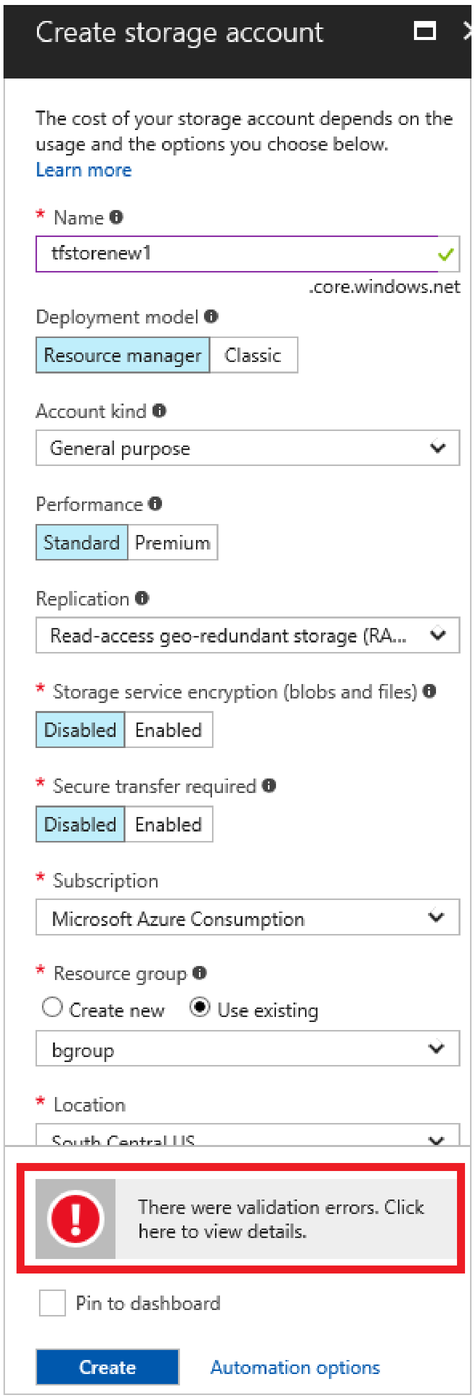
- validation errors
- deployment errors

Validation errors arise from scenarios that can be determined before deployment. They include syntax errors in your template, or trying to deploy resources that would exceed your subscription quotas. Deployment errors arise from conditions that occur during the deployment process. They include trying to access a resource that is being deployed in parallel.

Both types of errors return an error code that you use to troubleshoot the deployment. Both types of errors appear in the activity log. However, validation errors don't appear in your deployment history because the deployment never started.

Validation errors

When deploying through the portal, you see a validation error after submitting your values.



Create storage account

The cost of your storage account depends on the usage and the options you choose below.
Learn more

* Name: tfstorenew1 .core.windows.net

Deployment model: [✓ Resource manager] Classic

Account kind: General purpose

Performance: [✓ Standard] Premium

Replication: Read-access geo-redundant storage (RA...

* Storage service encryption (blobs and files): [✓ Disabled] Enabled

* Secure transfer required: [✓ Disabled] Enabled

* Subscription: Microsoft Azure Consumption

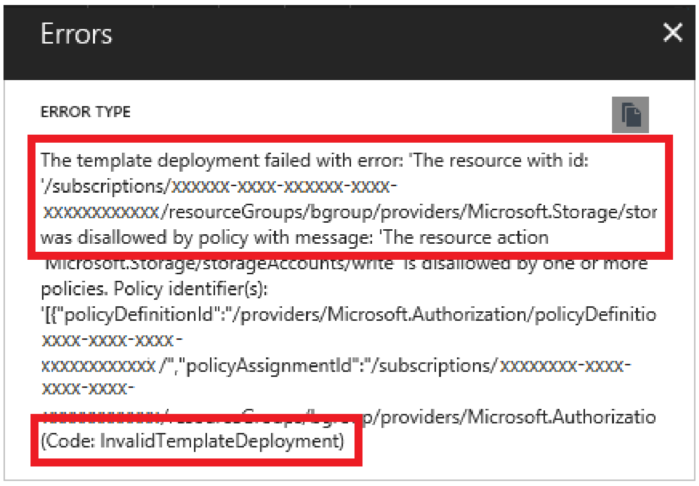
* Resource group: Create new [✓ Use existing]

bgroup

* Location: South Central US

There were validation errors. Click here to view details.

Select the message for more details. In the following image, you see an **InvalidTemplateDeployment** error and a message that indicates a policy blocked deployment.


Errors

ERROR TYPE

The template deployment failed with error: 'The resource with id: '/subscriptions/xxxxxx-xxxx-xxxxxx-xxxx-xxxxxxxxxxxx/resourceGroups/bgroup/providers/Microsoft.Storage/stor
was disallowed by policy with message: 'The resource action 'Microsoft.Storage/storageAccounts/write' is disallowed by one or more policies. Policy identifier(s): '[{"policyDefinitionId":"/providers/Microsoft.Authorization/policyDefinitio
xxxx-xxxx-xxxx-xxxxxxxxxxxx/","policyAssignmentId":"/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/resourceGroups/bgroup/providers/Microsoft.Authorizatio
(Code: InvalidTemplateDeployment)

Deployment errors

When the operation passes validation, but fails during deployment, you get a deployment error.

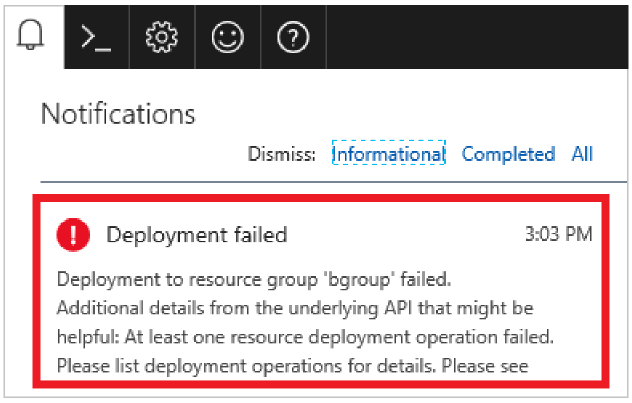
To see deployment error codes and messages with PowerShell, use:

```
(Get-AzResourceGroupDeploymentOperation -DeploymentName exampledeployment -ResourceGroupName
examplegroup).Properties.statusMessage
```

To see deployment error codes and messages with Azure CLI, use:

```
az group deployment operation list --name exampledeployment -g examplegroup --query "
[*].properties.statusMessage"
```

In the portal, select the notification.

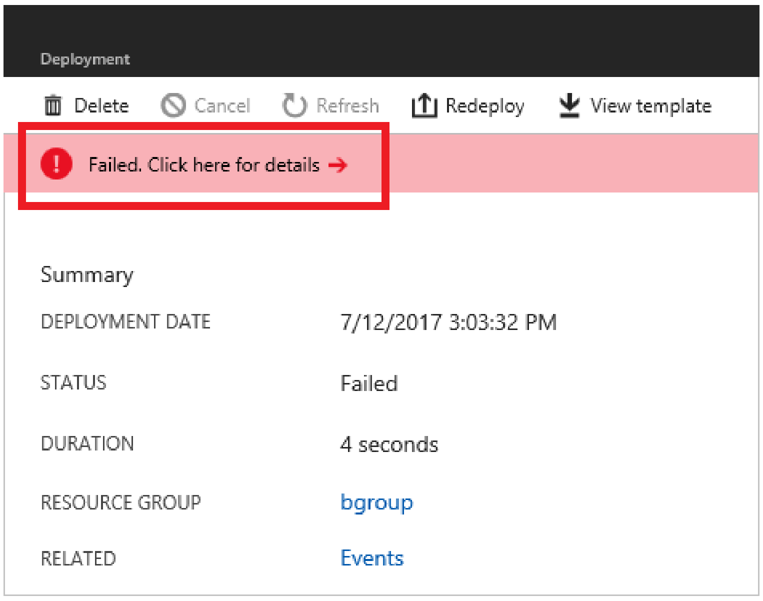

Notifications

Dismiss: Informational Completed All

Deployment failed 3:03 PM

Deployment to resource group 'bgroup' failed. Additional details from the underlying API that might be helpful: At least one resource deployment operation failed. Please list deployment operations for details. Please see

You see more details about the deployment. Select the option to find more information about the error.



Summary

DEPLOYMENT DATE 7/12/2017 3:03:32 PM

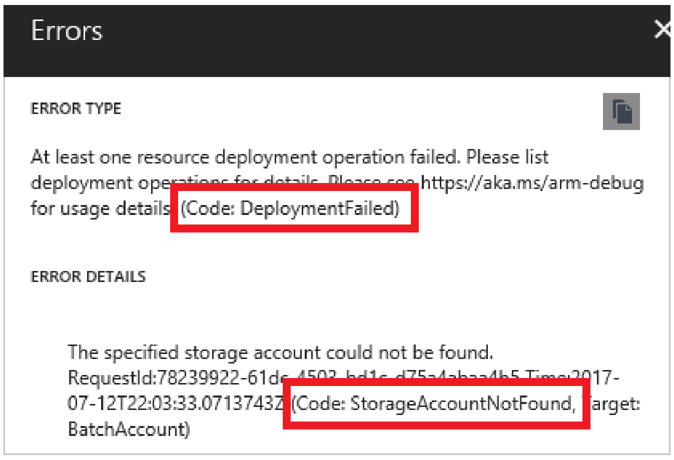
STATUS Failed

DURATION 4 seconds

RESOURCE GROUP bgroup

RELATED Events

You see the error message and error codes. Notice there are two error codes. The first error code (**DeploymentFailed**) is a general error that doesn't provide the details you need to solve the error. The second error code (**StorageAccountNotFound**) provides the details you need.



ERROR TYPE

At least one resource deployment operation failed. Please list deployment operations for details. Please see https://aka.ms/arm-debug for usage details (Code: DeploymentFailed)

ERROR DETAILS

The specified storage account could not be found. RequestId:78239922-61dc-4503-bd1c-d75a4aba4b5 Time:2017-07-12T22:03:33.0713743Z (Code: StorageAccountNotFound, Target: BatchAccount)

Enable debug logging

Sometimes you need more information about the request and response to learn what went wrong. During deployment, you can request that additional information is logged during a deployment.

PowerShell

In PowerShell, set the **DeploymentDebugLogLevel** parameter to All, ResponseContent, or RequestContent.

```
New-AzResourceGroupDeployment `
  -Name exampledeployment `
  -ResourceGroupName examplegroup `
  -TemplateFile c:\Azure\Templates\storage.json `
  -DeploymentDebugLogLevel All
```

Examine the request content with the following cmdlet:

```
(Get-AzResourceGroupDeploymentOperation `
-DeploymentName exampledeployment `
-ResourceGroupName examplegroup).Properties.request `
| ConvertTo-Json
```

Or, the response content with:

```
(Get-AzResourceGroupDeploymentOperation `
-DeploymentName exampledeployment `
-ResourceGroupName examplegroup).Properties.response `
| ConvertTo-Json
```

This information can help you determine whether a value in the template is being incorrectly set.

Azure CLI

Currently, Azure CLI doesn't support turning on debug logging, but you can retrieve debug logging.

Examine the deployment operations with the following command:

```
az group deployment operation list \
  --resource-group examplegroup \
  --name exampledeployment
```

Examine the request content with the following command:

```
az group deployment operation list \
  --name exampledeployment \
  -g examplegroup \
  --query [].properties.request
```

Examine the response content with the following command:

```
az group deployment operation list \
  --name exampledeployment \
  -g examplegroup \
  --query [].properties.response
```

Nested template

To log debug information for a nested template, use the **debugSetting** element.

```
{
    "apiVersion": "2016-09-01",
    "name": "nestedTemplate",
    "type": "Microsoft.Resources/deployments",
    "properties": {
        "mode": "Incremental",
        "templateLink": {
            "uri": "{template-uri}",
            "contentVersion": "1.0.0.0"
        },
        "debugSetting": {
           "detailLevel": "requestContent, responseContent"
        }
    }
}
```

Create a troubleshooting template

In some cases, the easiest way to troubleshoot your template is to test parts of it. You can create a simplified template that enables you to focus on the part that you believe is causing the error. For example, suppose you're receiving an error when referencing a resource. Rather than dealing with an entire template, create a template that returns the part that may be causing your problem. It can help you determine whether you're passing in the right parameters, using template functions correctly, and getting the resource you expect.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "storageName": {
        "type": "string"
    },
    "storageResourceGroup": {
        "type": "string"
    }
  },
  "variables": {},
  "resources": [],
  "outputs": {
    "exampleOutput": {
        "value": "[reference(resourceId(parameters('storageResourceGroup'),
'Microsoft.Storage/storageAccounts', parameters('storageName')), '2016-05-01')]",
        "type" : "object"
    }
  }
}
```

Or, suppose you are encountering deployment errors that you believe are related to incorrectly set dependencies. Test your template by breaking it into simplified templates. First, create a template that deploys only a single resource (like a SQL Server). When you are sure you have that resource correctly defined, add a resource that depends on it (like a SQL Database). When you have those two resources correctly defined, add other dependent resources (like auditing policies). In between each test deployment, delete the resource group to make sure you adequately testing the dependencies.

Next steps

- To go through a troubleshoot tutorial, see Tutorial: Troubleshoot Resource Manager template deployments
- To learn about auditing actions, see Audit operations with Resource Manager.
- To learn about actions to determine the errors during deployment, see View deployment operations.

Resolve errors for storage account names

5/21/2018 • 2 minutes to read • Edit Online

This article describes naming errors you may encounter when deploying a storage account.

Symptom

If your storage account name includes prohibited characters, you receive an error like:

```
Code=AccountNameInvalid
Message=S!torageckrexph7isnoc is not a valid storage account name. Storage account name must be
between 3 and 24 characters in length and use numbers and lower-case letters only.
```

For storage accounts, you must provide a name for the resource that is unique across Azure. If you do not provide a unique name, you receive an error like:

```
Code=StorageAccountAlreadyTaken
Message=The storage account named mystorage is already taken.
```

If you deploy a storage account with the same name as an existing storage account in your subscription, but provide a different location, you receive an error indicating the storage account already exists in a different location. Either delete the existing storage account, or provide the same location as the existing storage account.

Cause

Storage account names must be between 3 and 24 characters in length and use numbers and lower-case letters only. The name must be unique.

Solution

Make sure the storage account name is unique. You can create a unique name by concatenating your naming convention with the result of the uniqueString function.

```
"name": "[concat('storage', uniqueString(resourceGroup().id))]",
"type": "Microsoft.Storage/storageAccounts",
```

Make sure your storage account name does not exceed 24 characters. The uniqueString function returns 13 characters. If you concatenate a prefix or postfix to the **uniqueString** result, provide a value that is 11 characters or less.

```
"parameters": {
    "storageNamePrefix": {
      "type": "string",
      "maxLength": 11,
      "defaultValue": "storage",
      "metadata": {
        "description": "The value to use for starting the storage account name."
      }
    }
}
```

Make sure your storage account name does not include any upper-case letters or special characters.

Resolve errors for invalid template

6/18/2019 • 4 minutes to read • Edit Online

This article describes how to resolve invalid template errors.

Symptom

When deploying a template, you receive an error indicating:

```
Code=InvalidTemplate
Message=<varies>
```

The error message depends on the type of error.

Cause

This error can result from several different types of errors. They usually involve a syntax or structural error in the template.

Solution 1 - syntax error

If you receive an error message that indicates the template failed validation, you may have a syntax problem in your template.

```
Code=InvalidTemplate
Message=Deployment template validation failed
```

This error is easy to make because template expressions can be intricate. For example, the following name assignment for a storage account has one set of brackets, three functions, three sets of parentheses, one set of single quotes, and one property:

```
"name": "[concat('storage', uniqueString(resourceGroup().id))]",
```

If you don't provide the matching syntax, the template produces a value that is different than your intention.

When you receive this type of error, carefully review the expression syntax. Consider using a JSON editor like Visual Studio or Visual Studio Code, which can warn you about syntax errors.

Solution 2 - incorrect segment lengths

Another invalid template error occurs when the resource name isn't in the correct format.

```
Code=InvalidTemplate
Message=Deployment template validation failed: 'The template resource {resource-name}'
for type {resource-type} has incorrect segment lengths.
```

A root level resource must have one less segment in the name than in the resource type. Each segment is differentiated by a slash. In the following example, the type has two segments and the name has one segment, so it's a **valid name**.

```
{
  "type": "Microsoft.Web/serverfarms",
  "name": "myHostingPlanName",
  ...
}
```

But the next example is **not a valid name** because it has the same number of segments as the type.

```
{
  "type": "Microsoft.Web/serverfarms",
  "name": "appPlan/myHostingPlanName",
  ...
}
```

For child resources, the type and name have the same number of segments. This number of segments makes sense because the full name and type for the child includes the parent name and type. Therefore, the full name still has one less segment than the full type.

```
"resources": [
    {
        "type": "Microsoft.KeyVault/vaults",
        "name": "contosokeyvault",
        ...
        "resources": [
            {
                "type": "secrets",
                "name": "appPassword",
                ...
            }
        ]
    }
]
```

Getting the segments right can be tricky with Resource Manager types that are applied across resource providers. For example, applying a resource lock to a web site requires a type with four segments. Therefore, the name is three segments:

```
{
    "type": "Microsoft.Web/sites/providers/locks",
    "name": "[concat(variables('siteName'),'/Microsoft.Authorization/MySiteLock')]",
    ...
}
```

Solution 3 - parameter is not valid

If you provide a parameter value that is not one of the allowed values, you receive a message similar to the following error:

```
Code=InvalidTemplate;
Message=Deployment template validation failed: 'The provided value {parameter value}
for the template parameter {parameter name} is not valid. The parameter value is not
part of the allowed values
```

Double check the allowed values in the template, and provide one during deployment. For more information about allowed parameter values, see Parameters section of Azure Resource Manager templates.

Solution 4 - Too many target resource groups

If you specify more than five target resource groups in a single deployment, you receive this error. Consider either consolidating the number of resource groups in your deployment, or deploying some of the templates as separate deployments. For more information, see Deploy Azure resources to more than one subscription or resource group.

Solution 5 - circular dependency detected

You receive this error when resources depend on each other in a way that prevents the deployment from starting. A combination of interdependencies makes two or more resource wait for other resources that are also waiting. For example, resource1 depends on resource3, resource2 depends on resource1, and resource3 depends on resource2. You can usually solve this problem by removing unnecessary dependencies.

To solve a circular dependency:

1. In your template, find the resource identified in the circular dependency.
2. For that resource, examine the **dependsOn** property and any uses of the **reference** function to see which resources it depends on.
3. Examine those resources to see which resources they depend on. Follow the dependencies until you notice a resource that depends on the original resource.
4. For the resources involved in the circular dependency, carefully examine all uses of the **dependsOn** property to identify any dependencies that are not needed. Remove those dependencies. If you are unsure that a dependency is needed, try removing it.
5. Redeploy the template.

Removing values from the **dependsOn** property can cause errors when you deploy the template. If you get an error, add the dependency back into the template.

If that approach doesn't solve the circular dependency, consider moving part of your deployment logic into child resources (such as extensions or configuration settings). Configure those child resources to deploy after the resources involved in the circular dependency. For example, suppose you're deploying two virtual machines but you must set properties on each one that refer to the other. You can deploy them in the following order:

1. vm1
2. vm2
3. Extension on vm1 depends on vm1 and vm2. The extension sets values on vm1 that it gets from vm2.
4. Extension on vm2 depends on vm1 and vm2. The extension sets values on vm2 that it gets from vm1.

The same approach works for App Service apps. Consider moving configuration values into a child resource of the app resource. You can deploy two web apps in the following order:

1. webapp1
2. webapp2
3. config for webapp1 depends on webapp1 and webapp2. It contains app settings with values from webapp2.
4. config for webapp2 depends on webapp1 and webapp2. It contains app settings with values from webapp1.

2 minutes to read

Resolve errors for resource provider registration

2/18/2019 • 2 minutes to read • Edit Online

This article describes the errors you may encounter when using a resource provider that you haven't previously used in your subscription.

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

Symptom

When deploying resource, you may receive the following error code and message:

```
Code: NoRegisteredProviderFound
Message: No registered resource provider found for location {location}
and API version {api-version} for type {resource-type}.
```

Or, you may receive a similar message that states:

```
Code: MissingSubscriptionRegistration
Message: The subscription is not registered to use namespace {resource-provider-namespace}
```

The error message should give you suggestions for the supported locations and API versions. You can change your template to one of the suggested values. Most providers are registered automatically by the Azure portal or the command-line interface you're using, but not all. If you haven't used a particular resource provider before, you may need to register that provider.

Or, when disabling auto-shutdown for virtual machines, you may receive an error message similar to:

```
Code: AuthorizationFailed
Message: The client '<identifier>' with object id '<identifier>' does not have authorization to perform
action 'Microsoft.Compute/virtualMachines/read' over scope ...
```

Cause

You receive these errors for one of these reasons:

- The required resource provider hasn't been registered for your subscription
- API version not supported for the resource type
- Location not supported for the resource type
- For auto-shutdown of VMs, the Microsoft.DevTestLab resource provider must be registered.

Solution 1 - PowerShell

For PowerShell, use **Get-AzResourceProvider** to see your registration status.

```
Get-AzResourceProvider -ListAvailable
```

To register a provider, use **Register-AzResourceProvider** and provide the name of the resource provider you wish to register.

```
Register-AzResourceProvider -ProviderNamespace Microsoft.Cdn
```

To get the supported locations for a particular type of resource, use:

```
((Get-AzResourceProvider -ProviderNamespace Microsoft.Web).ResourceTypes | Where-Object ResourceTypeName -eq
sites).Locations
```

To get the supported API versions for a particular type of resource, use:

```
((Get-AzResourceProvider -ProviderNamespace Microsoft.Web).ResourceTypes | Where-Object ResourceTypeName -eq
sites).ApiVersions
```

Solution 2 - Azure CLI

To see whether the provider is registered, use the `az provider list` command.

```
az provider list
```

To register a resource provider, use the `az provider register` command, and specify the *namespace* to register.

```
az provider register --namespace Microsoft.Cdn
```

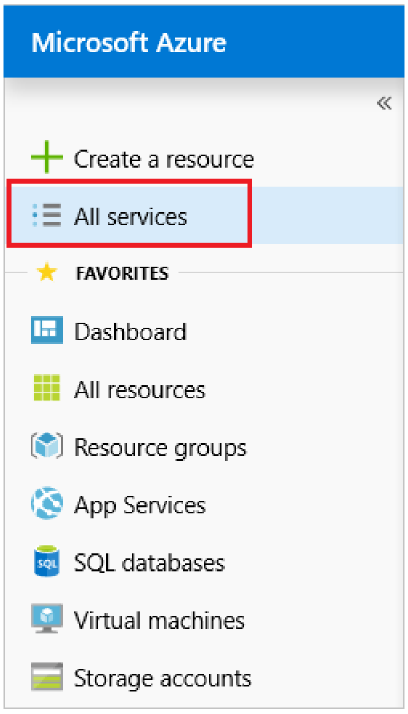
To see the supported locations and API versions for a resource type, use:

```
az provider show -n Microsoft.Web --query "resourceTypes[?resourceType=='sites'].locations"
```

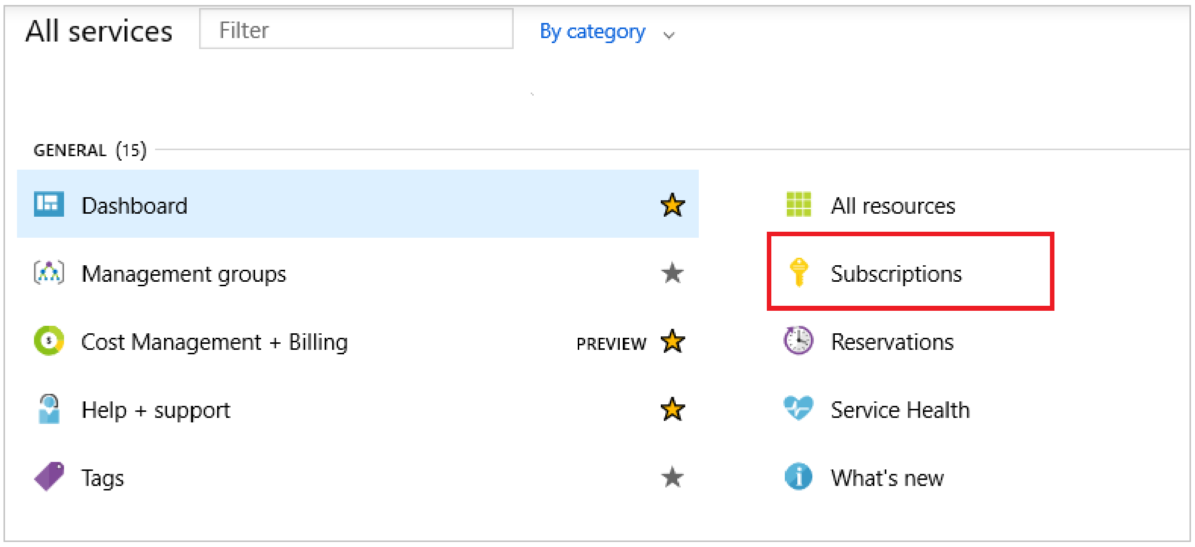
Solution 3 - Azure portal

You can see the registration status and register a resource provider namespace through the portal.

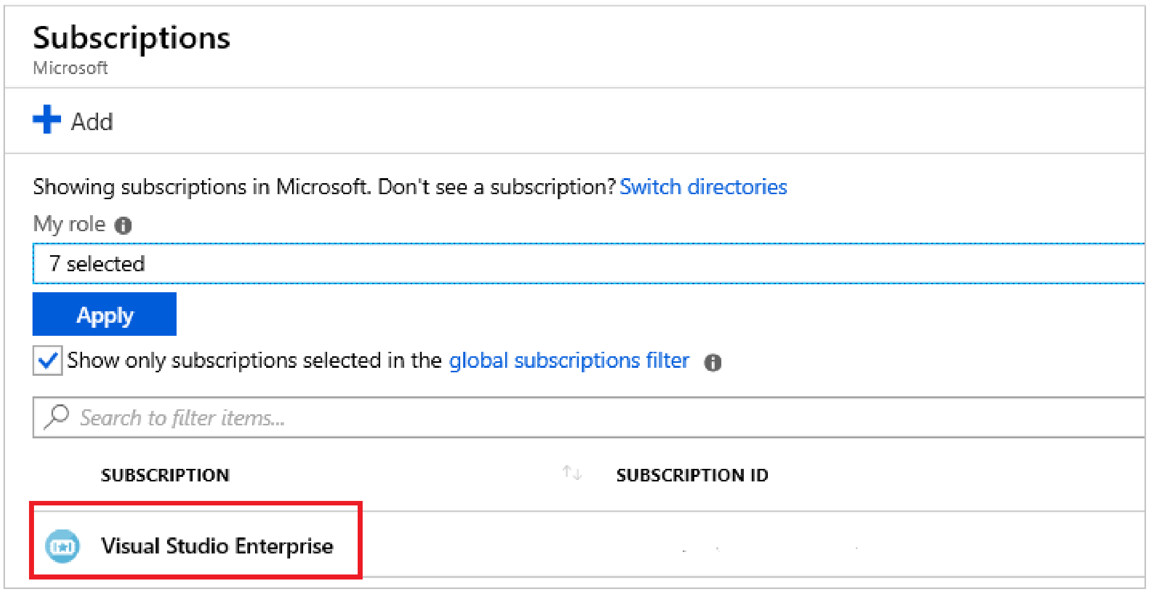
1. From the portal, select **All services**.



2. Select **Subscriptions**.



3. From the list of subscriptions, select the subscription you want to use for registering the resource provider.



Showing subscriptions in Microsoft. Don't see a subscription? Switch directories

My role

7 selected

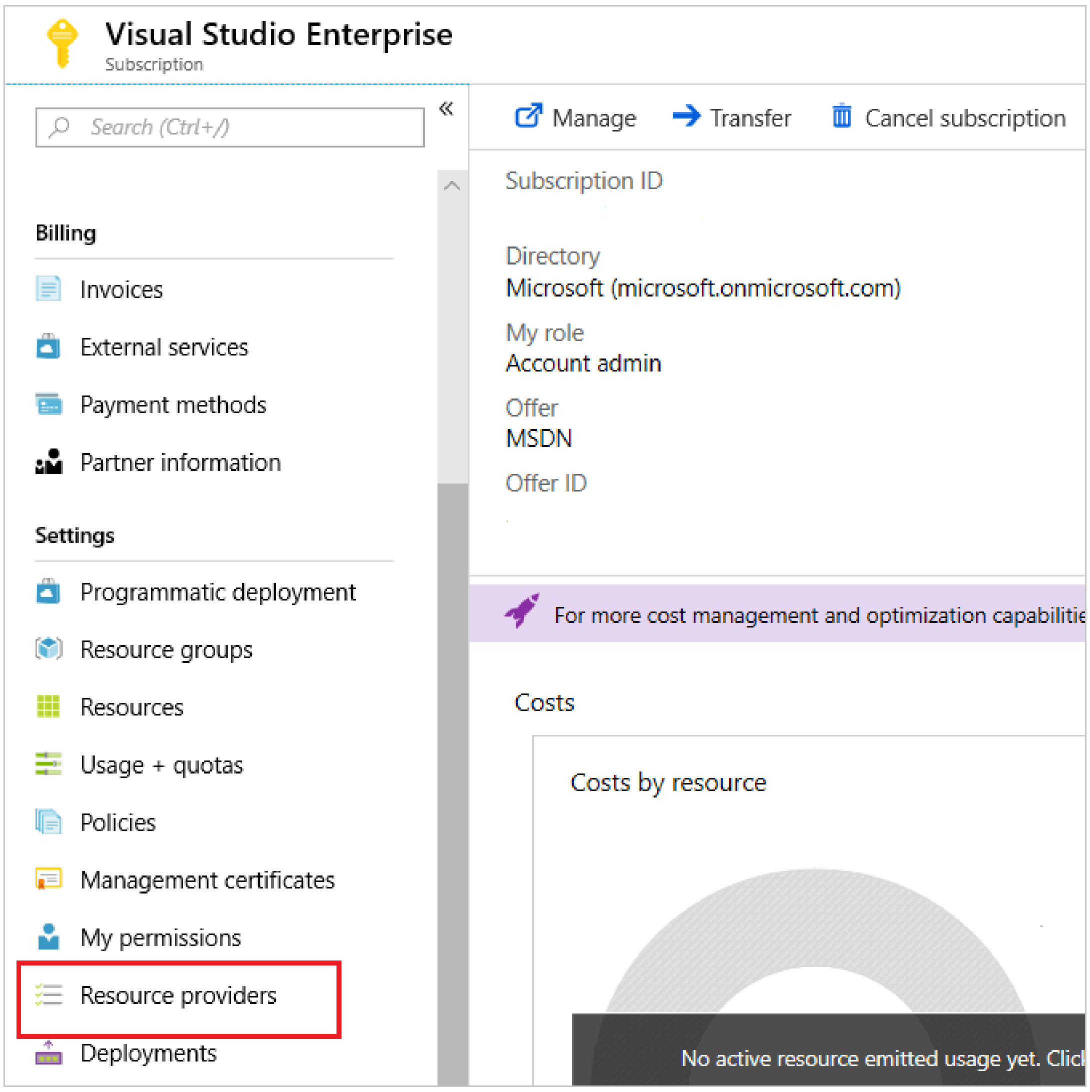
Apply

✓ Show only subscriptions selected in the global subscriptions filter

Search to filter items...

SUBSCRIPTION	SUBSCRIPTION ID
Visual Studio Enterprise	

4. For your subscription, select **Resource providers**.



Subscription ID

Directory
Microsoft (microsoft.onmicrosoft.com)

My role
Account admin

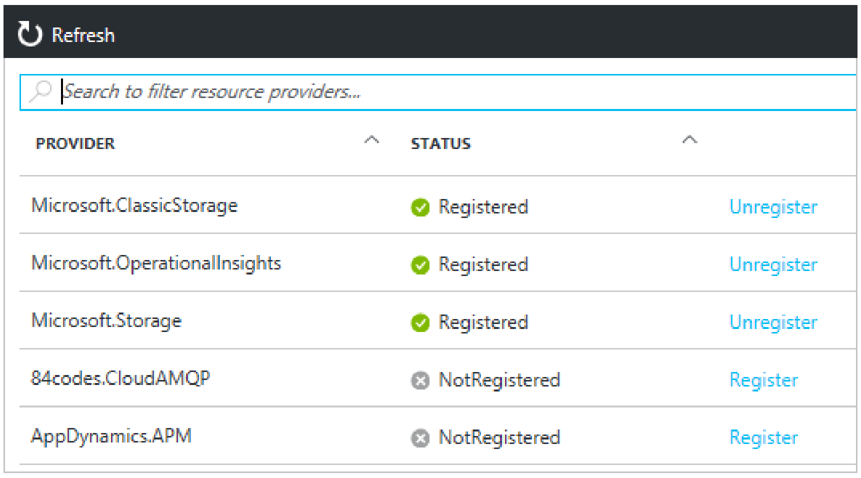
Offer
MSDN

Offer ID

For more cost management and optimization capabilitie

No active resource emitted usage yet. Clicl

5. Look at the list of resource providers, and if necessary, select the **Register** link to register the resource provider of the type you're trying to deploy.



PROVIDER	STATUS	
Microsoft.ClassicStorage	Registered	Unregister
Microsoft.OperationalInsights	Registered	Unregister
Microsoft.Storage	Registered	Unregister
84codes.CloudAMQP	NotRegistered	Register
AppDynamics.APM	NotRegistered	Register

Resolve not found errors for Azure resources

6/18/2019 • 3 minutes to read • Edit Online

This article describes the errors you may see when a resource can't be found during deployment.

Symptom

When your template includes the name of a resource that can't be resolved, you receive an error similar to:

```
Code=NotFound;
Message=Cannot find ServerFarm with name exampleplan.
```

If you use the reference or listKeys functions with a resource that can't be resolved, you receive the following error:

```
Code=ResourceNotFound;
Message=The Resource 'Microsoft.Storage/storageAccounts/{storage name}' under resource
group {resource group name} was not found.
```

Cause

Resource Manager needs to retrieve the properties for a resource, but can't identify the resource in your subscription.

Solution 1 - set dependencies

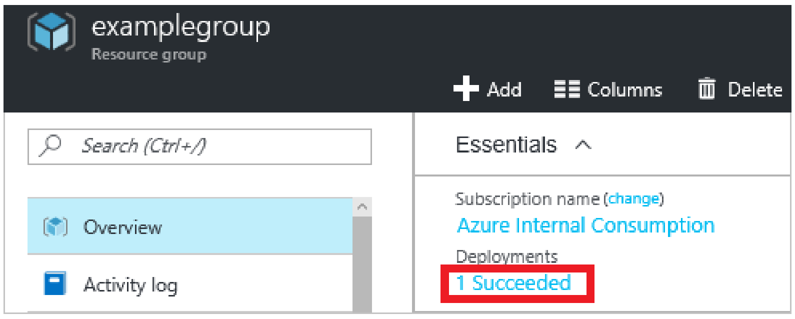
If you're trying to deploy the missing resource in the template, check whether you need to add a dependency. Resource Manager optimizes deployment by creating resources in parallel, when possible. If one resource must be deployed after another resource, you need to use the **dependsOn** element in your template. For example, when deploying a web app, the App Service plan must exist. If you haven't specified that the web app depends on the App Service plan, Resource Manager creates both resources at the same time. You get an error stating that the App Service plan resource can't be found, because it doesn't exist yet when attempting to set a property on the web app. You prevent this error by setting the dependency in the web app.

```
{
  "apiVersion": "2015-08-01",
  "type": "Microsoft.Web/sites",
  "dependsOn": [
    "[variables('hostingPlanName')]"
  ],
  ...
}
```

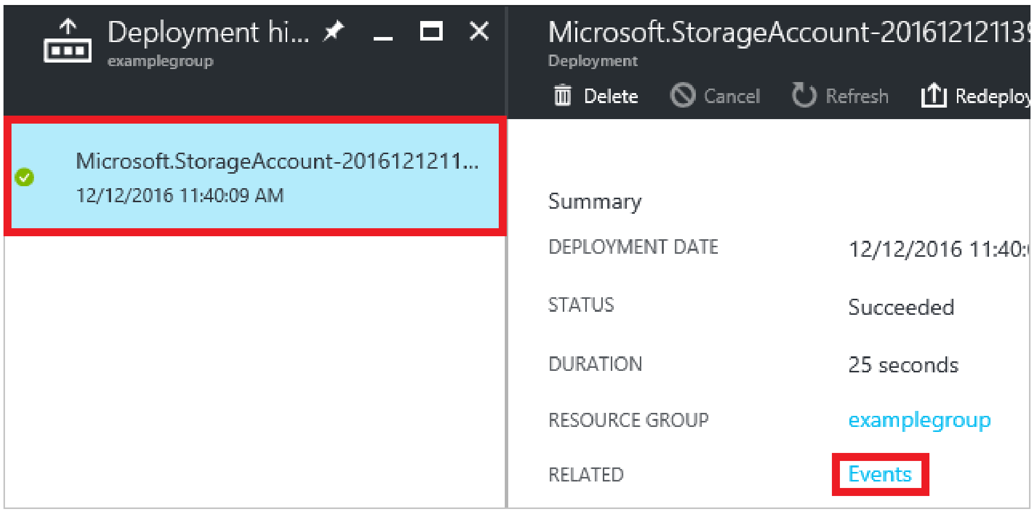
But, you want to avoid setting dependencies that aren't needed. When you have unnecessary dependencies, you prolong the duration of the deployment by preventing resources that aren't dependent on each other from being deployed in parallel. In addition, you may create circular dependencies that block the deployment. The reference function and list* functions creates an implicit dependency on the referenced resource, when that resource is deployed in the same template and is referenced by its name (not resource ID). Therefore, you may have more dependencies than the dependencies specified in the **dependsOn** property. The resourceId function doesn't create an implicit dependency or validate that the resource exists. The reference function and list* functions don't create an implicit dependency when the resource is referred to by its resource ID. To create an implicit dependency, pass the name of the resource that is deployed in the same template.

When you see dependency problems, you need to gain insight into the order of resource deployment. To view the order of deployment operations:

1. Select the deployment history for your resource group.



2. Select a deployment from the history, and select **Events**.

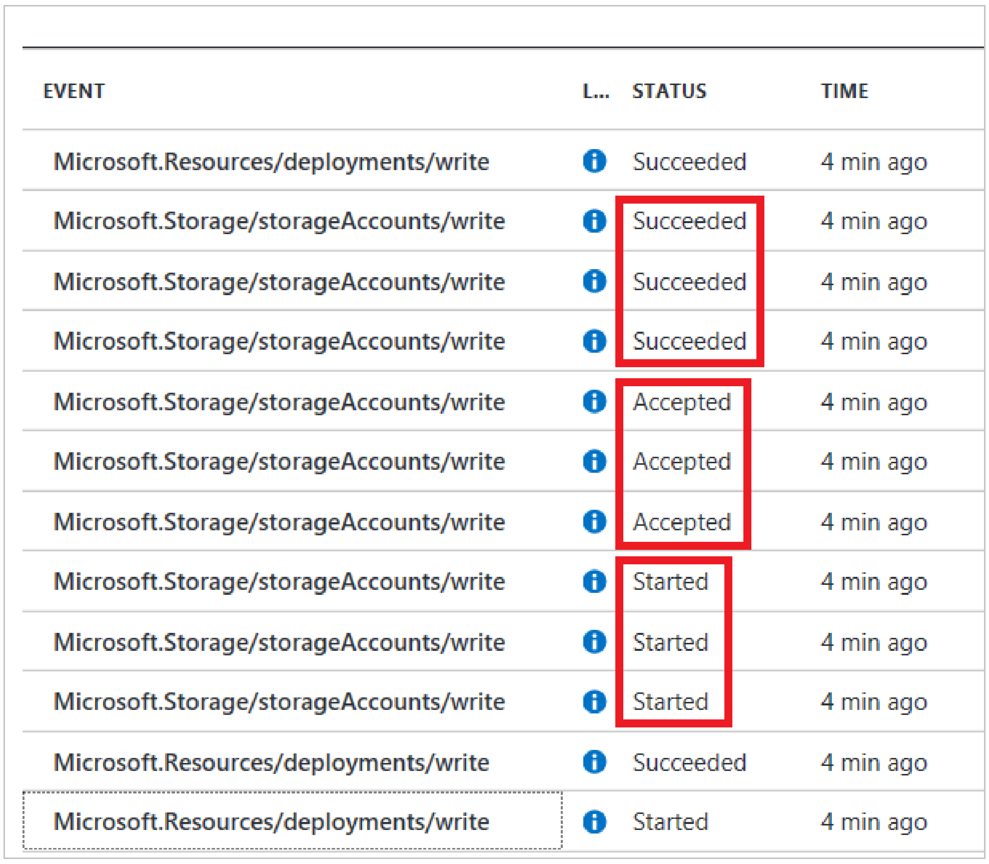


Microsoft.StorageAccount-2016121211...
12/12/2016 11:40:09 AM

Summary

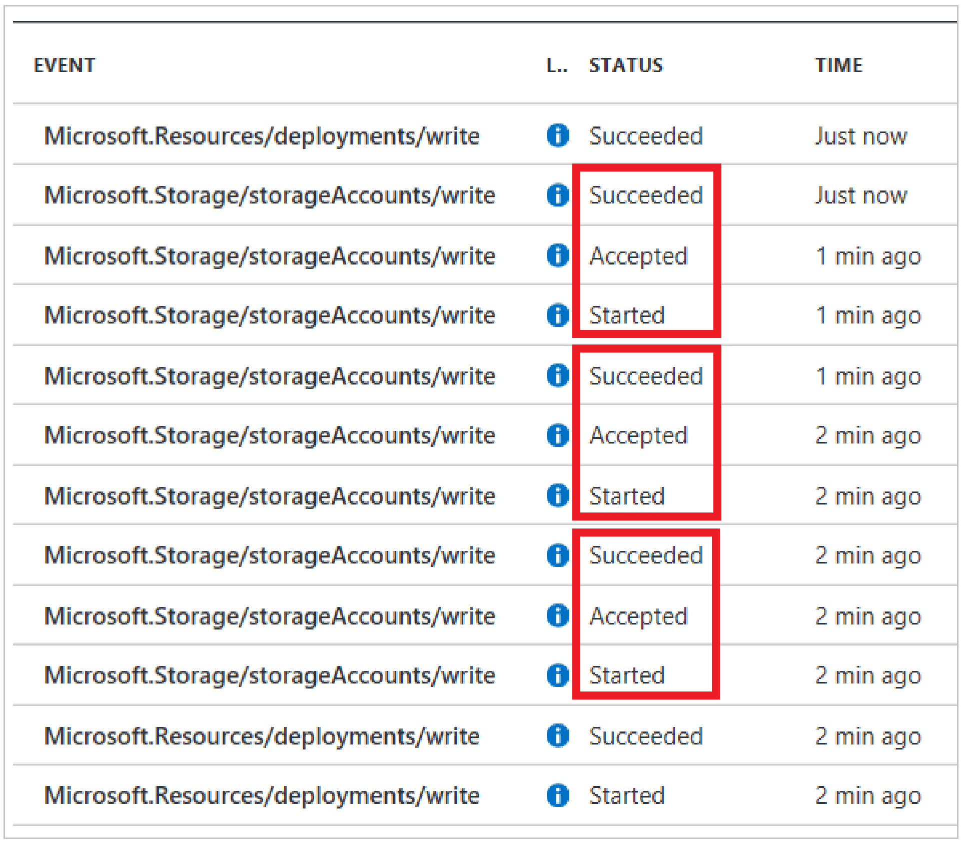
DEPLOYMENT DATE	12/12/2016 11:40:
STATUS	Succeeded
DURATION	25 seconds
RESOURCE GROUP	examplegroup
RELATED	Events

3. Examine the sequence of events for each resource. Pay attention to the status of each operation. For example, the following image shows three storage accounts that deployed in parallel. Notice that the three storage accounts are started at the same time.



EVENT	L...	STATUS	TIME
Microsoft.Resources/deployments/write	●	Succeeded	4 min ago
Microsoft.Storage/storageAccounts/write	●	Succeeded	4 min ago
Microsoft.Storage/storageAccounts/write	●	Succeeded	4 min ago
Microsoft.Storage/storageAccounts/write	●	Succeeded	4 min ago
Microsoft.Storage/storageAccounts/write	●	Accepted	4 min ago
Microsoft.Storage/storageAccounts/write	●	Accepted	4 min ago
Microsoft.Storage/storageAccounts/write	●	Accepted	4 min ago
Microsoft.Storage/storageAccounts/write	●	Started	4 min ago
Microsoft.Storage/storageAccounts/write	●	Started	4 min ago
Microsoft.Storage/storageAccounts/write	●	Started	4 min ago
Microsoft.Resources/deployments/write	●	Succeeded	4 min ago
Microsoft.Resources/deployments/write	●	Started	4 min ago

The next image shows three storage accounts that aren't deployed in parallel. The second storage account depends on the first storage account, and the third storage account depends on the second storage account. The first storage account is started, accepted, and completed before the next is started.



EVENT	L..	STATUS	TIME
Microsoft.Resources/deployments/write	●	Succeeded	Just now
Microsoft.Storage/storageAccounts/write	●	Succeeded	Just now
Microsoft.Storage/storageAccounts/write	●	Accepted	1 min ago
Microsoft.Storage/storageAccounts/write	●	Started	1 min ago
Microsoft.Storage/storageAccounts/write	●	Succeeded	1 min ago
Microsoft.Storage/storageAccounts/write	●	Accepted	2 min ago
Microsoft.Storage/storageAccounts/write	●	Started	2 min ago
Microsoft.Storage/storageAccounts/write	●	Succeeded	2 min ago
Microsoft.Storage/storageAccounts/write	●	Accepted	2 min ago
Microsoft.Storage/storageAccounts/write	●	Started	2 min ago
Microsoft.Resources/deployments/write	●	Succeeded	2 min ago
Microsoft.Resources/deployments/write	●	Started	2 min ago

Solution 2 - get resource from different resource group

When the resource exists in a different resource group than the one being deployed to, use the resourceId function to get the fully qualified name of the resource.

```
"properties": {
    "name": "[parameters('siteName')]",
    "serverFarmId": "[resourceId('plangroup', 'Microsoft.Web/serverfarms', parameters('hostingPlanName'))]"
}
```

Solution 3 - check reference function

Look for an expression that includes the reference function. The values you provide vary based on whether the resource is in the same template, resource group, and subscription. Double check that you're providing the required parameter values for your scenario. If the resource is in a different resource group, provide the full resource ID. For example, to reference a storage account in another resource group, use:

```
"[reference(resourceId('exampleResourceGroup', 'Microsoft.Storage/storageAccounts', 'myStorage'), '2017-06-01')]"
```

Resolve errors for parent resources

6/18/2019 • 2 minutes to read • Edit Online

This article describes the errors you may get when deploying a resource that is dependent on a parent resource.

Symptom

When deploying a resource that is a child to another resource, you may receive the following error:

```
Code=ParentResourceNotFound;
Message=Can not perform requested operation on nested resource. Parent resource 'exampleserver' not found."
```

Cause

When one resource is a child to another resource, the parent resource must exist before creating the child resource. The name of the child resource defines the connection with the parent resource. The name of the child resource is in the format `<parent-resource-name>/<child-resource-name>`. For example, a SQL Database might be defined as:

```
{
  "type": "Microsoft.Sql/servers/databases",
  "name": "[concat(variables('databaseServerName'), '/', parameters('databaseName'))]",
  ...
```

If you deploy both the server and the database in the same template, but don't specify a dependency on the server, the database deployment might start before the server has deployed.

If the parent resource already exists and isn't deployed in the same template, you get this error when Resource Manager can't associate the child resource with parent. This error might happen when the child resource isn't in the correct format, or the child resource is deployed to a resource group that is different than the resource group for parent resource.

Solution

To resolve this error when parent and child resources are deployed in the same template, include a dependency.

```
"dependsOn": [
    "[variables('databaseServerName')]"
]
```

To resolve this error when the parent resource was previously deployed in a different template, you don't set a dependency. Instead, deploy the child to the same resource group and provide the name of the parent resource.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "sqlServerName": {
            "type": "string"
        },
        "databaseName": {
            "type": "string"
        }
    },
    "resources": [
        {
            "apiVersion": "2014-04-01",
            "type": "Microsoft.Sql/servers/databases",
            "location": "[resourceGroup().location]",
            "name": "[concat(parameters('sqlServerName'), '/', parameters('databaseName'))]",
            "properties": {
                "collation": "SQL_Latin1_General_CP1_CI_AS",
                "edition": "Basic"
            }
        }
    ],
    "outputs": {}
}
```

For more information, see Define the order for deploying resources in Azure Resource Manager templates.

2 minutes to read

2 minutes to read

RequestDisallowedByPolicy error with Azure resource policy

3/26/2019 • 2 minutes to read • Edit Online

This article describes the cause of the RequestDisallowedByPolicy error, it also provides solution for this error.

Symptom

During deployment, you might receive a **RequestDisallowedByPolicy** error that prevents you from creating the resources. The following example shows the error:

```
{
  "statusCode": "Forbidden",
  "serviceRequestId": null,
  "statusMessage": "{\"error\":{\"code\":\"RequestDisallowedByPolicy\",\"message\":\"The resource action
'Microsoft.Network/publicIpAddresses/write' is disallowed by one or more policies. Policy identifier(s):
'/subscriptions/{guid}/providers/Microsoft.Authorization/policyDefinitions/regionPolicyDefinition'.\"}}",
  "responseBody": "{\"error\":{\"code\":\"RequestDisallowedByPolicy\",\"message\":\"The resource action
'Microsoft.Network/publicIpAddresses/write' is disallowed by one or more policies. Policy identifier(s):
'/subscriptions/{guid}/providers/Microsoft.Authorization/policyDefinitions/regionPolicyDefinition'.\"}}"
}
```

Troubleshooting

To retrieve details about the policy that blocked your deployment, use the following one of the methods:

PowerShell

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

In PowerShell, provide that policy identifier as the `Id` parameter to retrieve details about the policy that blocked your deployment.

```
(Get-AzPolicyDefinition -Id
"/subscriptions/{guid}/providers/Microsoft.Authorization/policyDefinitions/regionPolicyDefinition").Properties
.policyRule | ConvertTo-Json
```

Azure CLI

In Azure CLI, provide the name of the policy definition:

```
az policy definition show --name regionPolicyAssignment
```

Solution

For security or compliance, your subscription administrators might assign policies that limit how resources are deployed. For example, your subscription might have a policy that prevents creating Public IP addresses, Network Security Groups, User-Defined Routes, or route tables. The error message in the **Symptoms** section shows the name of the policy. To resolve this problem, review the resource policies, and determine how to deploy resources that comply with those policies.

For more information, see the following articles:

- What is Azure Policy?
- Create and manage policies to enforce compliance

Resolve reserved resource name errors

5/21/2018 • 2 minutes to read • Edit Online

This article describes the error you encounter when deploying a resource that includes a reserved word in its name.

Symptom

When deploying a resource that is available through a public endpoint, you may receive the following error:

```
Code=ReservedResourceName;
Message=The resource name <resource-name> or a part of the name is a trademarked or reserved word.
```

Cause

Resources that have a public endpoint cannot use reserved words or trademarks in the name.

The following words are reserved:

- ACCESS
- AZURE
- BING
- BIZSPARK
- BIZTALK
- CORTANA
- DIRECTX
- DOTNET
- DYNAMICS
- EXCEL
- EXCHANGE
- FOREFRONT
- GROOVE
- HOLOLENS
- HYPERV
- KINECT
- LYNC
- MSDN
- O365
- OFFICE
- OFFICE365
- ONEDRIVE
- ONENOTE
- OUTLOOK
- POWERPOINT
- SHAREPOINT
- SKYPE

- VISIO
- VISUALSTUDIO

The following words cannot be used as either a whole word or a substring in the name:

- LOGIN
- MICROSOFT
- WINDOWS
- XBOX

Solution

Provide a name that does not use one of the reserved words.

Resolve errors for resource quotas

1/31/2019 • 2 minutes to read • Edit Online

This article describes quota errors you may encounter when deploying resources.

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

Symptom

If you deploy a template that creates resources that exceed your Azure quotas, you get a deployment error that looks like:

```
Code=OperationNotAllowed
Message=Operation results in exceeding quota limits of Core.
Maximum allowed: 4, Current in use: 4, Additional requested: 2.
```

Or, you may see:

```
Code=ResourceQuotaExceeded
Message=Creating the resource of type <resource-type> would exceed the quota of <number>
resources of type <resource-type> per resource group. The current resource count is <number>,
please delete some resources of this type before creating a new one.
```

Cause

Quotas are applied per resource group, subscriptions, accounts, and other scopes. For example, your subscription may be configured to limit the number of cores for a region. If you attempt to deploy a virtual machine with more cores than the permitted amount, you receive an error stating the quota has been exceeded. For complete quota information, see Azure subscription and service limits, quotas, and constraints.

Troubleshooting

Azure CLI

For Azure CLI, use the `az vm list-usage` command to find virtual machine quotas.

```
az vm list-usage --location "South Central US"
```

Which returns:

```
[
  {
    "currentValue": 0,
    "limit": 2000,
    "name": {
      "localizedValue": "Availability Sets",
      "value": "availabilitySets"
    }
  },
  ...
]
```

PowerShell

For PowerShell, use the **Get-AzVMUsage** command to find virtual machine quotas.

```
Get-AzVMUsage -Location "South Central US"
```

Which returns:

```
Name                             Current Value Limit  Unit
----                             ------------- -----  ----
Availability Sets                            0  2000 Count
Total Regional Cores                         0   100 Count
Virtual Machines                             0 10000 Count
```

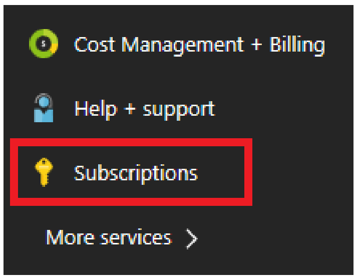
Solution

To request a quota increase, go to the portal and file a support issue. In the support issue, request an increase in your quota for the region into which you want to deploy.

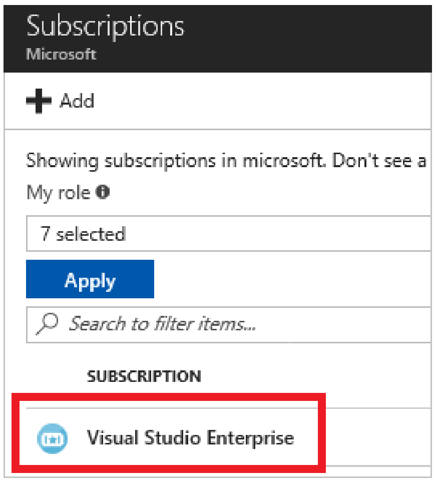
NOTE

Remember that for resource groups, the quota is for each individual region, not for the entire subscription. If you need to deploy 30 cores in West US, you have to ask for 30 Resource Manager cores in West US. If you need to deploy 30 cores in any of the regions to which you have access, you should ask for 30 Resource Manager cores in all regions.

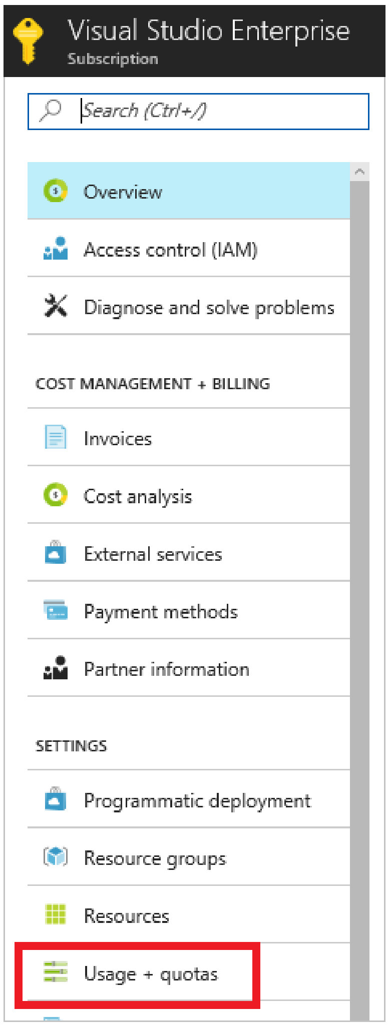
1. Select **Subscriptions**.



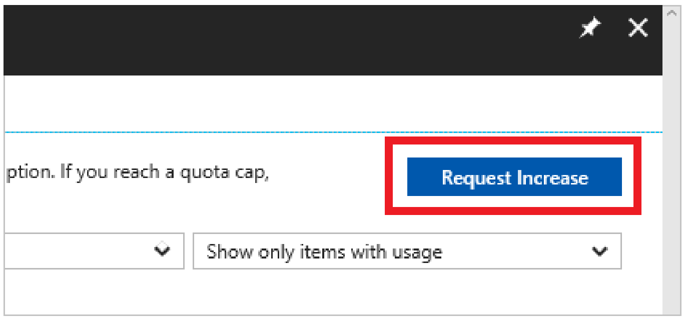
2. Select the subscription that needs an increased quota.


Subscriptions
Microsoft
Add
Showing subscriptions in microsoft. Don't see a
My role
7 selected
Apply
Search to filter items...
SUBSCRIPTION
Visual Studio Enterprise

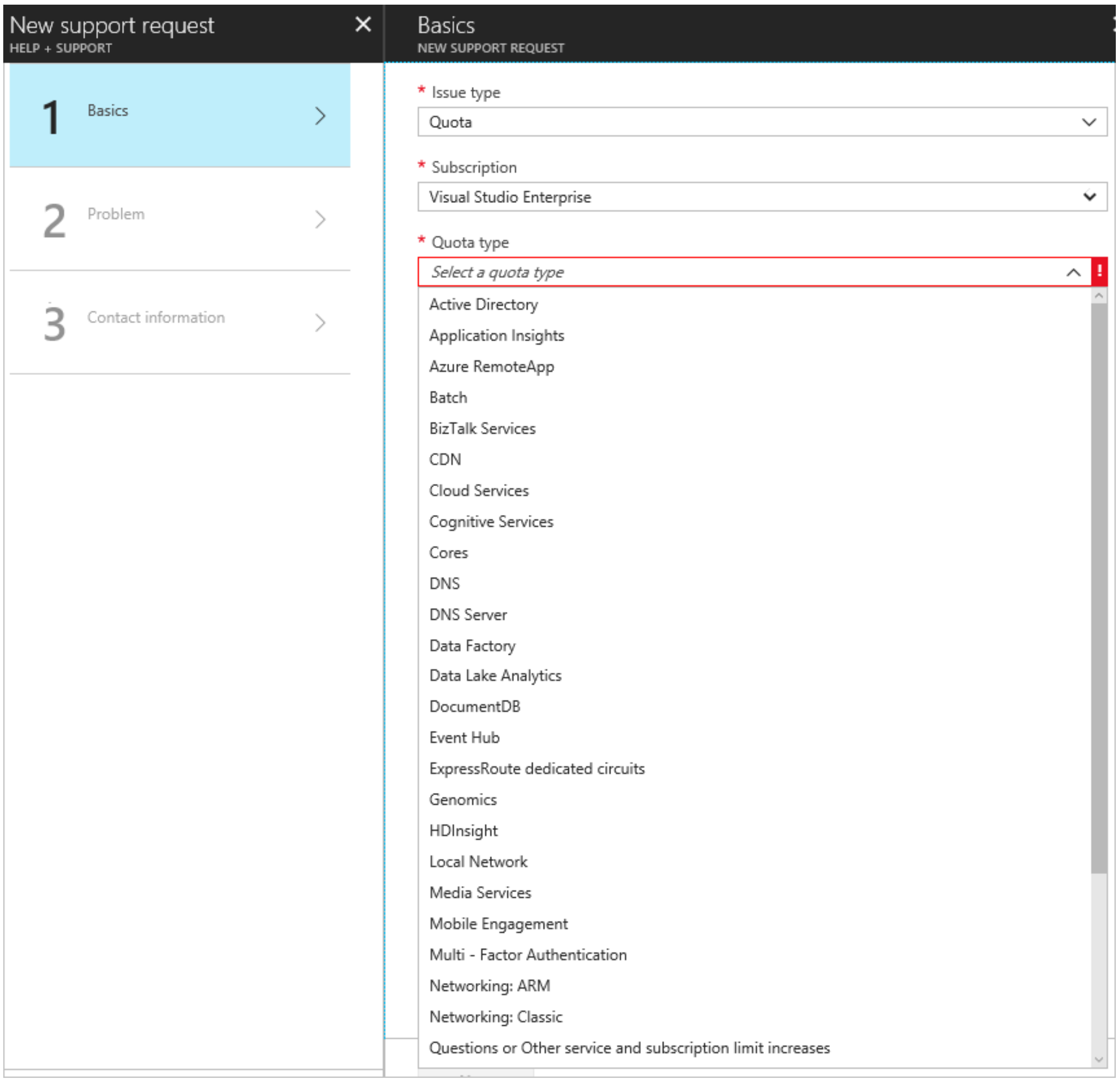
3. Select **Usage + quotas**


Visual Studio Enterprise
Subscription
Search (Ctrl+/)
Overview
Access control (IAM)
Diagnose and solve problems
COST MANAGEMENT + BILLING
Invoices
Cost analysis
External services
Payment methods
Partner information
SETTINGS
Programmatic deployment
Resource groups
Resources
Usage + quotas

4. In the upper right corner, select **Request increase**.


ption. If you reach a quota cap,
Request Increase
Show only items with usage

5. Fill in the forms for the type of quota you need to increase.


New support request
HELP + SUPPORT

1 Basics
2 Problem
3 Contact information

Basics
NEW SUPPORT REQUEST

* Issue type
Quota

* Subscription
Visual Studio Enterprise

* Quota type
Select a quota type

Active Directory
Application Insights
Azure RemoteApp
Batch
BizTalk Services
CDN
Cloud Services
Cognitive Services
Cores
DNS
DNS Server
Data Factory
Data Lake Analytics
DocumentDB
Event Hub
ExpressRoute dedicated circuits
Genomics
HDInsight
Local Network
Media Services
Mobile Engagement
Multi - Factor Authentication
Networking: ARM
Networking: Classic
Questions or Other service and subscription limit increases

Resolve errors for SKU not available

1/31/2019 • 2 minutes to read • Edit Online

This article describes how to resolve the **SkuNotAvailable** error. If you're unable to find a suitable SKU in that region or an alternative region that meets your business needs, submit a SKU request to Azure Support.

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

Symptom

When deploying a resource (typically a virtual machine), you receive the following error code and error message:

```
Code: SkuNotAvailable
Message: The requested tier for resource '<resource>' is currently not available in location '<location>'
for subscription '<subscriptionID>'. Please try another tier or deploy to a different location.
```

Cause

You receive this error when the resource SKU you've selected (such as VM size) isn't available for the location you've selected.

Solution 1 - PowerShell

To determine which SKUs are available in a region, use the Get-AzComputeResourceSku command. Filter the results by location. You must have the latest version of PowerShell for this command.

```
Get-AzComputeResourceSku | where {$_.Locations -icontains "centralus"}
```

The results include a list of SKUs for the location and any restrictions for that SKU. Notice that a SKU might be listed as `NotAvailableForSubscription`.

```
ResourceType          Name        Locations   Restriction                      Capability           Value
------------          ----        ---------   -----------                      ----------           -----
virtualMachines       Standard_A0 centralus   NotAvailableForSubscription      MaxResourceVolumeMB   20480
virtualMachines       Standard_A1 centralus   NotAvailableForSubscription      MaxResourceVolumeMB   71680
virtualMachines       Standard_A2 centralus   NotAvailableForSubscription      MaxResourceVolumeMB  138240
```

Solution 2 - Azure CLI

To determine which SKUs are available in a region, use the `az vm list-skus` command. Use the `--location` parameter to filter output to location you are using. Use the `--size` parameter to search by a partial size name.

```
az vm list-skus --location southcentralus --size Standard_F --output table
```

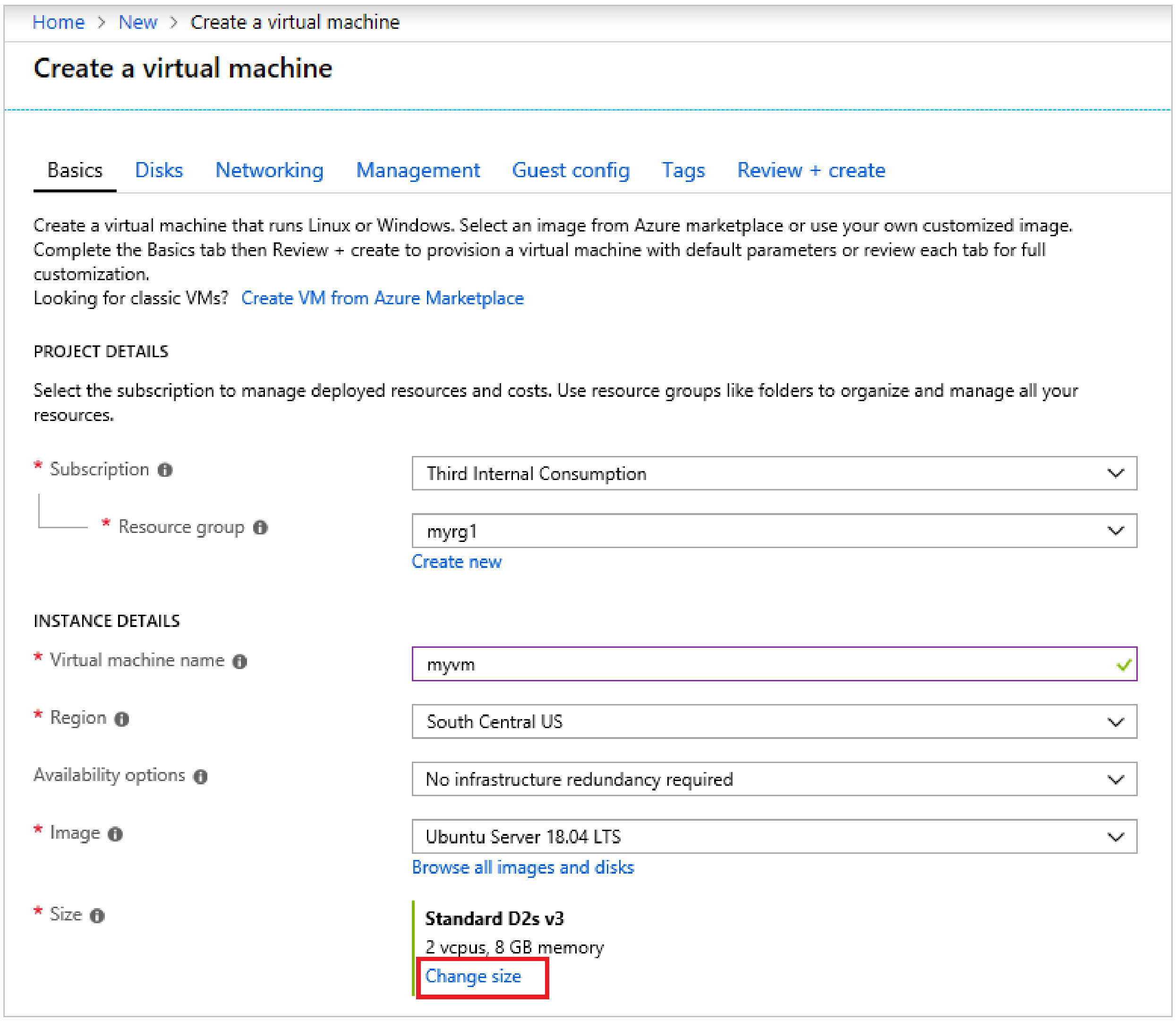
The command returns results like:

```
ResourceType     Locations       Name              Zones    Capabilities    Restrictions
---------------  --------------  ----------------  -------  --------------  --------------
virtualMachines  southcentralus  Standard_F1                ...             None
virtualMachines  southcentralus  Standard_F2                ...             None
virtualMachines  southcentralus  Standard_F4                ...             None
...
```

Solution 3 - Azure portal

To determine which SKUs are available in a region, use the portal. Sign in to the portal, and add a resource through the interface. As you set the values, you see the available SKUs for that resource. You don't need to complete the deployment.

For example, start the process of creating a virtual machine. To see other available size, select **Change size**.



Create a virtual machine

Create a virtual machine that runs Linux or Windows. Select an image from Azure marketplace or use your own customized image. Complete the Basics tab then Review + create to provision a virtual machine with default parameters or review each tab for full customization.
Looking for classic VMs? Create VM from Azure Marketplace

PROJECT DETAILS

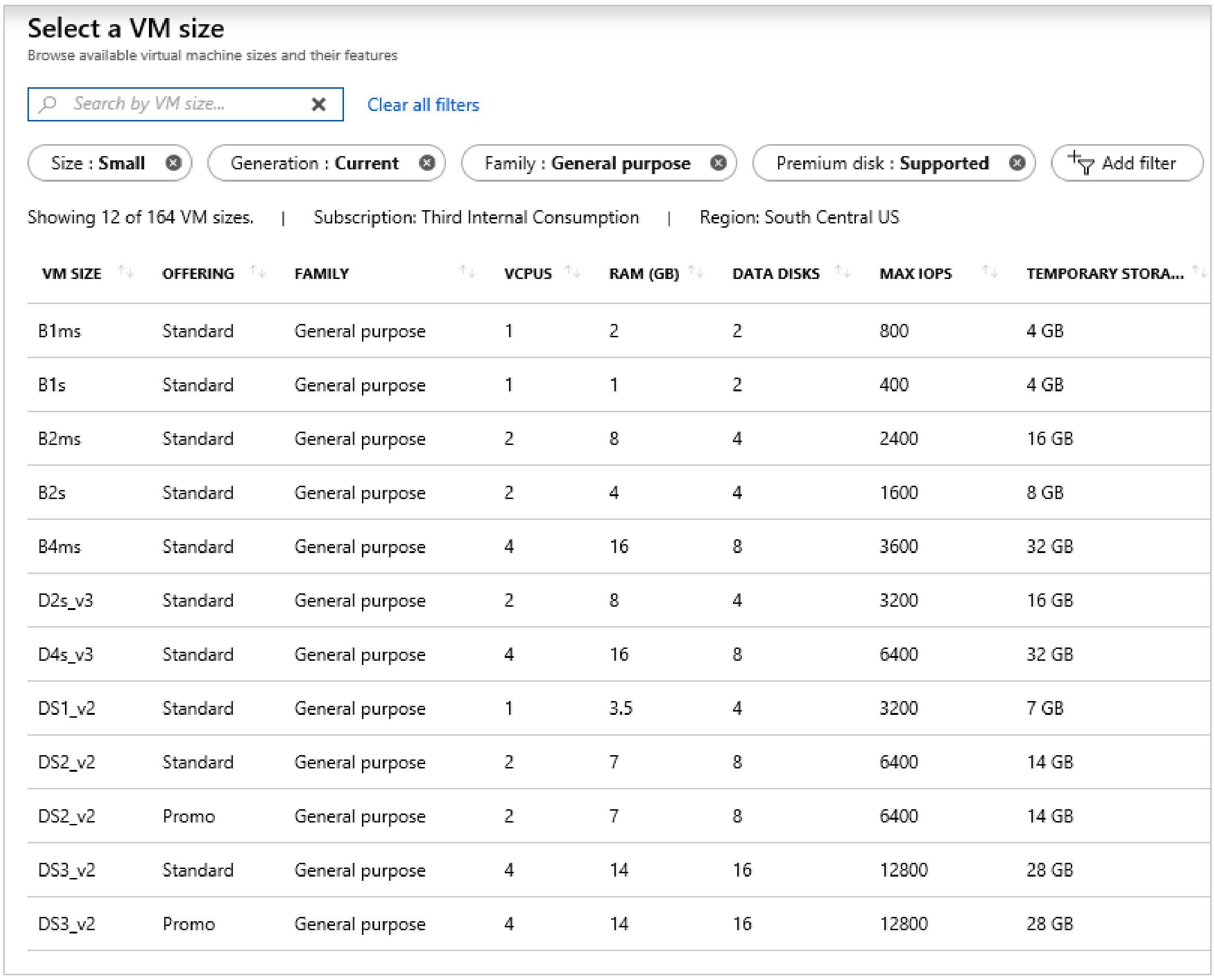
Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

* Subscription	Third Internal Consumption
* Resource group	myrg1 (Create new)

INSTANCE DETAILS

* Virtual machine name	myvm
* Region	South Central US
Availability options	No infrastructure redundancy required
* Image	Ubuntu Server 18.04 LTS (Browse all images and disks)
* Size	**Standard D2s v3** 2 vcpus, 8 GB memory (Change size)

You can filter and scroll through the available sizes.



Select a VM size

Browse available virtual machine sizes and their features

Search by VM size... | Clear all filters

Size : **Small** | Generation : **Current** | Family : **General purpose** | Premium disk : **Supported** | Add filter

Showing 12 of 164 VM sizes. | Subscription: Third Internal Consumption | Region: South Central US

VM SIZE	OFFERING	FAMILY	VCPUS	RAM (GB)	DATA DISKS	MAX IOPS	TEMPORARY STORA...
B1ms	Standard	General purpose	1	2	2	800	4 GB
B1s	Standard	General purpose	1	1	2	400	4 GB
B2ms	Standard	General purpose	2	8	4	2400	16 GB
B2s	Standard	General purpose	2	4	4	1600	8 GB
B4ms	Standard	General purpose	4	16	8	3600	32 GB
D2s_v3	Standard	General purpose	2	8	4	3200	16 GB
D4s_v3	Standard	General purpose	4	16	8	6400	32 GB
DS1_v2	Standard	General purpose	1	3.5	4	3200	7 GB
DS2_v2	Standard	General purpose	2	7	8	6400	14 GB
DS2_v2	Promo	General purpose	2	7	8	6400	14 GB
DS3_v2	Standard	General purpose	4	14	16	12800	28 GB
DS3_v2	Promo	General purpose	4	14	16	12800	28 GB

Solution 4 - REST

To determine which SKUs are available in a region, use the Resource Skus - List operation.

It returns available SKUs and regions in the following format:

```
{
  "value": [
    {
      "resourceType": "virtualMachines",
      "name": "Standard_A0",
      "tier": "Standard",
      "size": "A0",
      "locations": [
        "eastus"
      ],
      "restrictions": []
    },
    {
      "resourceType": "virtualMachines",
      "name": "Standard_A1",
      "tier": "Standard",
      "size": "A1",
      "locations": [
        "eastus"
      ],
      "restrictions": []
    },
    ...
  ]
}
```

2 minutes to read

Azure resource providers and types

4/26/2019 • 5 minutes to read • Edit Online

When deploying resources, you frequently need to retrieve information about the resource providers and types. In this article, you learn how to:

- View all resource providers in Azure
- Check registration status of a resource provider
- Register a resource provider
- View resource types for a resource provider
- View valid locations for a resource type
- View valid API versions for a resource type

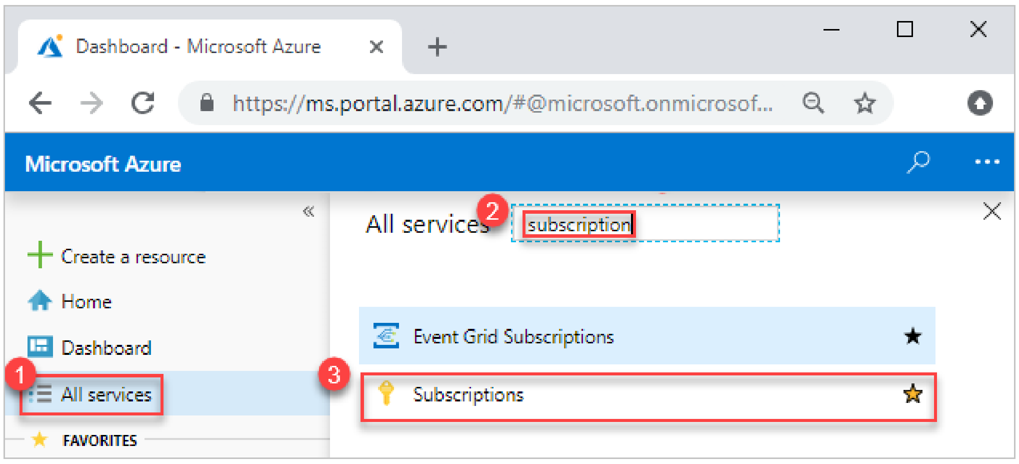
You can do these steps through the Azure portal, Azure PowerShell, or Azure CLI.

For a list that maps resource providers to Azure services, see Resource providers for Azure services.

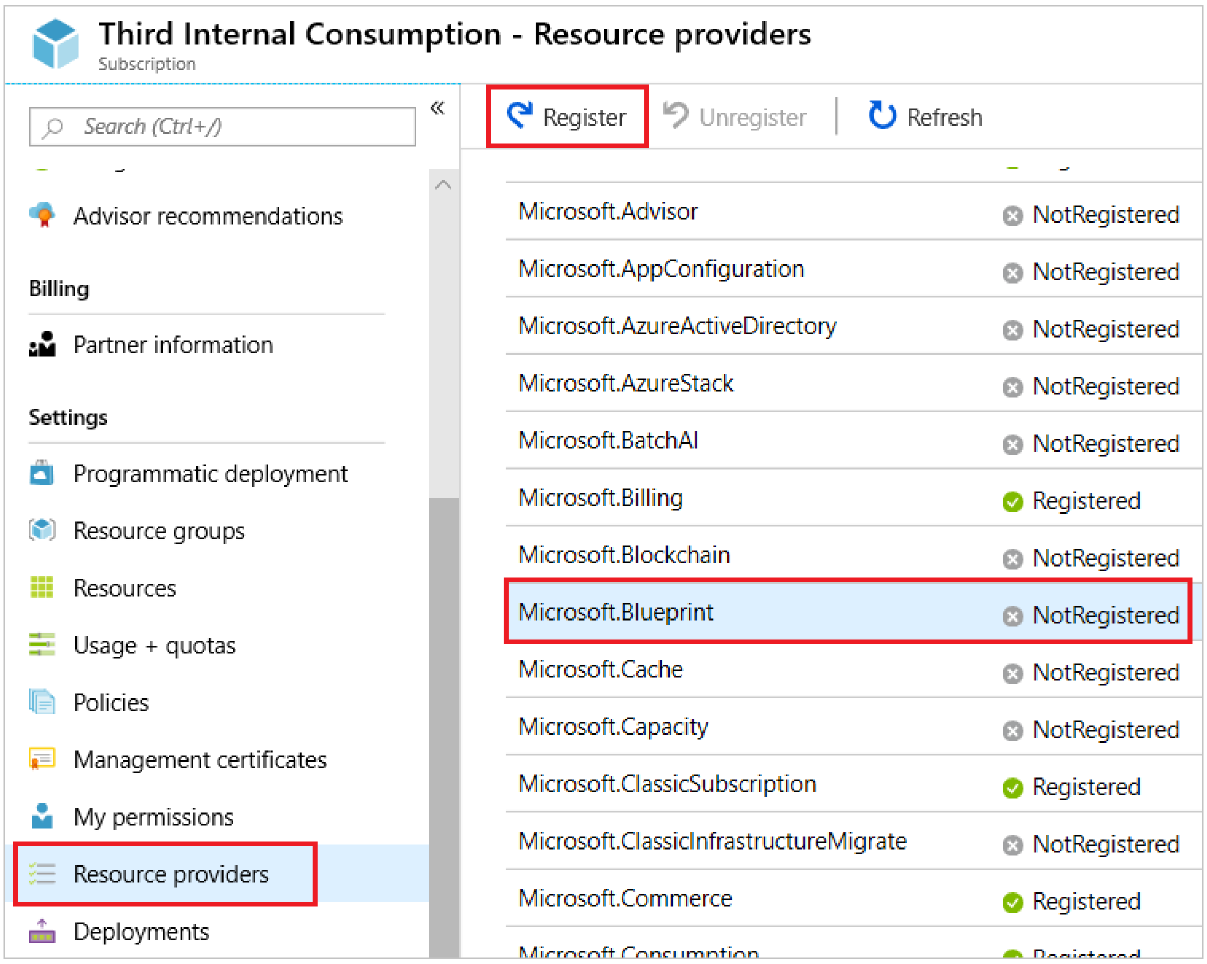
Azure portal

To see all resource providers, and the registration status for your subscription:

1. Sign in to the Azure portal.
2. Select **All services**.



3. In the **All services** box, enter **subscription**, and then select **Subscriptions**.
4. Select the subscription from the subscription list to view.
5. Select **Resource providers** and view the list of available resource providers.



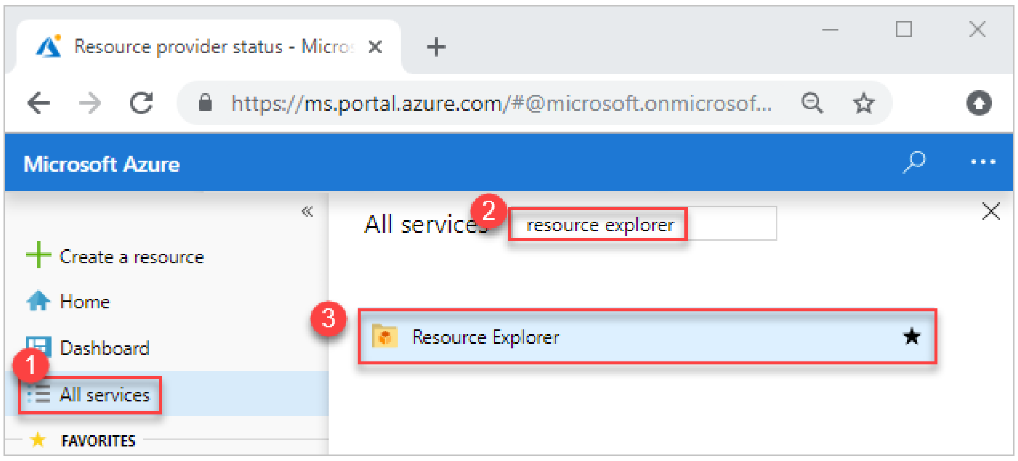
Microsoft.Advisor	NotRegistered
Microsoft.AppConfiguration	NotRegistered
Microsoft.AzureActiveDirectory	NotRegistered
Microsoft.AzureStack	NotRegistered
Microsoft.BatchAI	NotRegistered
Microsoft.Billing	Registered
Microsoft.Blockchain	NotRegistered
Microsoft.Blueprint	NotRegistered
Microsoft.Cache	NotRegistered
Microsoft.Capacity	NotRegistered
Microsoft.ClassicSubscription	Registered
Microsoft.ClassicInfrastructureMigrate	NotRegistered
Microsoft.Commerce	Registered

6. Registering a resource provider configures your subscription to work with the resource provider. The scope for registration is always the subscription. By default, many resource providers are automatically registered. However, you may need to manually register some resource providers. To register a resource provider, you must have permission to do the `/register/action` operation for the resource provider. This operation is included in the Contributor and Owner roles. To register a resource provider, select **Register**. In the previous screenshot, the **Register** link is highlighted for **Microsoft.Blueprint**.

   You can't unregister a resource provider when you still have resource types from that resource provider in your subscription.

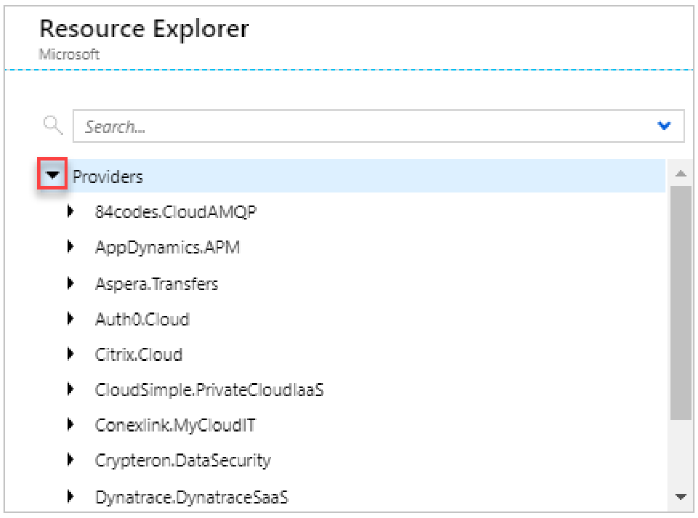
To see information for a particular resource provider:

1. Sign in to the Azure portal.
2. Select **All services**.

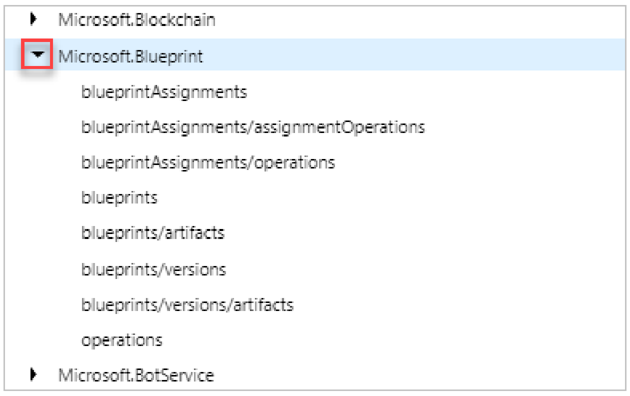


3. In the **All services** box, enter **resource explorer**, and then select **Resource Explorer**.

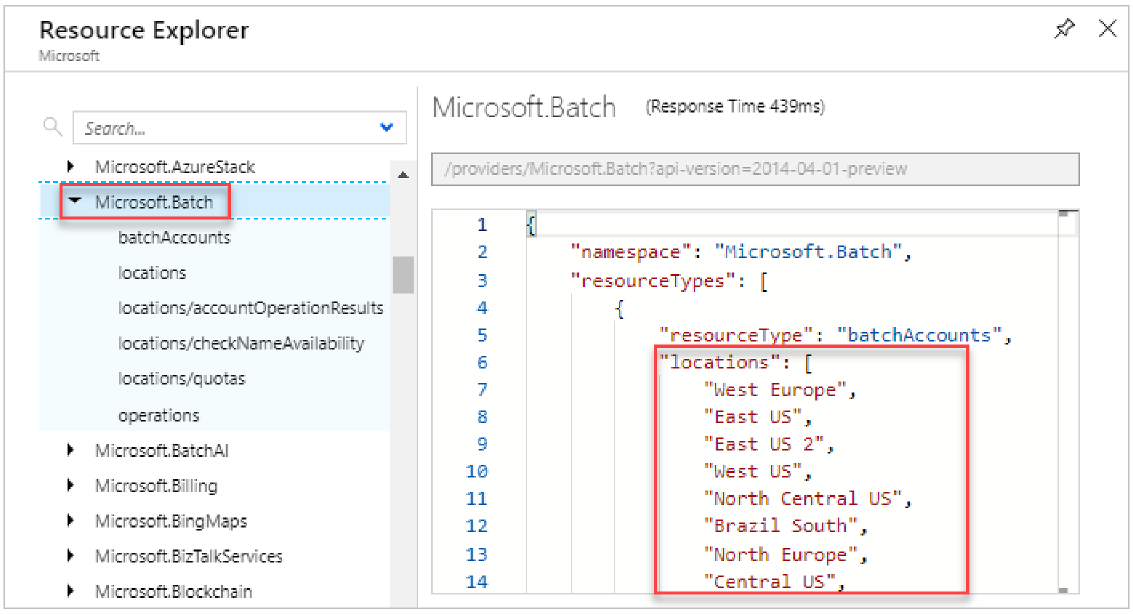
4. Expand **Providers** by selecting the right arrow.



5. Expand a resource provider and resource type that you want to view.

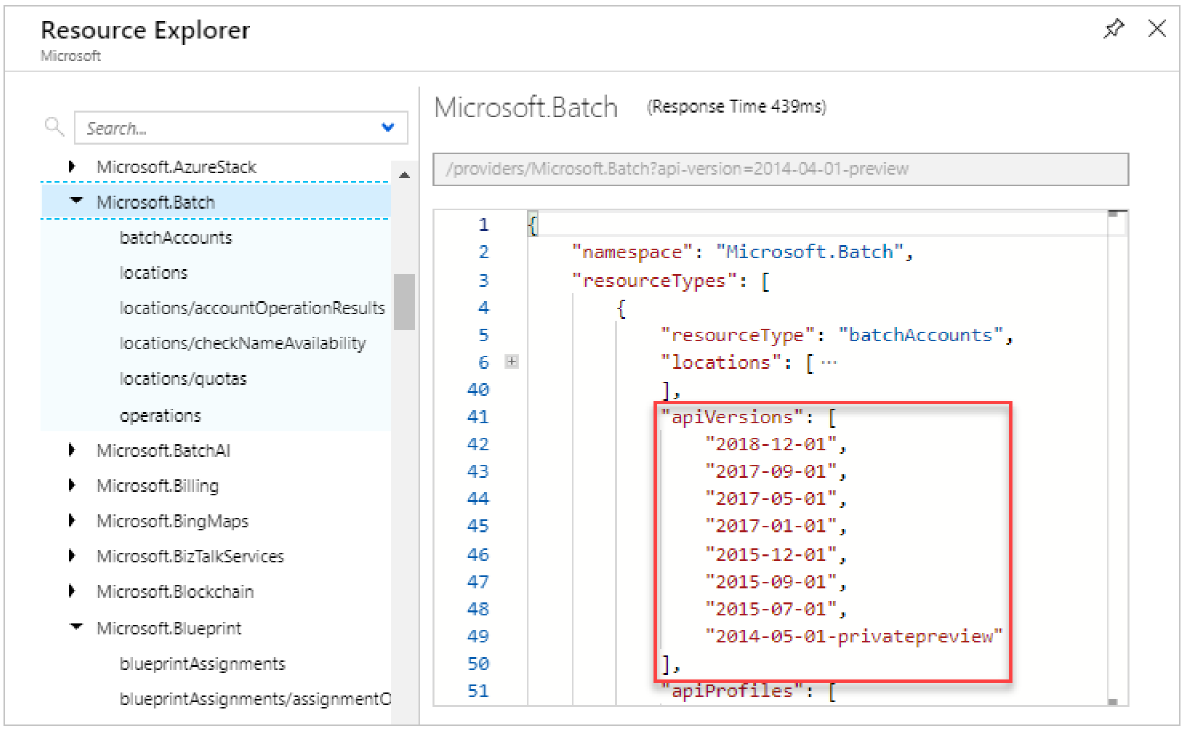


6. Resource Manager is supported in all regions, but the resources you deploy might not be supported in all regions. In addition, there may be limitations on your subscription that prevent you from using some regions that support the resource. The resource explorer displays valid locations for the resource type.



```
{
    "namespace": "Microsoft.Batch",
    "resourceTypes": [
        {
            "resourceType": "batchAccounts",
            "locations": [
                "West Europe",
                "East US",
                "East US 2",
                "West US",
                "North Central US",
                "Brazil South",
                "North Europe",
                "Central US",
```

7. The API version corresponds to a version of REST API operations that are released by the resource provider. As a resource provider enables new features, it releases a new version of the REST API. The resource explorer displays valid API versions for the resource type.



```
{
    "namespace": "Microsoft.Batch",
    "resourceTypes": [
        {
            "resourceType": "batchAccounts",
            "locations": [ ···
            ],
            "apiVersions": [
                "2018-12-01",
                "2017-09-01",
                "2017-05-01",
                "2017-01-01",
                "2015-12-01",
                "2015-09-01",
                "2015-07-01",
                "2014-05-01-privatepreview"
            ],
            "apiProfiles": [
```

Azure PowerShell

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

To see all resource providers in Azure, and the registration status for your subscription, use:

```
Get-AzResourceProvider -ListAvailable | Select-Object ProviderNamespace, RegistrationState
```

Which returns results similar to:

```
ProviderNamespace                RegistrationState
-------------------------------- ------------------
Microsoft.ClassicCompute         Registered
Microsoft.ClassicNetwork         Registered
Microsoft.ClassicStorage         Registered
Microsoft.CognitiveServices      Registered
...
```

Registering a resource provider configures your subscription to work with the resource provider. The scope for registration is always the subscription. By default, many resource providers are automatically registered. However, you may need to manually register some resource providers. To register a resource provider, you must have permission to do the `/register/action` operation for the resource provider. This operation is included in the Contributor and Owner roles.

```
Register-AzResourceProvider -ProviderNamespace Microsoft.Batch
```

Which returns results similar to:

```
ProviderNamespace : Microsoft.Batch
RegistrationState : Registering
ResourceTypes     : {batchAccounts, operations, locations, locations/quotas}
Locations         : {West Europe, East US, East US 2, West US...}
```

You can't unregister a resource provider when you still have resource types from that resource provider in your subscription.

To see information for a particular resource provider, use:

```
Get-AzResourceProvider -ProviderNamespace Microsoft.Batch
```

Which returns results similar to:

```
{ProviderNamespace : Microsoft.Batch
RegistrationState : Registered
ResourceTypes     : {batchAccounts}
Locations         : {West Europe, East US, East US 2, West US...}

...
```

To see the resource types for a resource provider, use:

```
(Get-AzResourceProvider -ProviderNamespace Microsoft.Batch).ResourceTypes.ResourceTypeName
```

Which returns:

```
batchAccounts
operations
locations
locations/quotas
```

The API version corresponds to a version of REST API operations that are released by the resource provider. As a resource provider enables new features, it releases a new version of the REST API.

To get the available API versions for a resource type, use:

```
((Get-AzResourceProvider -ProviderNamespace Microsoft.Batch).ResourceTypes | Where-Object ResourceTypeName -eq
batchAccounts).ApiVersions
```

Which returns:

```
2017-05-01
2017-01-01
2015-12-01
2015-09-01
2015-07-01
```

Resource Manager is supported in all regions, but the resources you deploy might not be supported in all regions. In addition, there may be limitations on your subscription that prevent you from using some regions that support the resource.

To get the supported locations for a resource type, use.

```
((Get-AzResourceProvider -ProviderNamespace Microsoft.Batch).ResourceTypes | Where-Object ResourceTypeName -eq
batchAccounts).Locations
```

Which returns:

```
West Europe
East US
East US 2
West US
...
```

Azure CLI

To see all resource providers in Azure, and the registration status for your subscription, use:

```
az provider list --query "[].{Provider:namespace, Status:registrationState}" --out table
```

Which returns results similar to:

```
Provider                         Status
-------------------------------- ----------------
Microsoft.ClassicCompute         Registered
Microsoft.ClassicNetwork         Registered
Microsoft.ClassicStorage         Registered
Microsoft.CognitiveServices      Registered
...
```

Registering a resource provider configures your subscription to work with the resource provider. The scope for registration is always the subscription. By default, many resource providers are automatically registered. However, you may need to manually register some resource providers. To register a resource provider, you must have permission to do the `/register/action` operation for the resource provider. This operation is included in the Contributor and Owner roles.

```
az provider register --namespace Microsoft.Batch
```

Which returns a message that registration is on-going.

You can't unregister a resource provider when you still have resource types from that resource provider in your subscription.

To see information for a particular resource provider, use:

```
az provider show --namespace Microsoft.Batch
```

Which returns results similar to:

```
{
    "id": "/subscriptions/####-####/providers/Microsoft.Batch",
    "namespace": "Microsoft.Batch",
    "registrationsState": "Registering",
    "resourceTypes:" [
        ...
    ]
}
```

To see the resource types for a resource provider, use:

```
az provider show --namespace Microsoft.Batch --query "resourceTypes[*].resourceType" --out table
```

Which returns:

```
Result
---------------
batchAccounts
operations
locations
locations/quotas
```

The API version corresponds to a version of REST API operations that are released by the resource provider. As a resource provider enables new features, it releases a new version of the REST API.

To get the available API versions for a resource type, use:

```
az provider show --namespace Microsoft.Batch --query "resourceTypes[?resourceType=='batchAccounts'].apiVersions
| [0]" --out table
```

Which returns:

```
Result
---------------
2017-05-01
2017-01-01
2015-12-01
2015-09-01
2015-07-01
```

Resource Manager is supported in all regions, but the resources you deploy might not be supported in all regions. In addition, there may be limitations on your subscription that prevent you from using some regions that support the resource.

To get the supported locations for a resource type, use.

```
az provider show --namespace Microsoft.Batch --query "resourceTypes[?resourceType=='batchAccounts'].locations |
[0]" --out table
```

Which returns:

```
Result
---------------
West Europe
East US
East US 2
West US
...
```

Next steps

- To learn about creating Resource Manager templates, see Authoring Azure Resource Manager templates.
- To view the resource provider template schemas, see Template reference.
- For a list that maps resource providers to Azure services, see Resource providers for Azure services.
- To view the operations for a resource provider, see Azure REST API.

Throttling Resource Manager requests

7/17/2019 • 4 minutes to read • Edit Online

For each Azure subscription and tenant, Resource Manager allows up to 12,000 read requests per hour and 1,200 write requests per hour. These limits are scoped to the security principal (user or application) making the requests and the subscription ID or tenant ID. If your requests come from more than one security principal, your limit across the subscription or tenant is greater than 12,000 and 1,200 per hour.

Requests are applied to either your subscription or your tenant. Subscription requests are ones that involve passing your subscription ID, such as retrieving the resource groups in your subscription. Tenant requests don't include your subscription ID, such as retrieving valid Azure locations.

These limits apply to each Azure Resource Manager instance. There are multiple instances in every Azure region, and Azure Resource Manager is deployed to all Azure regions. So, in practice, limits are effectively much higher than these limits, as user requests are usually serviced by many different instances.

If your application or script reaches these limits, you need to throttle your requests. This article shows you how to determine the remaining requests you have before reaching the limit, and how to respond when you've reached the limit.

When you reach the limit, you receive the HTTP status code **429 Too many requests**.

Azure Resource Graph limits the number of requests to its operations. The steps in this article to determine the remaining requests and how to respond when the limit is reached also apply to Resource Graph. However, Resource Graph sets its own limit and reset rate. For more information, see Throttle in Azure Resource Graph.

Remaining requests

You can determine the number of remaining requests by examining response headers. Read requests return a value in the header for the number of remaining read requests. Write requests include a value for the number of remaining write requests. The following table describes the response headers you can examine for those values:

RESPONSE HEADER	DESCRIPTION
x-ms-ratelimit-remaining-subscription-reads	Subscription scoped reads remaining. This value is returned on read operations.
x-ms-ratelimit-remaining-subscription-writes	Subscription scoped writes remaining. This value is returned on write operations.
x-ms-ratelimit-remaining-tenant-reads	Tenant scoped reads remaining
x-ms-ratelimit-remaining-tenant-writes	Tenant scoped writes remaining
x-ms-ratelimit-remaining-subscription-resource-requests	Subscription scoped resource type requests remaining. This header value is only returned if a service has overridden the default limit. Resource Manager adds this value instead of the subscription reads or writes.

RESPONSE HEADER	DESCRIPTION
x-ms-ratelimit-remaining-subscription-resource-entities-read	Subscription scoped resource type collection requests remaining. This header value is only returned if a service has overridden the default limit. This value provides the number of remaining collection requests (list resources).
x-ms-ratelimit-remaining-tenant-resource-requests	Tenant scoped resource type requests remaining. This header is only added for requests at tenant level, and only if a service has overridden the default limit. Resource Manager adds this value instead of the tenant reads or writes.
x-ms-ratelimit-remaining-tenant-resource-entities-read	Tenant scoped resource type collection requests remaining. This header is only added for requests at tenant level, and only if a service has overridden the default limit.

Retrieving the header values

Retrieving these header values in your code or script is no different than retrieving any header value.

For example, in **C#**, you retrieve the header value from an **HttpWebResponse** object named **response** with the following code:

```
response.Headers.GetValues("x-ms-ratelimit-remaining-subscription-reads").GetValue(0)
```

In **PowerShell**, you retrieve the header value from an Invoke-WebRequest operation.

```
$r = Invoke-WebRequest -Uri https://management.azure.com/subscriptions/{guid}/resourcegroups?api-version=2016-
09-01 -Method GET -Headers $authHeaders
$r.Headers["x-ms-ratelimit-remaining-subscription-reads"]
```

For a complete PowerShell example, see Check Resource Manager Limits for a Subscription.

If you want to see the remaining requests for debugging, you can provide the **-Debug** parameter on your **PowerShell** cmdlet.

```
Get-AzResourceGroup -Debug
```

Which returns many values, including the following response value:

```
DEBUG: ============================ HTTP RESPONSE ============================

Status Code:
OK

Headers:
Pragma                        : no-cache
x-ms-ratelimit-remaining-subscription-reads: 11999
```

To get write limits, use a write operation:

```
New-AzResourceGroup -Name myresourcegroup -Location westus -Debug
```

Which returns many values, including the following values:

```
DEBUG: ============================ HTTP RESPONSE ============================

Status Code:
Created

Headers:
Pragma                        : no-cache
x-ms-ratelimit-remaining-subscription-writes: 1199
```

In **Azure CLI**, you retrieve the header value by using the more verbose option.

```
az group list --verbose --debug
```

Which returns many values, including the following values:

```
msrest.http_logger : Response status: 200
msrest.http_logger : Response headers:
msrest.http_logger :     'Cache-Control': 'no-cache'
msrest.http_logger :     'Pragma': 'no-cache'
msrest.http_logger :     'Content-Type': 'application/json; charset=utf-8'
msrest.http_logger :     'Content-Encoding': 'gzip'
msrest.http_logger :     'Expires': '-1'
msrest.http_logger :     'Vary': 'Accept-Encoding'
msrest.http_logger :     'x-ms-ratelimit-remaining-subscription-reads': '11998'
```

To get write limits, use a write operation:

```
az group create -n myresourcegroup --location westus --verbose --debug
```

Which returns many values, including the following values:

```
msrest.http_logger : Response status: 201
msrest.http_logger : Response headers:
msrest.http_logger :     'Cache-Control': 'no-cache'
msrest.http_logger :     'Pragma': 'no-cache'
msrest.http_logger :     'Content-Length': '163'
msrest.http_logger :     'Content-Type': 'application/json; charset=utf-8'
msrest.http_logger :     'Expires': '-1'
msrest.http_logger :     'x-ms-ratelimit-remaining-subscription-writes': '1199'
```

Waiting before sending next request

When you reach the request limit, Resource Manager returns the **429** HTTP status code and a **Retry-After** value in the header. The **Retry-After** value specifies the number of seconds your application should wait (or sleep) before sending the next request. If you send a request before the retry value has elapsed, your request isn't processed and a new retry value is returned.

Next steps

- For a complete PowerShell example, see Check Resource Manager Limits for a Subscription.

- For more information about limits and quotas, see Azure subscription and service limits, quotas, and constraints.
- To learn about handling asynchronous REST requests, see Track asynchronous Azure operations.

Track asynchronous Azure operations

6/18/2019 • 3 minutes to read • Edit Online

Some Azure REST operations run asynchronously because the operation can't be completed quickly. This article describes how to track the status of asynchronous operations through values returned in the response.

Status codes for asynchronous operations

An asynchronous operation initially returns an HTTP status code of either:

- 201 (Created)
- 202 (Accepted)

When the operation successfully completes, it returns either:

- 200 (OK)
- 204 (No Content)

Refer to the REST API documentation to see the responses for the operation you're executing.

Monitor status of operation

The asynchronous REST operations return header values, which you use to determine the status of the operation. There are potentially three header values to examine:

- `Azure-AsyncOperation` - URL for checking the ongoing status of the operation. If your operation returns this value, always use it (instead of Location) to track the status of the operation.
- `Location` - URL for determining when an operation has completed. Use this value only when Azure-AsyncOperation isn't returned.
- `Retry-After` - The number of seconds to wait before checking the status of the asynchronous operation.

However, not every asynchronous operation returns all these values. For example, you may need to evaluate the Azure-AsyncOperation header value for one operation, and the Location header value for another operation.

You retrieve the header values as you would retrieve any header value for a request. For example, in C#, you retrieve the header value from an `HttpWebResponse` object named `response` with the following code:

```
response.Headers.GetValues("Azure-AsyncOperation").GetValue(0)
```

Azure-AsyncOperation request and response

To get the status of the asynchronous operation, send a GET request to the URL in Azure-AsyncOperation header value.

The body of the response from this operation contains information about the operation. The following example shows the possible values returned from the operation:

```
{
    "id": "{resource path from GET operation}",
    "name": "{operation-id}",
    "status" : "Succeeded | Failed | Canceled | {resource provider values}",
    "startTime": "2017-01-06T20:56:36.002812+00:00",
    "endTime": "2017-01-06T20:56:56.002812+00:00",
    "percentComplete": {double between 0 and 100 },
    "properties": {
        /* Specific resource provider values for successful operations */
    },
    "error" : {
        "code": "{error code}",
        "message": "{error description}"
    }
}
```

Only `status` is returned for all responses. The error object is returned when the status is Failed or Canceled. All other values are optional; therefore, the response you receive may look different than the example.

provisioningState values

Operations that create, update, or delete (PUT, PATCH, DELETE) a resource typically return a `provisioningState` value. When an operation has completed, one of following three values is returned:

- Succeeded
- Failed
- Canceled

All other values indicate the operation is still running. The resource provider can return a customized value that indicates its state. For example, you may receive **Accepted** when the request is received and running.

Example requests and responses

Start virtual machine (202 with Azure-AsyncOperation)

This example shows how to determine the status of **start** operation for virtual machines. The initial request is in the following format:

```
POST
https://management.azure.com/subscriptions/{subscription-id}/resourceGroups/{resource-
group}/providers/Microsoft.Compute/virtualMachines/{vm-name}/start?api-version=2016-03-30
```

It returns status code 202. Among the header values, you see:

```
Azure-AsyncOperation : https://management.azure.com/subscriptions/{subscription-
id}/providers/Microsoft.Compute/locations/{region}/operations/{operation-id}?api-version=2016-03-30
```

To check the status of the asynchronous operation, sending another request to that URL.

```
GET
https://management.azure.com/subscriptions/{subscription-
id}/providers/Microsoft.Compute/locations/{region}/operations/{operation-id}?api-version=2016-03-30
```

The response body contains the status of the operation:

```
{
  "startTime": "2017-01-06T18:58:24.7596323+00:00",
  "status": "InProgress",
  "name": "9a062a88-e463-4697-bef2-fe039df73a02"
}
```

Deploy resources (201 with Azure-AsyncOperation)

This example shows how to determine the status of **deployments** operation for deploying resources to Azure. The initial request is in the following format:

```
PUT
https://management.azure.com/subscriptions/{subscription-id}/resourcegroups/{resource-group}/providers/microsoft.resources/deployments/{deployment-name}?api-version=2016-09-01
```

It returns status code 201. The body of the response includes:

```
"provisioningState":"Accepted",
```

Among the header values, you see:

```
Azure-AsyncOperation: https://management.azure.com/subscriptions/{subscription-id}/resourcegroups/{resource-group}/providers/Microsoft.Resources/deployments/{deployment-name}/operationStatuses/{operation-id}?api-version=2016-09-01
```

To check the status of the asynchronous operation, sending another request to that URL.

```
GET
https://management.azure.com/subscriptions/{subscription-id}/resourcegroups/{resource-group}/providers/Microsoft.Resources/deployments/{deployment-name}/operationStatuses/{operation-id}?api-version=2016-09-01
```

The response body contains the status of the operation:

```
{"status":"Running"}
```

When the deployment is finished, the response contains:

```
{"status":"Succeeded"}
```

Create storage account (202 with Location and Retry-After)

This example shows how to determine the status of the **create** operation for storage accounts. The initial request is in the following format:

```
PUT
https://management.azure.com/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.Storage/storageAccounts/{storage-name}?api-version=2016-01-01
```

And the request body contains properties for the storage account:

```
{ "location": "South Central US", "properties": {}, "sku": { "name": "Standard_LRS" }, "kind": "Storage" }
```

It returns status code 202. Among the header values, you see the following two values:

```
Location: https://management.azure.com/subscriptions/{subscription-
id}/providers/Microsoft.Storage/operations/{operation-id}?monitor=true&api-version=2016-01-01
Retry-After: 17
```

After waiting for number of seconds specified in Retry-After, check the status of the asynchronous operation by sending another request to that URL.

```
GET
https://management.azure.com/subscriptions/{subscription-
id}/providers/Microsoft.Storage/operations/{operation-id}?monitor=true&api-version=2016-01-01
```

If the request is still running, you receive a status code 202. If the request has completed, your receive a status code 200, and the body of the response contains the properties of the storage account that has been created.

Next steps

- For documentation about each REST operation, see REST API documentation.
- for information about deploying templates through the Resource Manager REST API, see Deploy resources with Resource Manager templates and Resource Manager REST API.

Azure Resource Manager template functions

6/18/2019 • 2 minutes to read • Edit Online

This article describes all the functions you can use in an Azure Resource Manager template. For information about using functions in your template, see template syntax.

To create your own functions, see User-defined functions.

Array and object functions

Resource Manager provides several functions for working with arrays and objects.

- array
- coalesce
- concat
- contains
- createArray
- empty
- first
- intersection
- json
- last
- length
- min
- max
- range
- skip
- take
- union

Comparison functions

Resource Manager provides several functions for making comparisons in your templates.

- equals
- less
- lessOrEquals
- greater
- greaterOrEquals

Deployment value functions

Resource Manager provides the following functions for getting values from sections of the template and values related to the deployment:

- deployment
- parameters
- variables

Logical functions

Resource Manager provides the following functions for working with logical conditions:

- and
- bool
- if
- not
- or

Numeric functions

Resource Manager provides the following functions for working with integers:

- add
- copyIndex
- div
- float
- int
- min
- max
- mod
- mul
- sub

Resource functions

Resource Manager provides the following functions for getting resource values:

- listAccountSas
- listKeys
- listSecrets
- list*
- providers
- reference
- resourceGroup
- resourceId
- subscription

String functions

Resource Manager provides the following functions for working with strings:

- base64
- base64ToJson
- base64ToString
- concat
- contains
- dataUri
- dataUriToString
- empty

- endsWith
- first
- format
- guid
- indexOf
- last
- lastIndexOf
- length
- newGuid
- padLeft
- replace
- skip
- split
- startsWith
- string
- substring
- take
- toLower
- toUpper
- trim
- uniqueString
- uri
- uriComponent
- uriComponentToString
- utcNow

Next steps

- For a description of the sections in an Azure Resource Manager template, see Authoring Azure Resource Manager templates
- To merge multiple templates, see Using linked templates with Azure Resource Manager
- To iterate a specified number of times when creating a type of resource, see Create multiple instances of resources in Azure Resource Manager
- To see how to deploy the template you've created, see Deploy an application with Azure Resource Manager template

Array and object functions for Azure Resource Manager templates

6/18/2019 • 17 minutes to read • Edit Online

Resource Manager provides several functions for working with arrays and objects.

- array
- coalesce
- concat
- contains
- createArray
- empty
- first
- intersection
- json
- last
- length
- max
- min
- range
- skip
- take
- union

To get an array of string values delimited by a value, see split.

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

array

`array(convertToArray)`

Converts the value to an array.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
convertToArray	Yes	int, string, array, or object	The value to convert to an array.

Return value

An array.

Example

The following example template shows how to use the array function with different types.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "intToConvert": {
            "type": "int",
            "defaultValue": 1
        },
        "stringToConvert": {
            "type": "string",
            "defaultValue": "efgh"
        },
        "objectToConvert": {
            "type": "object",
            "defaultValue": {"a": "b", "c": "d"}
        }
    },
    "resources": [
    ],
    "outputs": {
        "intOutput": {
            "type": "array",
            "value": "[array(parameters('intToConvert'))]"
        },
        "stringOutput": {
            "type": "array",
            "value": "[array(parameters('stringToConvert'))]"
        },
        "objectOutput": {
            "type": "array",
            "value": "[array(parameters('objectToConvert'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
intOutput	Array	[1]
stringOutput	Array	["efgh"]
objectOutput	Array	[{"a": "b", "c": "d"}]

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/array.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/array.json
```

coalesce

```
coalesce(arg1, arg2, arg3, ...)
```

Returns first non-null value from the parameters. Empty strings, empty arrays, and empty objects are not null.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	int, string, array, or object	The first value to test for null.
additional args	No	int, string, array, or object	Additional values to test for null.

Return value

The value of the first non-null parameters, which can be a string, int, array, or object. Null if all parameters are null.

Example

The following example template shows the output from different uses of coalesce.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "objectToTest": {
            "type": "object",
            "defaultValue": {
                "null1": null,
                "null2": null,
                "string": "default",
                "int": 1,
                "object": {"first": "default"},
                "array": [1]
            }
        }
    },
    "resources": [
    ],
    "outputs": {
        "stringOutput": {
            "type": "string",
            "value": "[coalesce(parameters('objectToTest').null1, parameters('objectToTest').null2,
parameters('objectToTest').string)]"
        },
        "intOutput": {
            "type": "int",
            "value": "[coalesce(parameters('objectToTest').null1, parameters('objectToTest').null2,
parameters('objectToTest').int)]"
        },
        "objectOutput": {
            "type": "object",
            "value": "[coalesce(parameters('objectToTest').null1, parameters('objectToTest').null2,
parameters('objectToTest').object)]"
        },
        "arrayOutput": {
            "type": "array",
            "value": "[coalesce(parameters('objectToTest').null1, parameters('objectToTest').null2,
parameters('objectToTest').array)]"
        },
        "emptyOutput": {
            "type": "bool",
            "value": "[empty(coalesce(parameters('objectToTest').null1, parameters('objectToTest').null2))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
stringOutput	String	default
intOutput	Int	1
objectOutput	Object	{"first": "default"}
arrayOutput	Array	[1]
emptyOutput	Bool	True

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/coalesce.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/coalesce.json
```

concat

`concat(arg1, arg2, arg3, ...)`

Combines multiple arrays and returns the concatenated array, or combines multiple string values and returns the concatenated string.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	array or string	The first array or string for concatenation.
additional arguments	No	array or string	Additional arrays or strings in sequential order for concatenation.

This function can take any number of arguments, and can accept either strings or arrays for the parameters.

Return value

A string or array of concatenated values.

Example

The following example template shows how to combine two arrays.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "firstArray": {
            "type": "array",
            "defaultValue": [
                "1-1",
                "1-2",
                "1-3"
            ]
        },
        "secondArray": {
            "type": "array",
            "defaultValue": [
                "2-1",
                "2-2",
                "2-3"
            ]
        }
    },
    "resources": [
    ],
    "outputs": {
        "return": {
            "type": "array",
            "value": "[concat(parameters('firstArray'), parameters('secondArray'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
return	Array	["1-1", "1-2", "1-3", "2-1", "2-2", "2-3"]

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-manager/functions/concat-
array.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-manager/functions/concat-
array.json
```

The following example template shows how to combine two string values and return a concatenated string.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "prefix": {
            "type": "string",
            "defaultValue": "prefix"
        }
    },
    "resources": [],
    "outputs": {
        "concatOutput": {
            "value": "[concat(parameters('prefix'), '-', uniqueString(resourceGroup().id))]",
            "type" : "string"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
concatOutput	String	prefix-5yj4yjf5mbg72

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-manager/functions/concat-
string.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-manager/functions/concat-
string.json
```

contains

`contains(container, itemToFind)`

Checks whether an array contains a value, an object contains a key, or a string contains a substring. The string comparison is case-sensitive. However, when testing if an object contains a key, the comparison is case-insensitive.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
container	Yes	array, object, or string	The value that contains the value to find.
itemToFind	Yes	string or int	The value to find.

Return value

True if the item is found; otherwise, **False**.

Example

The following example template shows how to use contains with different types:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "stringToTest": {
            "type": "string",
            "defaultValue": "OneTwoThree"
        },
        "objectToTest": {
            "type": "object",
            "defaultValue": {"one": "a", "two": "b", "three": "c"}
        },
        "arrayToTest": {
            "type": "array",
            "defaultValue": ["one", "two", "three"]
        }
    },
    "resources": [
    ],
    "outputs": {
        "stringTrue": {
            "type": "bool",
            "value": "[contains(parameters('stringToTest'), 'e')]"
        },
        "stringFalse": {
            "type": "bool",
            "value": "[contains(parameters('stringToTest'), 'z')]"
        },
        "objectTrue": {
            "type": "bool",
            "value": "[contains(parameters('objectToTest'), 'one')]"
        },
        "objectFalse": {
            "type": "bool",
            "value": "[contains(parameters('objectToTest'), 'a')]"
        },
        "arrayTrue": {
            "type": "bool",
            "value": "[contains(parameters('arrayToTest'), 'three')]"
        },
        "arrayFalse": {
            "type": "bool",
            "value": "[contains(parameters('arrayToTest'), 'four')]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
stringTrue	Bool	True
stringFalse	Bool	False
objectTrue	Bool	True
objectFalse	Bool	False
arrayTrue	Bool	True

NAME	TYPE	VALUE
arrayFalse	Bool	False

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/contains.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/contains.json
```

createarray

`createArray (arg1, arg2, arg3, ...)`

Creates an array from the parameters.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	String, Integer, Array, or Object	The first value in the array.
additional arguments	No	String, Integer, Array, or Object	Additional values in the array.

Return value

An array.

Example

The following example template shows how to use createArray with different types:

```json
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "objectToTest": {
            "type": "object",
            "defaultValue": {"one": "a", "two": "b", "three": "c"}
        },
        "arrayToTest": {
            "type": "array",
            "defaultValue": ["one", "two", "three"]
        }
    },
    "resources": [
    ],
    "outputs": {
        "stringArray": {
            "type": "array",
            "value": "[createArray('a', 'b', 'c')]"
        },
        "intArray": {
            "type": "array",
            "value": "[createArray(1, 2, 3)]"
        },
        "objectArray": {
            "type": "array",
            "value": "[createArray(parameters('objectToTest'))]"
        },
        "arrayArray": {
            "type": "array",
            "value": "[createArray(parameters('arrayToTest'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
stringArray	Array	["a", "b", "c"]
intArray	Array	[1, 2, 3]
objectArray	Array	[{"one": "a", "two": "b", "three": "c"}]
arrayArray	Array	[["one", "two", "three"]]

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/createarray.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/createarray.json
```

empty

`empty(itemToTest)`

Determines if an array, object, or string is empty.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
itemToTest	Yes	array, object, or string	The value to check if it is empty.

Return value

Returns **True** if the value is empty; otherwise, **False**.

Example

The following example template checks whether an array, object, and string are empty.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "testArray": {
            "type": "array",
            "defaultValue": []
        },
        "testObject": {
            "type": "object",
            "defaultValue": {}
        },
        "testString": {
            "type": "string",
            "defaultValue": ""
        }
    },
    "resources": [
    ],
    "outputs": {
        "arrayEmpty": {
            "type": "bool",
            "value": "[empty(parameters('testArray'))]"
        },
        "objectEmpty": {
            "type": "bool",
            "value": "[empty(parameters('testObject'))]"
        },
        "stringEmpty": {
            "type": "bool",
            "value": "[empty(parameters('testString'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayEmpty	Bool	True
objectEmpty	Bool	True

NAME	TYPE	VALUE
stringEmpty	Bool	True

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/empty.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/empty.json
```

first

`first(arg1)`

Returns the first element of the array, or first character of the string.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	array or string	The value to retrieve the first element or character.

Return value

The type (string, int, array, or object) of the first element in an array, or the first character of a string.

Example

The following example template shows how to use the first function with an array and string.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "arrayToTest": {
            "type": "array",
            "defaultValue": ["one", "two", "three"]
        }
    },
    "resources": [
    ],
    "outputs": {
        "arrayOutput": {
            "type": "string",
            "value": "[first(parameters('arrayToTest'))]"
        },
        "stringOutput": {
            "type": "string",
            "value": "[first('One Two Three')]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayOutput	String	one
stringOutput	String	O

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/first.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/first.json
```

intersection

`intersection(arg1, arg2, arg3, ...)`

Returns a single array or object with the common elements from the parameters.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	array or object	The first value to use for finding common elements.
arg2	Yes	array or object	The second value to use for finding common elements.
additional arguments	No	array or object	Additional values to use for finding common elements.

Return value

An array or object with the common elements.

Example

The following example template shows how to use intersection with arrays and objects:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "firstObject": {
            "type": "object",
            "defaultValue": {"one": "a", "two": "b", "three": "c"}
        },
        "secondObject": {
            "type": "object",
            "defaultValue": {"one": "a", "two": "z", "three": "c"}
        },
        "firstArray": {
            "type": "array",
            "defaultValue": ["one", "two", "three"]
        },
        "secondArray": {
            "type": "array",
            "defaultValue": ["two", "three"]
        }
    },
    "resources": [
    ],
    "outputs": {
        "objectOutput": {
            "type": "object",
            "value": "[intersection(parameters('firstObject'), parameters('secondObject'))]"
        },
        "arrayOutput": {
            "type": "array",
            "value": "[intersection(parameters('firstArray'), parameters('secondArray'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
objectOutput	Object	{"one": "a", "three": "c"}
arrayOutput	Array	["two", "three"]

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/intersection.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/intersection.json
```

json

`json(arg1)`

Returns a JSON object.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	string	The value to convert to JSON.

Return value

The JSON object from the specified string, or an empty object when **null** is specified.

Remarks

If you need to include a parameter value or variable in the JSON object, use the concat function to create the string that you pass to the function.

Example

The following example template shows how to use the json function with arrays and objects:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "testValue": {
            "type": "string",
            "defaultValue": "demo value"
        }
    },
    "resources": [
    ],
    "outputs": {
        "jsonOutput": {
            "type": "object",
            "value": "[json('{\"a\": \"b\"}')]"
        },
        "nullOutput": {
            "type": "bool",
            "value": "[empty(json('null'))]"
        },
        "paramOutput": {
            "type": "object",
            "value": "[json(concat('{\"a\": \"', parameters('testValue'), '\"}'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
jsonOutput	Object	{"a": "b"}
nullOutput	Boolean	True
paramOutput	Object	{"a": "demo value"}

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/json.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/json.json
```

last

`last (arg1)`

Returns the last element of the array, or last character of the string.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	array or string	The value to retrieve the last element or character.

Return value

The type (string, int, array, or object) of the last element in an array, or the last character of a string.

Example

The following example template shows how to use the last function with an array and string.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "arrayToTest": {
            "type": "array",
            "defaultValue": ["one", "two", "three"]
        }
    },
    "resources": [
    ],
    "outputs": {
        "arrayOutput": {
            "type": "string",
            "value": "[last(parameters('arrayToTest'))]"
        },
        "stringOutput": {
            "type": "string",
            "value": "[last('One Two Three')]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayOutput	String	three

NAME	TYPE	VALUE
stringOutput	String	e

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/last.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/last.json
```

length

`length(arg1)`

Returns the number of elements in an array, or characters in a string.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	array or string	The array to use for getting the number of elements, or the string to use for getting the number of characters.

Return value

An int.

Example

The following example template shows how to use length with an array and string:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "arrayToTest": {
            "type": "array",
            "defaultValue": [
                "one",
                "two",
                "three"
            ]
        },
        "stringToTest": {
            "type": "string",
            "defaultValue": "One Two Three"
        }
    },
    "resources": [],
    "outputs": {
        "arrayLength": {
            "type": "int",
            "value": "[length(parameters('arrayToTest'))]"
        },
        "stringLength": {
            "type": "int",
            "value": "[length(parameters('stringToTest'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayLength	Int	3
stringLength	Int	13

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/length.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/length.json
```

You can use this function with an array to specify the number of iterations when creating resources. In the following example, the parameter **siteNames** would refer to an array of names to use when creating the web sites.

```
"copy": {
    "name": "websitescopy",
    "count": "[length(parameters('siteNames'))]"
}
```

For more information about using this function with an array, see Create multiple instances of resources in Azure

Resource Manager.

max

`max(arg1)`

Returns the maximum value from an array of integers or a comma-separated list of integers.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	array of integers, or comma-separated list of integers	The collection to get the maximum value.

Return value

An int representing the maximum value.

Example

The following example template shows how to use max with an array and a list of integers:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "arrayToTest": {
            "type": "array",
            "defaultValue": [0,3,2,5,4]
        }
    },
    "resources": [],
    "outputs": {
        "arrayOutput": {
            "type": "int",
            "value": "[max(parameters('arrayToTest'))]"
        },
        "intOutput": {
            "type": "int",
            "value": "[max(0,3,2,5,4)]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayOutput	Int	5
intOutput	Int	5

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/max.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/max.json
```

min

`min(arg1)`

Returns the minimum value from an array of integers or a comma-separated list of integers.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	array of integers, or comma-separated list of integers	The collection to get the minimum value.

Return value

An int representing the minimum value.

Example

The following example template shows how to use min with an array and a list of integers:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "arrayToTest": {
            "type": "array",
            "defaultValue": [0,3,2,5,4]
        }
    },
    "resources": [],
    "outputs": {
        "arrayOutput": {
            "type": "int",
            "value": "[min(parameters('arrayToTest'))]"
        },
        "intOutput": {
            "type": "int",
            "value": "[min(0,3,2,5,4)]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayOutput	Int	0
intOutput	Int	0

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/min.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/min.json
```

range

`range(startingInteger, numberOfElements)`

Creates an array of integers from a starting integer and containing a number of items.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
startingInteger	Yes	int	The first integer in the array.
numberofElements	Yes	int	The number of integers in the array.

Return value

An array of integers.

Example

The following example template shows how to use the range function:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "startingInt": {
            "type": "int",
            "defaultValue": 5
        },
        "numberOfElements": {
            "type": "int",
            "defaultValue": 3
        }
    },
    "resources": [],
    "outputs": {
        "rangeOutput": {
            "type": "array",
            "value": "[range(parameters('startingInt'),parameters('numberOfElements'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
rangeOutput	Array	[5, 6, 7]

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/range.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/range.json
```

skip

`skip(originalValue, numberToSkip)`

Returns an array with all the elements after the specified number in the array, or returns a string with all the characters after the specified number in the string.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
originalValue	Yes	array or string	The array or string to use for skipping.
numberToSkip	Yes	int	The number of elements or characters to skip. If this value is 0 or less, all the elements or characters in the value are returned. If it is larger than the length of the array or string, an empty array or string is returned.

Return value

An array or string.

Example

The following example template skips the specified number of elements in the array, and the specified number of characters in a string.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "testArray": {
            "type": "array",
            "defaultValue": [
                "one",
                "two",
                "three"
            ]
        },
        "elementsToSkip": {
            "type": "int",
            "defaultValue": 2
        },
        "testString": {
            "type": "string",
            "defaultValue": "one two three"
        },
        "charactersToSkip": {
            "type": "int",
            "defaultValue": 4
        }
    },
    "resources": [],
    "outputs": {
        "arrayOutput": {
            "type": "array",
            "value": "[skip(parameters('testArray'),parameters('elementsToSkip'))]"
        },
        "stringOutput": {
            "type": "string",
            "value": "[skip(parameters('testString'),parameters('charactersToSkip'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayOutput	Array	["three"]
stringOutput	String	two three

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/skip.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/skip.json
```

take

`take(originalValue, numberToTake)`

Returns an array with the specified number of elements from the start of the array, or a string with the specified number of characters from the start of the string.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
originalValue	Yes	array or string	The array or string to take the elements from.
numberToTake	Yes	int	The number of elements or characters to take. If this value is 0 or less, an empty array or string is returned. If it is larger than the length of the given array or string, all the elements in the array or string are returned.

Return value

An array or string.

Example

The following example template takes the specified number of elements from the array, and characters from a string.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "testArray": {
            "type": "array",
            "defaultValue": [
                "one",
                "two",
                "three"
            ]
        },
        "elementsToTake": {
            "type": "int",
            "defaultValue": 2
        },
        "testString": {
            "type": "string",
            "defaultValue": "one two three"
        },
        "charactersToTake": {
            "type": "int",
            "defaultValue": 2
        }
    },
    "resources": [],
    "outputs": {
        "arrayOutput": {
            "type": "array",
            "value": "[take(parameters('testArray'),parameters('elementsToTake'))]"
        },
        "stringOutput": {
            "type": "string",
            "value": "[take(parameters('testString'),parameters('charactersToTake'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayOutput	Array	["one", "two"]
stringOutput	String	on

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/take.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/take.json
```

union

`union(arg1, arg2, arg3, ...)`

Returns a single array or object with all elements from the parameters. Duplicate values or keys are only included once.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	array or object	The first value to use for joining elements.
arg2	Yes	array or object	The second value to use for joining elements.
additional arguments	No	array or object	Additional values to use for joining elements.

Return value

An array or object.

Example

The following example template shows how to use union with arrays and objects:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "firstObject": {
            "type": "object",
            "defaultValue": {"one": "a", "two": "b", "three": "c1"}
        },
        "secondObject": {
            "type": "object",
            "defaultValue": {"three": "c2", "four": "d", "five": "e"}
        },
        "firstArray": {
            "type": "array",
            "defaultValue": ["one", "two", "three"]
        },
        "secondArray": {
            "type": "array",
            "defaultValue": ["three", "four"]
        }
    },
    "resources": [
    ],
    "outputs": {
        "objectOutput": {
            "type": "object",
            "value": "[union(parameters('firstObject'), parameters('secondObject'))]"
        },
        "arrayOutput": {
            "type": "array",
            "value": "[union(parameters('firstArray'), parameters('secondArray'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
objectOutput	Object	{"one": "a", "two": "b", "three": "c2", "four": "d", "five": "e"}
arrayOutput	Array	["one", "two", "three", "four"]

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/union.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/union.json
```

Next steps

- For a description of the sections in an Azure Resource Manager template, see Authoring Azure Resource Manager templates.
- To merge multiple templates, see Using linked templates with Azure Resource Manager.
- To iterate a specified number of times when creating a type of resource, see Create multiple instances of resources in Azure Resource Manager.
- To see how to deploy the template you have created, see Deploy an application with Azure Resource Manager template.

Comparison functions for Azure Resource Manager templates

6/18/2019 • 6 minutes to read • Edit Online

Resource Manager provides several functions for making comparisons in your templates.

- equals
- greater
- greaterOrEquals
- less
- lessOrEquals

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

equals

`equals(arg1, arg2)`

Checks whether two values equal each other.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	int, string, array, or object	The first value to check for equality.
arg2	Yes	int, string, array, or object	The second value to check for equality.

Return value

Returns **True** if the values are equal; otherwise, **False**.

Remarks

The equals function is often used with the `condition` element to test whether a resource is deployed.

```
{
    "condition": "[equals(parameters('newOrExisting'),'new')]",
    "type": "Microsoft.Storage/storageAccounts",
    "name": "[variables('storageAccountName')]",
    "apiVersion": "2017-06-01",
    "location": "[resourceGroup().location]",
    "sku": {
        "name": "[variables('storageAccountType')]"
    },
    "kind": "Storage",
    "properties": {}
}
```

Example

The following example template checks different types of values for equality. All the default values return True.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "firstInt": {
            "type": "int",
            "defaultValue": 1
        },
        "secondInt": {
            "type": "int",
            "defaultValue": 1
        },
        "firstString": {
            "type": "string",
            "defaultValue": "a"
        },
        "secondString": {
            "type": "string",
            "defaultValue": "a"
        },
        "firstArray": {
            "type": "array",
            "defaultValue": ["a", "b"]
        },
        "secondArray": {
            "type": "array",
            "defaultValue": ["a", "b"]
        },
        "firstObject": {
            "type": "object",
            "defaultValue": {"a": "b"}
        },
        "secondObject": {
            "type": "object",
            "defaultValue": {"a": "b"}
        }
    },
    "resources": [
    ],
    "outputs": {
        "checkInts": {
            "type": "bool",
            "value": "[equals(parameters('firstInt'), parameters('secondInt') )]"
        },
        "checkStrings": {
            "type": "bool",
            "value": "[equals(parameters('firstString'), parameters('secondString'))]"
        },
        "checkArrays": {
            "type": "bool",
            "value": "[equals(parameters('firstArray'), parameters('secondArray'))]"
        },
        "checkObjects": {
            "type": "bool",
            "value": "[equals(parameters('firstObject'), parameters('secondObject'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
checkInts	Bool	True

NAME	TYPE	VALUE
checkStrings	Bool	True
checkArrays	Bool	True
checkObjects	Bool	True

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/equals.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/equals.json
```

The following example template uses not with **equals**.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "resources": [
    ],
    "outputs": {
        "checkNotEquals": {
            "type": "bool",
            "value": "[not(equals(1, 2))]"
        }
    }
}
```

The output from the preceding example is:

NAME	TYPE	VALUE
checkNotEquals	Bool	True

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-manager/functions/not-
equals.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-manager/functions/not-
equals.json
```

greater

`greater(arg1, arg2)`

Checks whether the first value is greater than the second value.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	int or string	The first value for the greater comparison.
arg2	Yes	int or string	The second value for the greater comparison.

Return value

Returns **True** if the first value is greater than the second value; otherwise, **False**.

Example

The following example template checks whether the one value is greater than the other.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "firstInt": {
            "type": "int",
            "defaultValue": 1
        },
        "secondInt": {
            "type": "int",
            "defaultValue": 2
        },
        "firstString": {
            "type": "string",
            "defaultValue": "A"
        },
        "secondString": {
            "type": "string",
            "defaultValue": "a"
        }
    },
    "resources": [
    ],
    "outputs": {
        "checkInts": {
            "type": "bool",
            "value": "[greater(parameters('firstInt'), parameters('secondInt') )]"
        },
        "checkStrings": {
            "type": "bool",
            "value": "[greater(parameters('firstString'), parameters('secondString'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
checkInts	Bool	False

NAME	TYPE	VALUE
checkStrings	Bool	True

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/greater.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/greater.json
```

greaterOrEquals

`greaterOrEquals(arg1, arg2)`

Checks whether the first value is greater than or equal to the second value.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	int or string	The first value for the greater or equal comparison.
arg2	Yes	int or string	The second value for the greater or equal comparison.

Return value

Returns **True** if the first value is greater than or equal to the second value; otherwise, **False**.

Example

The following example template checks whether the one value is greater than or equal to the other.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "firstInt": {
            "type": "int",
            "defaultValue": 1
        },
        "secondInt": {
            "type": "int",
            "defaultValue": 2
        },
        "firstString": {
            "type": "string",
            "defaultValue": "A"
        },
        "secondString": {
            "type": "string",
            "defaultValue": "a"
        }
    },
    "resources": [
    ],
    "outputs": {
        "checkInts": {
            "type": "bool",
            "value": "[greaterOrEquals(parameters('firstInt'), parameters('secondInt') )]"
        },
        "checkStrings": {
            "type": "bool",
            "value": "[greaterOrEquals(parameters('firstString'), parameters('secondString'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
checkInts	Bool	False
checkStrings	Bool	True

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/greaterorequals.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/greaterorequals.json
```

less

`less(arg1, arg2)`

Checks whether the first value is less than the second value.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	int or string	The first value for the less comparison.
arg2	Yes	int or string	The second value for the less comparison.

Return value

Returns **True** if the first value is less than the second value; otherwise, **False**.

Example

The following example template checks whether the one value is less than the other.

```json
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "firstInt": {
            "type": "int",
            "defaultValue": 1
        },
        "secondInt": {
            "type": "int",
            "defaultValue": 2
        },
        "firstString": {
            "type": "string",
            "defaultValue": "A"
        },
        "secondString": {
            "type": "string",
            "defaultValue": "a"
        }
    },
    "resources": [
    ],
    "outputs": {
        "checkInts": {
            "type": "bool",
            "value": "[less(parameters('firstInt'), parameters('secondInt') )]"
        },
        "checkStrings": {
            "type": "bool",
            "value": "[less(parameters('firstString'), parameters('secondString'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
checkInts	Bool	True
checkStrings	Bool	False

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/less.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/less.json
```

lessOrEquals

`lessOrEquals(arg1, arg2)`

Checks whether the first value is less than or equal to the second value.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	int or string	The first value for the less or equals comparison.
arg2	Yes	int or string	The second value for the less or equals comparison.

Return value

Returns **True** if the first value is less than or equal to the second value; otherwise, **False**.

Example

The following example template checks whether the one value is less than or equal to the other.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "firstInt": {
            "type": "int",
            "defaultValue": 1
        },
        "secondInt": {
            "type": "int",
            "defaultValue": 2
        },
        "firstString": {
            "type": "string",
            "defaultValue": "A"
        },
        "secondString": {
            "type": "string",
            "defaultValue": "a"
        }
    },
    "resources": [
    ],
    "outputs": {
        "checkInts": {
            "type": "bool",
            "value": "[lessOrEquals(parameters('firstInt'), parameters('secondInt') )]"
        },
        "checkStrings": {
            "type": "bool",
            "value": "[lessOrEquals(parameters('firstString'), parameters('secondString'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
checkInts	Bool	True
checkStrings	Bool	False

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/lessorequals.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/lessorequals.json
```

Next steps

- For a description of the sections in an Azure Resource Manager template, see Authoring Azure Resource Manager templates.

- To merge multiple templates, see Using linked templates with Azure Resource Manager.
- To iterate a specified number of times when creating a type of resource, see Create multiple instances of resources in Azure Resource Manager.
- To see how to deploy the template you have created, see Deploy an application with Azure Resource Manager template.

Deployment functions for Azure Resource Manager templates

6/18/2019 • 5 minutes to read • Edit Online

Resource Manager provides the following functions for getting values from sections of the template and values related to the deployment:

- deployment
- parameters
- variables

To get values from resources, resource groups, or subscriptions, see Resource functions.

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

deployment

`deployment()`

Returns information about the current deployment operation.

Return value

This function returns the object that is passed during deployment. The properties in the returned object differ based on whether the deployment object is passed as a link or as an in-line object. When the deployment object is passed in-line, such as when using the **-TemplateFile** parameter in Azure PowerShell to point to a local file, the returned object has the following format:

```
{
    "name": "",
    "properties": {
        "template": {
            "$schema": "",
            "contentVersion": "",
            "parameters": {},
            "variables": {},
            "resources": [
            ],
            "outputs": {}
        },
        "parameters": {},
        "mode": "",
        "provisioningState": ""
    }
}
```

When the object is passed as a link, such as when using the **-TemplateUri** parameter to point to a remote object, the object is returned in the following format:

```
{
    "name": "",
    "properties": {
        "templateLink": {
            "uri": ""
        },
        "template": {
            "$schema": "",
            "contentVersion": "",
            "parameters": {},
            "variables": {},
            "resources": [],
            "outputs": {}
        },
        "parameters": {},
        "mode": "",
        "provisioningState": ""
    }
}
```

When you deploy to an Azure subscription, instead of a resource group, the return object includes a `location` property. The location property is included when deploying either a local template or an external template.

Remarks

You can use deployment() to link to another template based on the URI of the parent template.

```
"variables": {
    "sharedTemplateUrl": "[uri(deployment().properties.templateLink.uri, 'shared-resources.json')]"
}
```

If you redeploy a template from the deployment history in the portal, the template is deployed as a local file. The `templateLink` property isn't returned in the deployment function. If your template relies on `templateLink` to construct a link to another template, don't use the portal to redeploy. Instead, use the commands you used to originally deploy the template.

Example

The following example template returns the deployment object:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "resources": [],
    "outputs": {
        "subscriptionOutput": {
            "value": "[deployment()]",
            "type" : "object"
        }
    }
}
```

The preceding example returns the following object:

```
{
  "name": "deployment",
  "properties": {
    "template": {
      "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
      "contentVersion": "1.0.0.0",
      "resources": [],
      "outputs": {
        "subscriptionOutput": {
          "type": "Object",
          "value": "[deployment()]"
        }
      }
    },
    "parameters": {},
    "mode": "Incremental",
    "provisioningState": "Accepted"
  }
}
```

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/deployment.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/deployment.json
```

For a subscription-level template that uses the deployment function, see subscription deployment function. It's deployed with either `az deployment create` or `New-AzDeployment` commands.

parameters

`parameters(parameterName)`

Returns a parameter value. The specified parameter name must be defined in the parameters section of the template.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
parameterName	Yes	string	The name of the parameter to return.

Return value

The value of the specified parameter.

Remarks

Typically, you use parameters to set resource values. The following example sets the name of web site to the parameter value passed in during deployment.

```
"parameters": {
  "siteName": {
      "type": "string"
  }
},
"resources": [
   {
      "apiVersion": "2016-08-01",
      "name": "[parameters('siteName')]",
      "type": "Microsoft.Web/Sites",
      ...
   }
]
```

Example

The following example template shows a simplified use of the parameters function.

```
{
 "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
 "contentVersion": "1.0.0.0",
 "parameters": {
  "stringParameter": {
   "type" : "string",
   "defaultValue": "option 1"
  },
        "intParameter": {
            "type": "int",
            "defaultValue": 1
        },
        "objectParameter": {
            "type": "object",
            "defaultValue": {"one": "a", "two": "b"}
        },
        "arrayParameter": {
            "type": "array",
            "defaultValue": [1, 2, 3]
        },
        "crossParameter": {
            "type": "string",
            "defaultValue": "[parameters('stringParameter')]"
        }
 },
 "variables": {},
 "resources": [],
 "outputs": {
  "stringOutput": {
   "value": "[parameters('stringParameter')]",
   "type" : "string"
  },
        "intOutput": {
   "value": "[parameters('intParameter')]",
   "type" : "int"
  },
        "objectOutput": {
   "value": "[parameters('objectParameter')]",
   "type" : "object"
  },
        "arrayOutput": {
   "value": "[parameters('arrayParameter')]",
   "type" : "array"
  },
        "crossOutput": {
   "value": "[parameters('crossParameter')]",
   "type" : "string"
  }
 }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
stringOutput	String	option 1
intOutput	Int	1
objectOutput	Object	{"one": "a", "two": "b"}
arrayOutput	Array	[1, 2, 3]

NAME	TYPE	VALUE
crossOutput	String	option 1

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/parameters.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/parameters.json
```

variables

`variables(variableName)`

Returns the value of variable. The specified variable name must be defined in the variables section of the template.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
variableName	Yes	String	The name of the variable to return.

Return value

The value of the specified variable.

Remarks

Typically, you use variables to simplify your template by constructing complex values only once. The following example constructs a unique name for a storage account.

```
"variables": {
    "storageName": "[concat('storage', uniqueString(resourceGroup().id))]"
},
"resources": [
    {
        "type": "Microsoft.Storage/storageAccounts",
        "name": "[variables('storageName')]",
        ...
    },
    {
        "type": "Microsoft.Compute/virtualMachines",
        "dependsOn": [
            "[variables('storageName')]"
        ],
        ...
    }
],
```

Example

The following example template returns different variable values.

```
{
 "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
 "contentVersion": "1.0.0.0",
 "parameters": {},
 "variables": {
  "var1": "myVariable",
  "var2": [ 1,2,3,4 ],
  "var3": "[ variables('var1') ]",
  "var4": {
   "property1": "value1",
   "property2": "value2"
    }
 },
 "resources": [],
 "outputs": {
  "exampleOutput1": {
   "value": "[variables('var1')]",
   "type" : "string"
  },
  "exampleOutput2": {
   "value": "[variables('var2')]",
   "type" : "array"
  },
  "exampleOutput3": {
   "value": "[variables('var3')]",
   "type" : "string"
  },
  "exampleOutput4": {
   "value": "[variables('var4')]",
   "type" : "object"
  }
 }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
exampleOutput1	String	myVariable
exampleOutput2	Array	[1, 2, 3, 4]
exampleOutput3	String	myVariable
exampleOutput4	Object	{"property1": "value1", "property2": "value2"}

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/variables.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/variables.json
```

Next steps

- For a description of the sections in an Azure Resource Manager template, see Authoring Azure Resource Manager templates.
- To merge several templates, see Using linked templates with Azure Resource Manager.
- To iterate a specified number of times when creating a type of resource, see Create multiple instances of resources in Azure Resource Manager.
- To see how to deploy the template you've created, see Deploy an application with Azure Resource Manager template.

Logical functions for Azure Resource Manager templates

6/26/2019 • 4 minutes to read • Edit Online

Resource Manager provides several functions for making comparisons in your templates.

- and
- bool
- if
- not
- or

and

```
and(arg1, arg2, ...)
```

Checks whether all parameter values are true.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	boolean	The first value to check whether is true.
arg2	Yes	boolean	The second value to check whether is true.
additional arguments	No	boolean	Additional arguments to check whether are true.

Return value

Returns **True** if all values are true; otherwise, **False**.

Examples

The following example template shows how to use logical functions.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "resources": [ ],
    "outputs": {
        "andExampleOutput": {
            "value": "[and(bool('true'), bool('false'))]",
            "type": "bool"
        },
        "orExampleOutput": {
            "value": "[or(bool('true'), bool('false'))]",
            "type": "bool"
        },
        "notExampleOutput": {
            "value": "[not(bool('true'))]",
            "type": "bool"
        }
    }
}
```

The output from the preceding example is:

NAME	TYPE	VALUE
andExampleOutput	Bool	False
orExampleOutput	Bool	True
notExampleOutput	Bool	False

bool

`bool(arg1)`

Converts the parameter to a boolean.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	string or int	The value to convert to a boolean.

Return value

A boolean of the converted value.

Examples

The following example template shows how to use bool with a string or integer.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "resources": [],
    "outputs": {
        "trueString": {
            "value": "[bool('true')]",
            "type" : "bool"
        },
        "falseString": {
            "value": "[bool('false')]",
            "type" : "bool"
        },
        "trueInt": {
            "value": "[bool(1)]",
            "type" : "bool"
        },
        "falseInt": {
            "value": "[bool(0)]",
            "type" : "bool"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
trueString	Bool	True
falseString	Bool	False
trueInt	Bool	True
falseInt	Bool	False

if

`if(condition, trueValue, falseValue)`

Returns a value based on whether a condition is true or false.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
condition	Yes	boolean	The value to check whether it's true or false.
trueValue	Yes	string, int, object, or array	The value to return when the condition is true.
falseValue	Yes	string, int, object, or array	The value to return when the condition is false.

Return value

Returns second parameter when first parameter is **True**; otherwise, returns third parameter.

Remarks

When the condition is **True**, only the true value is evaluated. When the condition is **False**, only the false value is evaluated. With the **if** function, you can include expressions that are only conditionally valid. For example, you can reference a resource that exists under one condition but not under the other condition. An example of conditionally evaluating expressions is shown in the following section.

Examples

The following example template shows how to use the `if` function.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "resources": [
    ],
    "outputs": {
        "yesOutput": {
            "type": "string",
            "value": "[if(equals('a', 'a'), 'yes', 'no')]"
        },
        "noOutput": {
            "type": "string",
            "value": "[if(equals('a', 'b'), 'yes', 'no')]"
        },
        "objectOutput": {
            "type": "object",
            "value": "[if(equals('a', 'a'), json('{\"test\": \"value1\"}'), json('null'))]"
        }
    }
}
```

The output from the preceding example is:

NAME	TYPE	VALUE
yesOutput	String	yes
noOutput	String	no
objectOutput	Object	{ "test": "value1" }

The following example template shows how to use this function with expressions that are only conditionally valid.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "vmName": {
            "type": "string"
        },
        "location": {
            "type": "string"
        },
        "logAnalytics": {
            "type": "string",
            "defaultValue": ""
        }
    },
    "resources": [
        {
            "condition": "[not(empty(parameters('logAnalytics')))]",
            "name": "[concat(parameters('vmName'),'/omsOnboarding')]",
            "type": "Microsoft.Compute/virtualMachines/extensions",
            "location": "[parameters('location')]",
            "apiVersion": "2017-03-30",
            "properties": {
                "publisher": "Microsoft.EnterpriseCloud.Monitoring",
                "type": "MicrosoftMonitoringAgent",
                "typeHandlerVersion": "1.0",
                "autoUpgradeMinorVersion": true,
                "settings": {
                    "workspaceId": "[if(not(empty(parameters('logAnalytics'))),
reference(parameters('logAnalytics'), '2015-11-01-preview').customerId, json('null'))]"
                },
                "protectedSettings": {
                    "workspaceKey": "[if(not(empty(parameters('logAnalytics'))),
listKeys(parameters('logAnalytics'), '2015-11-01-preview').primarySharedKey, json('null'))]"
                }
            }
        }
    ],
    "outputs": {
        "mgmtStatus": {
            "type": "string",
            "value": "[if(not(empty(parameters('logAnalytics'))), 'Enabled monitoring for VM!', 'Nothing to
enable')]"
        }
    }
}
```

not

`not(arg1)`

Converts boolean value to its opposite value.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	boolean	The value to convert.

Return value

Returns **True** when parameter is **False**. Returns **False** when parameter is **True**.

Examples

The following example template shows how to use logical functions.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "resources": [ ],
    "outputs": {
        "andExampleOutput": {
            "value": "[and(bool('true'), bool('false'))]",
            "type": "bool"
        },
        "orExampleOutput": {
            "value": "[or(bool('true'), bool('false'))]",
            "type": "bool"
        },
        "notExampleOutput": {
            "value": "[not(bool('true'))]",
            "type": "bool"
        }
    }
}
```

The output from the preceding example is:

NAME	TYPE	VALUE
andExampleOutput	Bool	False
orExampleOutput	Bool	True
notExampleOutput	Bool	False

The following example template uses **not** with equals.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "resources": [
    ],
    "outputs": {
        "checkNotEquals": {
            "type": "bool",
            "value": "[not(equals(1, 2))]"
        }
    }
```

The output from the preceding example is:

NAME	TYPE	VALUE
checkNotEquals	Bool	True

or

`or(arg1, arg2, ...)`

Checks whether any parameter value is true.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	boolean	The first value to check whether is true.
arg2	Yes	boolean	The second value to check whether is true.
additional arguments	No	boolean	Additional arguments to check whether are true.

Return value

Returns **True** if any value is true; otherwise, **False**.

Examples

The following example template shows how to use logical functions.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "resources": [ ],
    "outputs": {
        "andExampleOutput": {
            "value": "[and(bool('true'), bool('false'))]",
            "type": "bool"
        },
        "orExampleOutput": {
            "value": "[or(bool('true'), bool('false'))]",
            "type": "bool"
        },
        "notExampleOutput": {
            "value": "[not(bool('true'))]",
            "type": "bool"
        }
    }
}
```

The output from the preceding example is:

NAME	TYPE	VALUE
andExampleOutput	Bool	False
orExampleOutput	Bool	True
notExampleOutput	Bool	False

Next steps

- For a description of the sections in an Azure Resource Manager template, see Authoring Azure Resource Manager templates.
- To merge multiple templates, see Using linked templates with Azure Resource Manager.
- To iterate a specified number of times when creating a type of resource, see Create multiple instances of resources in Azure Resource Manager.
- To see how to deploy the template you've created, see Deploy an application with Azure Resource Manager template.

Numeric functions for Azure Resource Manager templates

6/18/2019 • 8 minutes to read • Edit Online

Resource Manager provides the following functions for working with integers:

- add
- copyIndex
- div
- float
- int
- max
- min
- mod
- mul
- sub

NOTE
This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

add

`add(operand1, operand2)`

Returns the sum of the two provided integers.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
operand1	Yes	int	First number to add.
operand2	Yes	int	Second number to add.

Return value

An integer that contains the sum of the parameters.

Example

The following example template adds two parameters.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "first": {
            "type": "int",
            "defaultValue": 5,
            "metadata": {
                "description": "First integer to add"
            }
        },
        "second": {
            "type": "int",
            "defaultValue": 3,
            "metadata": {
                "description": "Second integer to add"
            }
        }
    },
    "resources": [
    ],
    "outputs": {
        "addResult": {
            "type": "int",
            "value": "[add(parameters('first'), parameters('second'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
addResult	Int	8

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/add.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/add.json
```

copyIndex

`copyIndex(loopName, offset)`

Returns the index of an iteration loop.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
loopName	No	string	The name of the loop for getting the iteration.

PARAMETER	REQUIRED	TYPE	DESCRIPTION
offset	No	int	The number to add to the zero-based iteration value.

Remarks

This function is always used with a **copy** object. If no value is provided for **offset**, the current iteration value is returned. The iteration value starts at zero. You can use iteration loops when defining either resources or variables.

The **loopName** property enables you to specify whether copyIndex is referring to a resource iteration or property iteration. If no value is provided for **loopName**, the current resource type iteration is used. Provide a value for **loopName** when iterating on a property.

For a complete description of how you use **copyIndex**, see Create multiple instances of resources in Azure Resource Manager.

For an example of using **copyIndex** when defining a variable, see Variables.

Example

The following example shows a copy loop and the index value included in the name.

```
"resources": [
  {
    "name": "[concat('examplecopy-', copyIndex())]",
    "type": "Microsoft.Web/sites",
    "copy": {
      "name": "websitescopy",
      "count": "[parameters('count')]"
    },
    ...
  }
]
```

Return value

An integer representing the current index of the iteration.

div

`div(operand1, operand2)`

Returns the integer division of the two provided integers.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
operand1	Yes	int	The number being divided.
operand2	Yes	int	The number that is used to divide. Cannot be 0.

Return value

An integer representing the division.

Example

The following example template divides one parameter by another parameter.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "first": {
            "type": "int",
            "defaultValue": 8,
            "metadata": {
                "description": "Integer being divided"
            }
        },
        "second": {
            "type": "int",
            "defaultValue": 3,
            "metadata": {
                "description": "Integer used to divide"
            }
        }
    },
    "resources": [
    ],
    "outputs": {
        "divResult": {
            "type": "int",
            "value": "[div(parameters('first'), parameters('second'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
divResult	Int	2

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/div.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/div.json
```

float

`float(arg1)`

Converts the value to a floating point number. You only use this function when passing custom parameters to an application, such as a Logic App.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	string or int	The value to convert to a floating point number.

Return value

A floating point number.

Example

The following example shows how to use float to pass parameters to a Logic App:

```
{
    "type": "Microsoft.Logic/workflows",
    "properties": {
        ...
        "parameters": {
            "custom1": {
                "value": "[float('3.0')]"
            },
            "custom2": {
                "value": "[float(3)]"
            },
```

int

`int(valueToConvert)`

Converts the specified value to an integer.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
valueToConvert	Yes	string or int	The value to convert to an integer.

Return value

An integer of the converted value.

Example

The following example template converts the user-provided parameter value to integer.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "stringToConvert": {
            "type": "string",
            "defaultValue": "4"
        }
    },
    "resources": [
    ],
    "outputs": {
        "intResult": {
            "type": "int",
            "value": "[int(parameters('stringToConvert'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
intResult	Int	4

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/int.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/int.json
```

max

`max (arg1)`

Returns the maximum value from an array of integers or a comma-separated list of integers.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	array of integers, or comma-separated list of integers	The collection to get the maximum value.

Return value

An integer representing the maximum value from the collection.

Example

The following example template shows how to use max with an array and a list of integers:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "arrayToTest": {
            "type": "array",
            "defaultValue": [0,3,2,5,4]
        }
    },
    "resources": [],
    "outputs": {
        "arrayOutput": {
            "type": "int",
            "value": "[max(parameters('arrayToTest'))]"
        },
        "intOutput": {
            "type": "int",
            "value": "[max(0,3,2,5,4)]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayOutput	Int	5
intOutput	Int	5

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/max.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/max.json
```

min

`min (arg1)`

Returns the minimum value from an array of integers or a comma-separated list of integers.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	array of integers, or comma-separated list of integers	The collection to get the minimum value.

Return value

An integer representing minimum value from the collection.

Example

The following example template shows how to use min with an array and a list of integers:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "arrayToTest": {
            "type": "array",
            "defaultValue": [0,3,2,5,4]
        }
    },
    "resources": [],
    "outputs": {
        "arrayOutput": {
            "type": "int",
            "value": "[min(parameters('arrayToTest'))]"
        },
        "intOutput": {
            "type": "int",
            "value": "[min(0,3,2,5,4)]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayOutput	Int	0
intOutput	Int	0

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/min.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/min.json
```

mod

`mod(operand1, operand2)`

Returns the remainder of the integer division using the two provided integers.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
operand1	Yes	int	The number being divided.

PARAMETER	REQUIRED	TYPE	DESCRIPTION
operand2	Yes	int	The number that is used to divide, Cannot be 0.

Return value

An integer representing the remainder.

Example

The following example template returns the remainder of dividing one parameter by another parameter.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "first": {
            "type": "int",
            "defaultValue": 7,
            "metadata": {
                "description": "Integer being divided"
            }
        },
        "second": {
            "type": "int",
            "defaultValue": 3,
            "metadata": {
                "description": "Integer used to divide"
            }
        }
    },
    "resources": [
    ],
    "outputs": {
        "modResult": {
            "type": "int",
            "value": "[mod(parameters('first'), parameters('second'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
modResult	Int	1

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/mod.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/mod.json
```

mul

`mul(operand1, operand2)`

Returns the multiplication of the two provided integers.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
operand1	Yes	int	First number to multiply.
operand2	Yes	int	Second number to multiply.

Return value

An integer representing the multiplication.

Example

The following example template multiplies one parameter by another parameter.

```json
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "first": {
            "type": "int",
            "defaultValue": 5,
            "metadata": {
                "description": "First integer to multiply"
            }
        },
        "second": {
            "type": "int",
            "defaultValue": 3,
            "metadata": {
                "description": "Second integer to multiply"
            }
        }
    },
    "resources": [
    ],
    "outputs": {
        "mulResult": {
            "type": "int",
            "value": "[mul(parameters('first'), parameters('second'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
mulResult	Int	15

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/mul.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/mul.json
```

sub

`sub(operand1, operand2)`

Returns the subtraction of the two provided integers.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
operand1	Yes	int	The number that is subtracted from.
operand2	Yes	int	The number that is subtracted.

Return value

An integer representing the subtraction.

Example

The following example template subtracts one parameter from another parameter.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "first": {
            "type": "int",
            "defaultValue": 7,
            "metadata": {
                "description": "Integer subtracted from"
            }
        },
        "second": {
            "type": "int",
            "defaultValue": 3,
            "metadata": {
                "description": "Integer to subtract"
            }
        }
    },
    "resources": [
    ],
    "outputs": {
        "subResult": {
            "type": "int",
            "value": "[sub(parameters('first'), parameters('second'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
subResult	Int	4

To deploy this example template with Azure CLI, use:

```
az group deployment create -g functionexamplegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/sub.json
```

To deploy this example template with PowerShell, use:

```
New-AzResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/sub.json
```

Next steps

- For a description of the sections in an Azure Resource Manager template, see Authoring Azure Resource Manager templates.
- To merge multiple templates, see Using linked templates with Azure Resource Manager.
- To iterate a specified number of times when creating a type of resource, see Create multiple instances of resources in Azure Resource Manager.
- To see how to deploy the template you have created, see Deploy an application with Azure Resource Manager template.

Resource functions for Azure Resource Manager templates

7/18/2019 • 13 minutes to read • Edit Online

Resource Manager provides the following functions for getting resource values:

- list*
- providers
- reference
- resourceGroup
- resourceId
- subscription

To get values from parameters, variables, or the current deployment, see Deployment value functions.

list*

`list{Value}(resourceName or resourceIdentifier, apiVersion, functionValues)`

The syntax for this function varies by name of the list operations. Each implementation returns values for the resource type that supports a list operation. The operation name must start with `list`. Some common usages are `listKeys` and `listSecrets`.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
resourceName or resourceIdentifier	Yes	string	Unique identifier for the resource.
apiVersion	Yes	string	API version of resource runtime state. Typically, in the format, **yyyy-mm-dd**.
functionValues	No	object	An object that has values for the function. Only provide this object for functions that support receiving an object with parameter values, such as **listAccountSas** on a storage account. An example of passing function values is shown in this article.

Implementations

The possible uses of list* are shown in the following table.

RESOURCE TYPE	FUNCTION NAME
Microsoft.AnalysisServices/servers	listGatewayStatus

RESOURCE TYPE	FUNCTION NAME
Microsoft.AppConfiguration/configurationStores	ListKeys
Microsoft.Automation/automationAccounts	listKeys
Microsoft.Batch/batchAccounts	listkeys
Microsoft.BatchAI/workspaces/experiments/jobs	listoutputfiles
Microsoft.Blockchain/blockchainMembers	listApiKeys
Microsoft.Blockchain/blockchainMembers/transactionNodes	listApiKeys
Microsoft.BotService/botServices/channels	listChannelWithKeys
Microsoft.Cache/redis	listKeys
Microsoft.CognitiveServices/accounts	listKeys
Microsoft.ContainerRegistry/registries	listBuildSourceUploadUrl
Microsoft.ContainerRegistry/registries	listCredentials
Microsoft.ContainerRegistry/registries	listPolicies
Microsoft.ContainerRegistry/registries	listUsages
Microsoft.ContainerRegistry/registries/webhooks	listEvents
Microsoft.ContainerRegistry/registries/runs	listLogSasUrl
Microsoft.ContainerRegistry/registries/tasks	listDetails
Microsoft.ContainerService/managedClusters	listClusterAdminCredential
Microsoft.ContainerService/managedClusters	listClusterUserCredential
Microsoft.ContainerService/managedClusters/accessProfiles	listCredential
Microsoft.DataBox/jobs	listCredentials
Microsoft.DataFactory/datafactories/gateways	listauthkeys
Microsoft.DataFactory/factories/integrationruntimes	listauthkeys
Microsoft.DataLakeAnalytics/accounts/storageAccounts/Containers	listSasTokens
Microsoft.Devices/iotHubs	listkeys
Microsoft.Devices/provisioningServices/keys	listkeys

RESOURCE TYPE	FUNCTION NAME
Microsoft.Devices/provisioningServices	listkeys
Microsoft.DevTestLab/labs	ListVhds
Microsoft.DevTestLab/labs/schedules	ListApplicable
Microsoft.DevTestLab/labs/users/serviceFabrics	ListApplicableSchedules
Microsoft.DevTestLab/labs/virtualMachines	ListApplicableSchedules
Microsoft.DocumentDB/databaseAccounts	listConnectionStrings
Microsoft.DocumentDB/databaseAccounts	listKeys
Microsoft.DomainRegistration	listDomainRecommendations
Microsoft.DomainRegistration/topLevelDomains	listAgreements
Microsoft.EventGrid/domains	listKeys
Microsoft.EventGrid/topics	listKeys
Microsoft.EventHub/namespaces/authorizationRules	listkeys
Microsoft.EventHub/namespaces/disasterRecoveryConfigs/authorizationRules	listkeys
Microsoft.EventHub/namespaces/eventhubs/authorizationRules	listkeys
Microsoft.ImportExport/jobs	listBitLockerKeys
Microsoft.LabServices/users	ListEnvironments
Microsoft.LabServices/users	ListLabs
Microsoft.Logic/integrationAccounts/agreements	listContentCallbackUrl
Microsoft.Logic/integrationAccounts/assemblies	listContentCallbackUrl
Microsoft.Logic/integrationAccounts	listCallbackUrl
Microsoft.Logic/integrationAccounts	listKeyVaultKeys
Microsoft.Logic/integrationAccounts/maps	listContentCallbackUrl
Microsoft.Logic/integrationAccounts/partners	listContentCallbackUrl
Microsoft.Logic/integrationAccounts/schemas	listContentCallbackUrl

RESOURCE TYPE	FUNCTION NAME
Microsoft.Logic/workflows	listCallbackUrl
Microsoft.Logic/workflows	listSwagger
Microsoft.Logic/workflows/triggers	listCallbackUrl
Microsoft.Logic/workflows/versions/triggers	listCallbackUrl
Microsoft.MachineLearning/webServices	listkeys
Microsoft.MachineLearning/Workspaces	listworkspacekeys
Microsoft.MachineLearningServices/workspaces/computes	listKeys
Microsoft.MachineLearningServices/workspaces	listKeys
Microsoft.Maps/accounts	listKeys
Microsoft.Media/mediaservices/assets	listContainerSas
Microsoft.Media/mediaservices/assets	listStreamingLocators
Microsoft.Media/mediaservices/streamingLocators	listContentKeys
Microsoft.Media/mediaservices/streamingLocators	listPaths
Microsoft.Network/applicationSecurityGroups	listIpConfigurations
Microsoft.NotificationHubs/Namespaces/authorizationRules	listkeys
Microsoft.NotificationHubs/Namespaces/NotificationHubs/authorizationRules	listkeys
Microsoft.OperationalInsights/workspaces	listKeys
Microsoft.Relay/namespaces/authorizationRules	listkeys
Microsoft.Relay/namespaces/disasterRecoveryConfigs/authorizationRules	listkeys
Microsoft.Relay/namespaces/HybridConnections/authorizationRules	listkeys
Microsoft.Relay/namespaces/WcfRelays/authorizationRules	listkeys
Microsoft.Search/searchServices	listAdminKeys
Microsoft.Search/searchServices	listQueryKeys
Microsoft.ServiceBus/namespaces/authorizationRules	listkeys

RESOURCE TYPE	FUNCTION NAME
Microsoft.ServiceBus/namespaces/disasterRecoveryConfigs/authorizationRules	listkeys
Microsoft.ServiceBus/namespaces/queues/authorizationRules	listkeys
Microsoft.ServiceBus/namespaces/topics/authorizationRules	listkeys
Microsoft.SignalRService/SignalR	listkeys
Microsoft.Storage/storageAccounts	listAccountSas
Microsoft.Storage/storageAccounts	listkeys
Microsoft.Storage/storageAccounts	listServiceSas
Microsoft.StorSimple/managers/devices	listFailoverSets
Microsoft.StorSimple/managers/devices	listFailoverTargets
Microsoft.StorSimple/managers	listActivationKey
Microsoft.StorSimple/managers	listPublicEncryptionKey
Microsoft.Web/connectionGateways	ListStatus
microsoft.web/connections	listconsentlinks
Microsoft.Web/customApis	listWsdlInterfaces
microsoft.web/locations	listwsdlinterfaces
microsoft.web/apimanagementaccounts/apis/connections	listconnectionkeys
microsoft.web/apimanagementaccounts/apis/connections	listsecrets
microsoft.web/sites/functions	listsecrets
microsoft.web/sites/hybridconnectionnamespaces/relays	listkeys
microsoft.web/sites	listsyncfunctiontriggerstatus
microsoft.web/sites/slots/functions	listsecrets

To determine which resource types have a list operation, you have the following options:

- View the REST API operations for a resource provider, and look for list operations. For example, storage accounts have the listKeys operation.
- Use the Get-AzProviderOperation PowerShell cmdlet. The following example gets all list operations for storage accounts:

```
Get-AzProviderOperation -OperationSearchString "Microsoft.Storage/*" | where {$_.Operation -like
"*list*"} | FT Operation
```

- Use the following Azure CLI command to filter only the list operations:

```
az provider operation show --namespace Microsoft.Storage --query "resourceTypes[?
name=='storageAccounts'].operations[].name | [?contains(@, 'list')]"
```

Return value

The returned object varies by the list function you use. For example, the listKeys for a storage account returns the following format:

```
{
  "keys": [
    {
      "keyName": "key1",
      "permissions": "Full",
      "value": "{value}"
    },
    {
      "keyName": "key2",
      "permissions": "Full",
      "value": "{value}"
    }
  ]
}
```

Other list functions have different return formats. To see the format of a function, include it in the outputs section as shown in the example template.

Remarks

Specify the resource by using either the resource name or the resourceId function. When using a list function in the same template that deploys the referenced resource, use the resource name.

If you use a **list** function in a resource that is conditionally deployed, the function is evaluated even if the resource isn't deployed. You get an error if the **list** function refers to a resource that doesn't exist. Use the **if** function to make sure the function is only evaluated when the resource is being deployed. See the if function for a sample template that uses if and list with a conditionally deployed resource.

Example

The following example template shows how to return the primary and secondary keys from a storage account in the outputs section. It also returns a SAS token for the storage account.

To get the SAS token, pass an object for the expiry time. The expiry time must be in the future. This example is intended to show how you use the list functions. Typically, you would use the SAS token in a resource value rather than return it as an output value. Output values are stored in the deployment history and aren't secure.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "storagename": {
            "type": "string"
        },
        "location": {
            "type": "string",
            "defaultValue": "southcentralus"
        },
        "accountSasProperties": {
            "type": "object",
            "defaultValue": {
                "signedServices": "b",
                "signedPermission": "r",
                "signedExpiry": "2018-08-20T11:00:00Z",
                "signedResourceTypes": "s"
            }
        }
    },
    "resources": [
        {
            "apiVersion": "2018-02-01",
            "name": "[parameters('storagename')]",
            "location": "[parameters('location')]",
            "type": "Microsoft.Storage/storageAccounts",
            "sku": {
                "name": "Standard_LRS"
            },
            "kind": "StorageV2",
            "properties": {
                "supportsHttpsTrafficOnly": false,
                "accessTier": "Hot",
                "encryption": {
                    "services": {
                        "blob": {
                            "enabled": true
                        },
                        "file": {
                            "enabled": true
                        }
                    },
                    "keySource": "Microsoft.Storage"
                }
            },
            "dependsOn": []
        }
    ],
    "outputs": {
        "keys": {
            "type": "object",
            "value": "[listKeys(parameters('storagename'), '2018-02-01')]"
        },
        "accountSAS": {
            "type": "object",
            "value": "[listAccountSas(parameters('storagename'), '2018-02-01',
parameters('accountSasProperties'))]"
        }
    }
}
```

providers

```
providers(providerNamespace, [resourceType])
```

Returns information about a resource provider and its supported resource types. If you don't provide a resource type, the function returns all the supported types for the resource provider.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
providerNamespace	Yes	string	Namespace of the provider
resourceType	No	string	The type of resource within the specified namespace.

Return value

Each supported type is returned in the following format:

```
{
    "resourceType": "{name of resource type}",
    "locations": [ all supported locations ],
    "apiVersions": [ all supported API versions ]
}
```

Array ordering of the returned values isn't guaranteed.

Example

The following example template shows how to use the provider function:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "providerNamespace": {
            "type": "string"
        },
        "resourceType": {
            "type": "string"
        }
    },
    "resources": [],
    "outputs": {
        "providerOutput": {
            "value": "[providers(parameters('providerNamespace'), parameters('resourceType'))]",
            "type" : "object"
        }
    }
}
```

For the **Microsoft.Web** resource provider and **sites** resource type, the preceding example returns an object in the following format:

```
{
  "resourceType": "sites",
  "locations": [
    "South Central US",
    "North Europe",
    "West Europe",
    "Southeast Asia",
    ...
  ],
  "apiVersions": [
    "2016-08-01",
    "2016-03-01",
    "2015-08-01-preview",
    "2015-08-01",
    ...
  ]
}
```

reference

`reference(resourceName or resourceIdentifier, [apiVersion], ['Full'])`

Returns an object representing a resource's runtime state.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
resourceName or resourceIdentifier	Yes	string	Name or unique identifier of a resource.
apiVersion	No	string	API version of the specified resource. Include this parameter when the resource isn't provisioned within same template. Typically, in the format, **yyyy-mm-dd**.
'Full'	No	string	Value that specifies whether to return the full resource object. If you don't specify `'Full'`, only the properties object of the resource is returned. The full object includes values such as the resource ID and location.

Return value

Every resource type returns different properties for the reference function. The function doesn't return a single, predefined format. Also, the returned value differs based on whether you specified the full object. To see the properties for a resource type, return the object in the outputs section as shown in the example.

Remarks

The reference function retrieves the runtime state of either a previously deployed resource or a resource deployed in the current template. This article shows examples for both scenarios. When referencing a resource in the current template, provide only the resource name as a parameter. When referencing a previously deployed resource, provide the resource ID and an API version for the resource. You can determine valid API versions for your resource in the template reference.

The reference function can only be used in the properties of a resource definition and the outputs section of a template or deployment. When used with property iteration, you can use the reference function for `input` because the expression is assigned to the resource property. You can't use it with `count` because the count must be determined before the reference function is resolved.

You can't use the reference function in the outputs of a nested template to return a resource you've deployed in the nested template. Instead, use a linked template.

By using the reference function, you implicitly declare that one resource depends on another resource if the referenced resource is provisioned within same template and you refer to the resource by its name (not resource ID). You don't need to also use the dependsOn property. The function isn't evaluated until the referenced resource has completed deployment.

If you use the **reference** function in a resource that is conditionally deployed, the function is evaluated even if the resource isn't deployed. You get an error if the **reference** function refers to a resource that doesn't exist. Use the **if** function to make sure the function is only evaluated when the resource is being deployed. See the if function for a sample template that uses if and reference with a conditionally deployed resource.

To see the property names and values for a resource type, create a template that returns the object in the outputs section. If you have an existing resource of that type, your template returns the object without deploying any new resources.

Typically, you use the **reference** function to return a particular value from an object, such as the blob endpoint URI or fully qualified domain name.

```
"outputs": {
    "BlobUri": {
        "value": "[reference(concat('Microsoft.Storage/storageAccounts/', parameters('storageAccountName')),
'2016-01-01').primaryEndpoints.blob]",
        "type" : "string"
    },
    "FQDN": {
        "value": "[reference(concat('Microsoft.Network/publicIPAddresses/', parameters('ipAddressName')),
'2016-03-30').dnsSettings.fqdn]",
        "type" : "string"
    }
}
```

Use `'Full'` when you need resource values that aren't part of the properties schema. For example, to set key vault access policies, get the identity properties for a virtual machine.

```
{
  "type": "Microsoft.KeyVault/vaults",
  "properties": {
    "tenantId": "[reference(concat('Microsoft.Compute/virtualMachines/', variables('vmName')), '2017-03-30',
'Full').identity.tenantId]",
    "accessPolicies": [
      {
        "tenantId": "[reference(concat('Microsoft.Compute/virtualMachines/', variables('vmName')), '2017-03-
30', 'Full').identity.tenantId]",
        "objectId": "[reference(concat('Microsoft.Compute/virtualMachines/', variables('vmName')), '2017-03-
30', 'Full').identity.principalId]",
        "permissions": {
          "keys": [
            "all"
          ],
          "secrets": [
            "all"
          ]
        }
      }
    ],
    ...
```

For the complete example of the preceding template, see Windows to Key Vault. A similar example is available for Linux.

Example

The following example template deploys a resource, and references that resource.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
      "storageAccountName": {
          "type": "string"
      }
  },
  "resources": [
    {
      "name": "[parameters('storageAccountName')]",
      "type": "Microsoft.Storage/storageAccounts",
      "apiVersion": "2016-12-01",
      "sku": {
        "name": "Standard_LRS"
      },
      "kind": "Storage",
      "location": "[resourceGroup().location]",
      "tags": {},
      "properties": {
      }
    }
  ],
  "outputs": {
      "referenceOutput": {
          "type": "object",
          "value": "[reference(parameters('storageAccountName'))]"
      },
      "fullReferenceOutput": {
        "type": "object",
        "value": "[reference(parameters('storageAccountName'), '2016-12-01', 'Full')]"
      }
    }
}
```

The preceding example returns the two objects. The properties object is in the following format:

```
{
   "creationTime": "2017-10-09T18:55:40.5863736Z",
   "primaryEndpoints": {
     "blob": "https://examplestorage.blob.core.windows.net/",
     "file": "https://examplestorage.file.core.windows.net/",
     "queue": "https://examplestorage.queue.core.windows.net/",
     "table": "https://examplestorage.table.core.windows.net/"
   },
   "primaryLocation": "southcentralus",
   "provisioningState": "Succeeded",
   "statusOfPrimary": "available",
   "supportsHttpsTrafficOnly": false
}
```

The full object is in the following format:

```
{
  "apiVersion":"2016-12-01",
  "location":"southcentralus",
  "sku": {
    "name":"Standard_LRS",
    "tier":"Standard"
  },
  "tags":{},
  "kind":"Storage",
  "properties": {
    "creationTime":"2017-10-09T18:55:40.5863736Z",
    "primaryEndpoints": {
      "blob":"https://examplestorage.blob.core.windows.net/",
      "file":"https://examplestorage.file.core.windows.net/",
      "queue":"https://examplestorage.queue.core.windows.net/",
      "table":"https://examplestorage.table.core.windows.net/"
    },
    "primaryLocation":"southcentralus",
    "provisioningState":"Succeeded",
    "statusOfPrimary":"available",
    "supportsHttpsTrafficOnly":false
  },
  "subscriptionId":"<subscription-id>",
  "resourceGroupName":"functionexamplegroup",
  "resourceId":"Microsoft.Storage/storageAccounts/examplestorage",
  "referenceApiVersion":"2016-12-01",
  "condition":true,
  "isConditionTrue":true,
  "isTemplateResource":false,
  "isAction":false,
  "provisioningOperation":"Read"
}
```

The following example template references a storage account that isn't deployed in this template. The storage account already exists within the same subscription.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "storageResourceGroup": {
            "type": "string"
        },
        "storageAccountName": {
            "type": "string"
        }
    },
    "resources": [],
    "outputs": {
        "ExistingStorage": {
            "value": "[reference(resourceId(parameters('storageResourceGroup'),
'Microsoft.Storage/storageAccounts', parameters('storageAccountName')), '2018-07-01')]",
            "type": "object"
        }
    }
}
```

resourceGroup

`resourceGroup()`

Returns an object that represents the current resource group.

Return value

The returned object is in the following format:

```
{
  "id": "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}",
  "name": "{resourceGroupName}",
  "location": "{resourceGroupLocation}",
  "tags": {
  },
  "properties": {
    "provisioningState": "{status}"
  }
}
```

Remarks

The `resourceGroup()` function can't be used in a template that is deployed at the subscription level. It can only be used in templates that are deployed to a resource group.

A common use of the resourceGroup function is to create resources in the same location as the resource group. The following example uses the resource group location to assign the location for a web site.

```
"resources": [
   {
      "apiVersion": "2016-08-01",
      "type": "Microsoft.Web/sites",
      "name": "[parameters('siteName')]",
      "location": "[resourceGroup().location]",
      ...
   }
]
```

You can also use the resourceGroup function to apply tags from the resource group to a resource. For more information, see Apply tags from resource group.

Example

The following example template returns the properties of the resource group.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "resources": [],
    "outputs": {
        "resourceGroupOutput": {
            "value": "[resourceGroup()]",
            "type" : "object"
        }
    }
}
```

The preceding example returns an object in the following format:

```
{
  "id": "/subscriptions/{subscription-id}/resourceGroups/examplegroup",
  "name": "examplegroup",
  "location": "southcentralus",
  "properties": {
    "provisioningState": "Succeeded"
  }
}
```

resourceId

`resourceId([subscriptionId], [resourceGroupName], resourceType, resourceName1, [resourceName2]...)`

Returns the unique identifier of a resource. You use this function when the resource name is ambiguous or not provisioned within the same template.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
subscriptionId	No	string (In GUID format)	Default value is the current subscription. Specify this value when you need to retrieve a resource in another subscription.
resourceGroupName	No	string	Default value is current resource group. Specify this value when you need to retrieve a resource in another resource group.
resourceType	Yes	string	Type of resource including resource provider namespace.
resourceName1	Yes	string	Name of resource.

PARAMETER	REQUIRED	TYPE	DESCRIPTION
resourceName2	No	string	Next resource name segment if resource is nested.

Return value

The identifier is returned in the following format:

```
/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/{resourceProviderNamespace}/{reso
urceType}/{resourceName}
```

Remarks

When used with a subscription-level deployment, the `resourceId()` function can only retrieve the ID of resources deployed at that level. For example, you can get the ID of a policy definition or role definition, but not the ID of a storage account. For deployments to a resource group, the opposite is true. You can't get the resource ID of resources deployed at the subscription-level.

The parameter values you specify depend on whether the resource is in the same subscription and resource group as the current deployment. To get the resource ID for a storage account in the same subscription and resource group, use:

```
"[resourceId('Microsoft.Storage/storageAccounts','examplestorage')]"
```

To get the resource ID for a storage account in the same subscription but a different resource group, use:

```
"[resourceId('otherResourceGroup', 'Microsoft.Storage/storageAccounts','examplestorage')]"
```

To get the resource ID for a storage account in a different subscription and resource group, use:

```
"[resourceId('xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx', 'otherResourceGroup',
'Microsoft.Storage/storageAccounts','examplestorage')]"
```

To get the resource ID for a database in a different resource group, use:

```
"[resourceId('otherResourceGroup', 'Microsoft.SQL/servers/databases', parameters('serverName'),
parameters('databaseName'))]"
```

To get the resource ID of a subscription-level resource when deploying at the subscription scope, use:

```
"[resourceId('Microsoft.Authorization/policyDefinitions', 'locationpolicy')]"
```

Often, you need to use this function when using a storage account or virtual network in an alternate resource group. The following example shows how a resource from an external resource group can easily be used:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
      "virtualNetworkName": {
          "type": "string"
      },
      "virtualNetworkResourceGroup": {
          "type": "string"
      },
      "subnet1Name": {
          "type": "string"
      },
      "nicName": {
          "type": "string"
      }
  },
  "variables": {
      "subnet1Ref": "[resourceId(parameters('virtualNetworkResourceGroup'),
'Microsoft.Network/virtualNetworks/subnets', parameters('virtualNetworkName'), parameters('subnet1Name'))]"
  },
  "resources": [
  {
      "apiVersion": "2015-05-01-preview",
      "type": "Microsoft.Network/networkInterfaces",
      "name": "[parameters('nicName')]",
      "location": "[parameters('location')]",
      "properties": {
          "ipConfigurations": [{
              "name": "ipconfig1",
              "properties": {
                  "privateIPAllocationMethod": "Dynamic",
                  "subnet": {
                      "id": "[variables('subnet1Ref')]"
                  }
              }
          }]
       }
  }]
}
```

Example

The following example template returns the resource ID for a storage account in the resource group:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "resources": [],
    "outputs": {
        "sameRGOutput": {
            "value": "[resourceId('Microsoft.Storage/storageAccounts','examplestorage')]",
            "type" : "string"
        },
        "differentRGOutput": {
            "value": "[resourceId('otherResourceGroup',
'Microsoft.Storage/storageAccounts','examplestorage')]",
            "type" : "string"
        },
        "differentSubOutput": {
            "value": "[resourceId('11111111-1111-1111-1111-111111111111', 'otherResourceGroup',
'Microsoft.Storage/storageAccounts','examplestorage')]",
            "type" : "string"
        },
        "nestedResourceOutput": {
            "value": "[resourceId('Microsoft.SQL/servers/databases', 'serverName', 'databaseName')]",
            "type" : "string"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
sameRGOutput	String	/subscriptions/{current-sub-id}/resourceGroups/examplegroup/providers/Microsoft.Storage/storageAccounts/examplestorage
differentRGOutput	String	/subscriptions/{current-sub-id}/resourceGroups/otherResourceGroup/providers/Microsoft.Storage/storageAccounts/examplestorage
differentSubOutput	String	/subscriptions/11111111-1111-1111-1111-111111111111/resourceGroups/otherResourceGroup/providers/Microsoft.Storage/storageAccounts/examplestorage
nestedResourceOutput	String	/subscriptions/{current-sub-id}/resourceGroups/examplegroup/providers/Microsoft.SQL/servers/serverName/databases/databaseName

subscription

`subscription()`

Returns details about the subscription for the current deployment.

Return value

The function returns the following format:

```
{
    "id": "/subscriptions/{subscription-id}",
    "subscriptionId": "{subscription-id}",
    "tenantId": "{tenant-id}",
    "displayName": "{name-of-subscription}"
}
```

Example

The following example template shows the subscription function called in the outputs section.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "resources": [],
    "outputs": {
        "subscriptionOutput": {
            "value": "[subscription()]",
            "type" : "object"
        }
    }
}
```

Next steps

- For a description of the sections in an Azure Resource Manager template, see Authoring Azure Resource Manager templates.
- To merge multiple templates, see Using linked templates with Azure Resource Manager.
- To iterate a specified number of times when creating a type of resource, see Create multiple instances of resources in Azure Resource Manager.
- To see how to deploy the template you've created, see Deploy an application with Azure Resource Manager template.

String functions for Azure Resource Manager templates

6/18/2019 • 29 minutes to read • Edit Online

Resource Manager provides the following functions for working with strings:

- base64
- base64ToJson
- base64ToString
- concat
- contains
- dataUri
- dataUriToString
- empty
- endsWith
- first
- format
- guid
- indexOf
- last
- lastIndexOf
- length
- newGuid
- padLeft
- replace
- skip
- split
- startsWith
- string
- substring
- take
- toLower
- toUpper
- trim
- uniqueString
- uri
- uriComponent
- uriComponentToString
- utcNow

base64

`base64(inputString)`

Returns the base64 representation of the input string.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
inputString	Yes	string	The value to return as a base64 representation.

Return value

A string containing the base64 representation.

Examples

The following example template shows how to use the base64 function.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "stringData": {
            "type": "string",
            "defaultValue": "one, two, three"
        },
        "jsonFormattedData": {
            "type": "string",
            "defaultValue": "{'one': 'a', 'two': 'b'}"
        }
    },
    "variables": {
        "base64String": "[base64(parameters('stringData'))]",
        "base64Object": "[base64(parameters('jsonFormattedData'))]"
    },
    "resources": [
    ],
    "outputs": {
        "base64Output": {
            "type": "string",
            "value": "[variables('base64String')]"
        },
        "toStringOutput": {
            "type": "string",
            "value": "[base64ToString(variables('base64String'))]"
        },
        "toJsonOutput": {
            "type": "object",
            "value": "[base64ToJson(variables('base64Object'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
base64Output	String	b25lLCB0d28sIHRocmVl
toStringOutput	String	one, two, three
toJsonOutput	Object	{"one": "a", "two": "b"}

base64ToJson

`base64tojson`

Converts a base64 representation to a JSON object.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
base64Value	Yes	string	The base64 representation to convert to a JSON object.

Return value

A JSON object.

Examples

The following example template uses the base64ToJson function to convert a base64 value:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "stringData": {
            "type": "string",
            "defaultValue": "one, two, three"
        },
        "jsonFormattedData": {
            "type": "string",
            "defaultValue": "{'one': 'a', 'two': 'b'}"
        }
    },
    "variables": {
        "base64String": "[base64(parameters('stringData'))]",
        "base64Object": "[base64(parameters('jsonFormattedData'))]"
    },
    "resources": [
    ],
    "outputs": {
        "base64Output": {
            "type": "string",
            "value": "[variables('base64String')]"
        },
        "toStringOutput": {
            "type": "string",
            "value": "[base64ToString(variables('base64String'))]"
        },
        "toJsonOutput": {
            "type": "object",
            "value": "[base64ToJson(variables('base64Object'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
base64Output	String	b25lLCB0d28sIHRocmVl
toStringOutput	String	one, two, three
toJsonOutput	Object	{"one": "a", "two": "b"}

base64ToString

`base64ToString(base64Value)`

Converts a base64 representation to a string.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
base64Value	Yes	string	The base64 representation to convert to a string.

Return value

A string of the converted base64 value.

Examples

The following example template uses the base64ToString function to convert a base64 value:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "stringData": {
            "type": "string",
            "defaultValue": "one, two, three"
        },
        "jsonFormattedData": {
            "type": "string",
            "defaultValue": "{'one': 'a', 'two': 'b'}"
        }
    },
    "variables": {
        "base64String": "[base64(parameters('stringData'))]",
        "base64Object": "[base64(parameters('jsonFormattedData'))]"
    },
    "resources": [
    ],
    "outputs": {
        "base64Output": {
            "type": "string",
            "value": "[variables('base64String')]"
        },
        "toStringOutput": {
            "type": "string",
            "value": "[base64ToString(variables('base64String'))]"
        },
        "toJsonOutput": {
            "type": "object",
            "value": "[base64ToJson(variables('base64Object'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
base64Output	String	b25lLCB0d28sIHRocmVl
toStringOutput	String	one, two, three

NAME	TYPE	VALUE
toJsonOutput	Object	{"one": "a", "two": "b"}

concat

`concat (arg1, arg2, arg3, ...)`

Combines multiple string values and returns the concatenated string, or combines multiple arrays and returns the concatenated array.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	string or array	The first value for concatenation.
additional arguments	No	string	Additional values in sequential order for concatenation.

Return value

A string or array of concatenated values.

Examples

The following example template shows how to combine two string values and return a concatenated string.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "prefix": {
            "type": "string",
            "defaultValue": "prefix"
        }
    },
    "resources": [],
    "outputs": {
        "concatOutput": {
            "value": "[concat(parameters('prefix'), '-', uniqueString(resourceGroup().id))]",
            "type" : "string"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
concatOutput	String	prefix-5yj4yjf5mbg72

The following example template shows how to combine two arrays.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "firstArray": {
            "type": "array",
            "defaultValue": [
                "1-1",
                "1-2",
                "1-3"
            ]
        },
        "secondArray": {
            "type": "array",
            "defaultValue": [
                "2-1",
                "2-2",
                "2-3"
            ]
        }
    },
    "resources": [
    ],
    "outputs": {
        "return": {
            "type": "array",
            "value": "[concat(parameters('firstArray'), parameters('secondArray'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
return	Array	["1-1", "1-2", "1-3", "2-1", "2-2", "2-3"]

contains

`contains (container, itemToFind)`

Checks whether an array contains a value, an object contains a key, or a string contains a substring. The string comparison is case-sensitive. However, when testing if an object contains a key, the comparison is case-insensitive.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
container	Yes	array, object, or string	The value that contains the value to find.
itemToFind	Yes	string or int	The value to find.

Return value

True if the item is found; otherwise, **False**.

Examples

The following example template shows how to use contains with different types:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "stringToTest": {
            "type": "string",
            "defaultValue": "OneTwoThree"
        },
        "objectToTest": {
            "type": "object",
            "defaultValue": {"one": "a", "two": "b", "three": "c"}
        },
        "arrayToTest": {
            "type": "array",
            "defaultValue": ["one", "two", "three"]
        }
    },
    "resources": [
    ],
    "outputs": {
        "stringTrue": {
            "type": "bool",
            "value": "[contains(parameters('stringToTest'), 'e')]"
        },
        "stringFalse": {
            "type": "bool",
            "value": "[contains(parameters('stringToTest'), 'z')]"
        },
        "objectTrue": {
            "type": "bool",
            "value": "[contains(parameters('objectToTest'), 'one')]"
        },
        "objectFalse": {
            "type": "bool",
            "value": "[contains(parameters('objectToTest'), 'a')]"
        },
        "arrayTrue": {
            "type": "bool",
            "value": "[contains(parameters('arrayToTest'), 'three')]"
        },
        "arrayFalse": {
            "type": "bool",
            "value": "[contains(parameters('arrayToTest'), 'four')]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
stringTrue	Bool	True
stringFalse	Bool	False
objectTrue	Bool	True
objectFalse	Bool	False
arrayTrue	Bool	True
arrayFalse	Bool	False

dataUri

`dataUri(stringToConvert)`

Converts a value to a data URI.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
stringToConvert	Yes	string	The value to convert to a data URI.

Return value

A string formatted as a data URI.

Examples

The following example template converts a value to a data URI, and converts a data URI to a string:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "stringToTest": {
            "type": "string",
            "defaultValue": "Hello"
        },
        "dataFormattedString": {
            "type": "string",
            "defaultValue": "data:;base64,SGVsbG8sIFdvcmxkIQ=="
        }
    },
    "resources": [],
    "outputs": {
        "dataUriOutput": {
            "value": "[dataUri(parameters('stringToTest'))]",
            "type" : "string"
        },
        "toStringOutput": {
            "type": "string",
            "value": "[dataUriToString(parameters('dataFormattedString'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
dataUriOutput	String	data:text/plain;charset=utf8;base64,SGVsbG8=
toStringOutput	String	Hello, World!

dataUriToString

`dataUriToString(dataUriToConvert)`

Converts a data URI formatted value to a string.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
dataUriToConvert	Yes	string	The data URI value to convert.

Return value

A string containing the converted value.

Examples

The following example template converts a value to a data URI, and converts a data URI to a string:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "stringToTest": {
            "type": "string",
            "defaultValue": "Hello"
        },
        "dataFormattedString": {
            "type": "string",
            "defaultValue": "data:;base64,SGVsbG8sIFdvcmxkIQ=="
        }
    },
    "resources": [],
    "outputs": {
        "dataUriOutput": {
            "value": "[dataUri(parameters('stringToTest'))]",
            "type" : "string"
        },
        "toStringOutput": {
            "type": "string",
            "value": "[dataUriToString(parameters('dataFormattedString'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
dataUriOutput	String	data:text/plain;charset=utf8;base64,SGVsbG8=
toStringOutput	String	Hello, World!

empty

`empty(itemToTest)`

Determines if an array, object, or string is empty.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION

PARAMETER	REQUIRED	TYPE	DESCRIPTION
itemToTest	Yes	array, object, or string	The value to check if it's empty.

Return value

Returns **True** if the value is empty; otherwise, **False**.

Examples

The following example template checks whether an array, object, and string are empty.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "testArray": {
            "type": "array",
            "defaultValue": []
        },
        "testObject": {
            "type": "object",
            "defaultValue": {}
        },
        "testString": {
            "type": "string",
            "defaultValue": ""
        }
    },
    "resources": [
    ],
    "outputs": {
        "arrayEmpty": {
            "type": "bool",
            "value": "[empty(parameters('testArray'))]"
        },
        "objectEmpty": {
            "type": "bool",
            "value": "[empty(parameters('testObject'))]"
        },
        "stringEmpty": {
            "type": "bool",
            "value": "[empty(parameters('testString'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayEmpty	Bool	True
objectEmpty	Bool	True
stringEmpty	Bool	True

endsWith

`endsWith(stringToSearch, stringToFind)`

Determines whether a string ends with a value. The comparison is case-insensitive.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
stringToSearch	Yes	string	The value that contains the item to find.
stringToFind	Yes	string	The value to find.

Return value

True if the last character or characters of the string match the value; otherwise, **False**.

Examples

The following example template shows how to use the startsWith and endsWith functions:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "resources": [],
    "outputs": {
        "startsTrue": {
            "value": "[startsWith('abcdef', 'ab')]",
            "type" : "bool"
        },
        "startsCapTrue": {
            "value": "[startsWith('abcdef', 'A')]",
            "type" : "bool"
        },
        "startsFalse": {
            "value": "[startsWith('abcdef', 'e')]",
            "type" : "bool"
        },
        "endsTrue": {
            "value": "[endsWith('abcdef', 'ef')]",
            "type" : "bool"
        },
        "endsCapTrue": {
            "value": "[endsWith('abcdef', 'F')]",
            "type" : "bool"
        },
        "endsFalse": {
            "value": "[endsWith('abcdef', 'e')]",
            "type" : "bool"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
startsTrue	Bool	True
startsCapTrue	Bool	True
startsFalse	Bool	False
endsTrue	Bool	True

NAME	TYPE	VALUE
endsCapTrue	Bool	True
endsFalse	Bool	False

first

`first(arg1)`

Returns the first character of the string, or first element of the array.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	array or string	The value to retrieve the first element or character.

Return value

A string of the first character, or the type (string, int, array, or object) of the first element in an array.

Examples

The following example template shows how to use the first function with an array and string.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "arrayToTest": {
            "type": "array",
            "defaultValue": ["one", "two", "three"]
        }
    },
    "resources": [
    ],
    "outputs": {
        "arrayOutput": {
            "type": "string",
            "value": "[first(parameters('arrayToTest'))]"
        },
        "stringOutput": {
            "type": "string",
            "value": "[first('One Two Three')]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayOutput	String	one
stringOutput	String	O

format

`format(formatString, arg1, arg2, ...)`

Creates a formatted string from input values.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
formatString	Yes	string	The composite format string.
arg1	Yes	string, integer, or boolean	The value to include in the formatted string.
additional arguments	No	string, integer, or boolean	Additional values to include in the formatted string.

Remarks

Use this function to format a string in your template. It uses the same formatting options as the System.String.Format method in .NET.

Examples

The following example template shows how to use the format function.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "greeting": {
            "type": "string",
            "defaultValue": "Hello"
        },
        "name": {
            "type": "string",
            "defaultValue": "User"
        },
        "numberToFormat": {
            "type": "int",
            "defaultValue": 8175133
        }
    },
    "resources": [
    ],
    "outputs": {
        "formatTest": {
            "type": "string",
            "value": "[format('{0}, {1}. Formatted number: {2:N0}', parameters('greeting'), parameters('name'),
parameters('numberToFormat'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
formatTest	String	Hello, User. Formatted number: 8,175,133

guid

`guid(baseString, ...)`

Creates a value in the format of a globally unique identifier based on the values provided as parameters.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
baseString	Yes	string	The value used in the hash function to create the GUID.
additional parameters as needed	No	string	You can add as many strings as needed to create the value that specifies the level of uniqueness.

Remarks

This function is helpful when you need to create a value in the format of a globally unique identifier. You provide parameter values that limit the scope of uniqueness for the result. You can specify whether the name is unique down to subscription, resource group, or deployment.

The returned value isn't a random string, but rather the result of a hash function on the parameters. The returned value is 36 characters long. It isn't globally unique. To create a new GUID that isn't based on that hash value of the parameters, use the newGuid function.

The following examples show how to use guid to create a unique value for commonly used levels.

Unique scoped to subscription

```
"[guid(subscription().subscriptionId)]"
```

Unique scoped to resource group

```
"[guid(resourceGroup().id)]"
```

Unique scoped to deployment for a resource group

```
"[guid(resourceGroup().id, deployment().name)]"
```

Return value

A string containing 36 characters in the format of a globally unique identifier.

Examples

The following example template returns results from guid:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {},
    "variables": {},
    "resources": [],
    "outputs": {
        "guidPerSubscription": {
            "value": "[guid(subscription().subscriptionId)]",
            "type": "string"
        },
        "guidPerResourceGroup": {
            "value": "[guid(resourceGroup().id)]",
            "type": "string"
        },
        "guidPerDeployment": {
            "value": "[guid(resourceGroup().id, deployment().name)]",
            "type": "string"
        }
    }
}
```

indexOf

`indexOf(stringToSearch, stringToFind)`

Returns the first position of a value within a string. The comparison is case-insensitive.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
stringToSearch	Yes	string	The value that contains the item to find.
stringToFind	Yes	string	The value to find.

Return value

An integer that represents the position of the item to find. The value is zero-based. If the item isn't found, -1 is returned.

Examples

The following example template shows how to use the indexOf and lastIndexOf functions:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "resources": [],
    "outputs": {
        "firstT": {
            "value": "[indexOf('test', 't')]",
            "type" : "int"
        },
        "lastT": {
            "value": "[lastIndexOf('test', 't')]",
            "type" : "int"
        },
        "firstString": {
            "value": "[indexOf('abcdef', 'CD')]",
            "type" : "int"
        },
        "lastString": {
            "value": "[lastIndexOf('abcdef', 'AB')]",
            "type" : "int"
        },
        "notFound": {
            "value": "[indexOf('abcdef', 'z')]",
            "type" : "int"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
firstT	Int	0
lastT	Int	3
firstString	Int	2
lastString	Int	0
notFound	Int	-1

last

`last (arg1)`

Returns last character of the string, or the last element of the array.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	array or string	The value to retrieve the last element or character.

Return value

A string of the last character, or the type (string, int, array, or object) of the last element in an array.

Examples

The following example template shows how to use the last function with an array and string.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "arrayToTest": {
            "type": "array",
            "defaultValue": ["one", "two", "three"]
        }
    },
    "resources": [
    ],
    "outputs": {
        "arrayOutput": {
            "type": "string",
            "value": "[last(parameters('arrayToTest'))]"
        },
        "stringOutput": {
            "type": "string",
            "value": "[last('One Two Three')]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayOutput	String	three
stringOutput	String	e

lastIndexOf

`lastIndexOf(stringToSearch, stringToFind)`

Returns the last position of a value within a string. The comparison is case-insensitive.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
stringToSearch	Yes	string	The value that contains the item to find.
stringToFind	Yes	string	The value to find.

Return value

An integer that represents the last position of the item to find. The value is zero-based. If the item isn't found, -1 is returned.

Examples

The following example template shows how to use the indexOf and lastIndexOf functions:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "resources": [],
    "outputs": {
        "firstT": {
            "value": "[indexOf('test', 't')]",
            "type" : "int"
        },
        "lastT": {
            "value": "[lastIndexOf('test', 't')]",
            "type" : "int"
        },
        "firstString": {
            "value": "[indexOf('abcdef', 'CD')]",
            "type" : "int"
        },
        "lastString": {
            "value": "[lastIndexOf('abcdef', 'AB')]",
            "type" : "int"
        },
        "notFound": {
            "value": "[indexOf('abcdef', 'z')]",
            "type" : "int"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
firstT	Int	0
lastT	Int	3
firstString	Int	2
lastString	Int	0
notFound	Int	-1

length

`length(string)`

Returns the number of characters in a string, or elements in an array.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	array or string	The array to use for getting the number of elements, or the string to use for getting the number of characters.

Return value

An int.

Examples

The following example template shows how to use length with an array and string:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "arrayToTest": {
            "type": "array",
            "defaultValue": [
                "one",
                "two",
                "three"
            ]
        },
        "stringToTest": {
            "type": "string",
            "defaultValue": "One Two Three"
        }
    },
    "resources": [],
    "outputs": {
        "arrayLength": {
            "type": "int",
            "value": "[length(parameters('arrayToTest'))]"
        },
        "stringLength": {
            "type": "int",
            "value": "[length(parameters('stringToTest'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayLength	Int	3
stringLength	Int	13

newGuid

`newGuid()`

Returns a value in the format of a globally unique identifier. **This function can only be used in the default value for a parameter.**

Remarks

You can only use this function within an expression for the default value of a parameter. Using this function anywhere else in a template returns an error. The function isn't allowed in other parts of the template because it returns a different value each time it's called. Deploying the same template with the same parameters wouldn't reliably produce the same results.

The newGuid function differs from the guid function because it doesn't take any parameters. When you call guid with the same parameter, it returns the same identifier each time. Use guid when you need to reliably generate the same GUID for a specific environment. Use newGuid when you need a different identifier each time, such as deploying resources to a test environment.

If you use the option to redeploy an earlier successful deployment, and the earlier deployment includes a parameter that uses newGuid, the parameter isn't reevaluated. Instead, the parameter value from the earlier deployment is automatically reused in the rollback deployment.

In a test environment, you may need to repeatedly deploy resources that only live for a short time. Rather than constructing unique names, you can use newGuid with uniqueString to create unique names.

Be careful redeploying a template that relies on the newGuid function for a default value. When you redeploy and don't provide a value for the parameter, the function is reevaluated. If you want to update an existing resource rather than create a new one, pass in the parameter value from the earlier deployment.

Return value

A string containing 36 characters in the format of a globally unique identifier.

Examples

The following example template shows a parameter with a new identifier.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "guidValue": {
            "type": "string",
            "defaultValue": "[newGuid()]"
        }
    },
    "resources": [
    ],
    "outputs": {
        "guidOutput": {
            "type": "string",
            "value": "[parameters('guidValue')]"
        }
    }
}
```

The output from the preceding example varies for each deployment but will be similar to:

NAME	TYPE	VALUE
guidOutput	string	b76a51fc-bd72-4a77-b9a2-3c29e7d2e551

The following example uses the newGuid function to create a unique name for a storage account. This template might work for test environment where the storage account exists for a short time and isn't redeployed.

```json
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "guidValue": {
            "type": "string",
            "defaultValue": "[newGuid()]"
        }
    },
    "variables": {
        "storageName": "[concat('storage', uniqueString(parameters('guidValue')))]"
    },
    "resources": [
        {
            "type": "Microsoft.Storage/storageAccounts",
            "name": "[variables('storageName')]",
            "location": "West US",
            "apiVersion": "2018-07-01",
            "sku":{
                "name": "Standard_LRS"
            },
            "kind": "StorageV2",
            "properties": {}
        }
    ],
    "outputs": {
        "nameOutput": {
            "type": "string",
            "value": "[variables('storageName')]"
        }
    }
}
```

The output from the preceding example varies for each deployment but will be similar to:

NAME	TYPE	VALUE
nameOutput	string	storagenziwvyru7uxie

padLeft

`padLeft(valueToPad, totalLength, paddingCharacter)`

Returns a right-aligned string by adding characters to the left until reaching the total specified length.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
valueToPad	Yes	string or int	The value to right-align.
totalLength	Yes	int	The total number of characters in the returned string.
paddingCharacter	No	single character	The character to use for left-padding until the total length is reached. The default value is a space.

If the original string is longer than the number of characters to pad, no characters are added.

Return value

A string with at least the number of specified characters.

Examples

The following example template shows how to pad the user-provided parameter value by adding the zero character until it reaches the total number of characters.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "testString": {
            "type": "string",
            "defaultValue": "123"
        }
    },
    "resources": [],
    "outputs": {
        "stringOutput": {
            "type": "string",
            "value": "[padLeft(parameters('testString'),10,'0')]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
stringOutput	String	0000000123

replace

`replace(originalString, oldString, newString)`

Returns a new string with all instances of one string replaced by another string.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
originalString	Yes	string	The value that has all instances of one string replaced by another string.
oldString	Yes	string	The string to be removed from the original string.
newString	Yes	string	The string to add in place of the removed string.

Return value

A string with the replaced characters.

Examples

The following example template shows how to remove all dashes from the user-provided string, and how to replace part of the string with another string.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "testString": {
            "type": "string",
            "defaultValue": "123-123-1234"
        }
    },
    "resources": [],
    "outputs": {
        "firstOutput": {
            "type": "string",
            "value": "[replace(parameters('testString'),'-', '')]"
        },
        "secondOutput": {
            "type": "string",
            "value": "[replace(parameters('testString'),'1234', 'xxxx')]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
firstOutput	String	1231231234
secondOutput	String	123-123-xxxx

skip

`skip(originalValue, numberToSkip)`

Returns a string with all the characters after the specified number of characters, or an array with all the elements after the specified number of elements.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
originalValue	Yes	array or string	The array or string to use for skipping.
numberToSkip	Yes	int	The number of elements or characters to skip. If this value is 0 or less, all the elements or characters in the value are returned. If it's larger than the length of the array or string, an empty array or string is returned.

Return value

An array or string.

Examples

The following example template skips the specified number of elements in the array, and the specified number of characters in a string.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "testArray": {
            "type": "array",
            "defaultValue": [
                "one",
                "two",
                "three"
            ]
        },
        "elementsToSkip": {
            "type": "int",
            "defaultValue": 2
        },
        "testString": {
            "type": "string",
            "defaultValue": "one two three"
        },
        "charactersToSkip": {
            "type": "int",
            "defaultValue": 4
        }
    },
    "resources": [],
    "outputs": {
        "arrayOutput": {
            "type": "array",
            "value": "[skip(parameters('testArray'),parameters('elementsToSkip'))]"
        },
        "stringOutput": {
            "type": "string",
            "value": "[skip(parameters('testString'),parameters('charactersToSkip'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayOutput	Array	["three"]
stringOutput	String	two three

split

`split(inputString, delimiter)`

Returns an array of strings that contains the substrings of the input string that are delimited by the specified delimiters.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
inputString	Yes	string	The string to split.
delimiter	Yes	string or array of strings	The delimiter to use for splitting the string.

Return value

An array of strings.

Examples

The following example template splits the input string with a comma, and with either a comma or a semi-colon.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "firstString": {
            "type": "string",
            "defaultValue": "one,two,three"
        },
        "secondString": {
            "type": "string",
            "defaultValue": "one;two,three"
        }
    },
    "variables": {
        "delimiters": [ ",", ";" ]
    },
    "resources": [],
    "outputs": {
        "firstOutput": {
            "type": "array",
            "value": "[split(parameters('firstString'),',')]"
        },
        "secondOutput": {
            "type": "array",
            "value": "[split(parameters('secondString'),variables('delimiters'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
firstOutput	Array	["one", "two", "three"]
secondOutput	Array	["one", "two", "three"]

startsWith

`startsWith(stringToSearch, stringToFind)`

Determines whether a string starts with a value. The comparison is case-insensitive.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
stringToSearch	Yes	string	The value that contains the item to find.
stringToFind	Yes	string	The value to find.

Return value

True if the first character or characters of the string match the value; otherwise, **False**.

Examples

The following example template shows how to use the startsWith and endsWith functions:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "resources": [],
    "outputs": {
        "startsTrue": {
            "value": "[startsWith('abcdef', 'ab')]",
            "type" : "bool"
        },
        "startsCapTrue": {
            "value": "[startsWith('abcdef', 'A')]",
            "type" : "bool"
        },
        "startsFalse": {
            "value": "[startsWith('abcdef', 'e')]",
            "type" : "bool"
        },
        "endsTrue": {
            "value": "[endsWith('abcdef', 'ef')]",
            "type" : "bool"
        },
        "endsCapTrue": {
            "value": "[endsWith('abcdef', 'F')]",
            "type" : "bool"
        },
        "endsFalse": {
            "value": "[endsWith('abcdef', 'e')]",
            "type" : "bool"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
startsTrue	Bool	True
startsCapTrue	Bool	True
startsFalse	Bool	False
endsTrue	Bool	True
endsCapTrue	Bool	True
endsFalse	Bool	False

string

`string(valueToConvert)`

Converts the specified value to a string.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
valueToConvert	Yes	Any	The value to convert to string. Any type of value can be converted, including objects and arrays.

Return value

A string of the converted value.

Examples

The following example template shows how to convert different types of values to strings:

```json
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "testObject": {
            "type": "object",
            "defaultValue": {
                "valueA": 10,
                "valueB": "Example Text"
            }
        },
        "testArray": {
            "type": "array",
            "defaultValue": [
                "a",
                "b",
                "c"
            ]
        },
        "testInt": {
            "type": "int",
            "defaultValue": 5
        }
    },
    "resources": [],
    "outputs": {
        "objectOutput": {
            "type": "string",
            "value": "[string(parameters('testObject'))]"
        },
        "arrayOutput": {
            "type": "string",
            "value": "[string(parameters('testArray'))]"
        },
        "intOutput": {
            "type": "string",
            "value": "[string(parameters('testInt'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
objectOutput	String	{"valueA":10,"valueB":"Example Text"}
arrayOutput	String	["a","b","c"]

NAME	TYPE	VALUE
intOutput	String	5

substring

`substring(stringToParse, startIndex, length)`

Returns a substring that starts at the specified character position and contains the specified number of characters.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
stringToParse	Yes	string	The original string from which the substring is extracted.
startIndex	No	int	The zero-based starting character position for the substring.
length	No	int	The number of characters for the substring. Must refer to a location within the string. Must be zero or greater.

Return value

The substring. Or, an empty string if the length is zero.

Remarks

The function fails when the substring extends beyond the end of the string, or when length is less than zero. The following example fails with the error "The index and length parameters must refer to a location within the string. The index parameter: '0', the length parameter: '11', the length of the string parameter: '10'.".

```
"parameters": {
    "inputString": { "type": "string", "value": "1234567890" }
},
"variables": {
    "prefix": "[substring(parameters('inputString'), 0, 11)]"
}
```

Examples

The following example template extracts a substring from a parameter.

```
{
 "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
 "contentVersion": "1.0.0.0",
 "parameters": {
  "testString": {
   "type": "string",
   "defaultValue": "one two three"
  }
 },
 "resources": [],
 "outputs": {
  "substringOutput": {
   "value": "[substring(parameters('testString'), 4, 3)]",
   "type": "string"
  }
 }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
substringOutput	String	two

take

`take(originalValue, numberToTake)`

Returns a string with the specified number of characters from the start of the string, or an array with the specified number of elements from the start of the array.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
originalValue	Yes	array or string	The array or string to take the elements from.
numberToTake	Yes	int	The number of elements or characters to take. If this value is 0 or less, an empty array or string is returned. If it's larger than the length of the given array or string, all the elements in the array or string are returned.

Return value

An array or string.

Examples

The following example template takes the specified number of elements from the array, and characters from a string.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "testArray": {
            "type": "array",
            "defaultValue": [
                "one",
                "two",
                "three"
            ]
        },
        "elementsToTake": {
            "type": "int",
            "defaultValue": 2
        },
        "testString": {
            "type": "string",
            "defaultValue": "one two three"
        },
        "charactersToTake": {
            "type": "int",
            "defaultValue": 2
        }
    },
    "resources": [],
    "outputs": {
        "arrayOutput": {
            "type": "array",
            "value": "[take(parameters('testArray'),parameters('elementsToTake'))]"
        },
        "stringOutput": {
            "type": "string",
            "value": "[take(parameters('testString'),parameters('charactersToTake'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayOutput	Array	["one", "two"]
stringOutput	String	on

toLower

`toLower(stringToChange)`

Converts the specified string to lower case.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
stringToChange	Yes	string	The value to convert to lower case.

Return value

The string converted to lower case.

Examples

The following example template converts a parameter value to lower case and to upper case.

```
{
 "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
 "contentVersion": "1.0.0.0",
 "parameters": {
  "testString": {
   "type": "string",
   "defaultValue": "One Two Three"
  }
 },
 "resources": [],
 "outputs": {
  "toLowerOutput": {
   "value": "[toLower(parameters('testString'))]",
   "type": "string"
  },
        "toUpperOutput": {
            "type": "string",
            "value": "[toUpper(parameters('testString'))]"
        }
 }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
toLowerOutput	String	one two three
toUpperOutput	String	ONE TWO THREE

toUpper

`toUpper(stringToChange)`

Converts the specified string to upper case.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
stringToChange	Yes	string	The value to convert to upper case.

Return value

The string converted to upper case.

Examples

The following example template converts a parameter value to lower case and to upper case.

```
{
 "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
 "contentVersion": "1.0.0.0",
 "parameters": {
  "testString": {
   "type": "string",
   "defaultValue": "One Two Three"
  }
 },
 "resources": [],
 "outputs": {
  "toLowerOutput": {
   "value": "[toLower(parameters('testString'))]",
   "type": "string"
  },
        "toUpperOutput": {
            "type": "string",
            "value": "[toUpper(parameters('testString'))]"
        }
 }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
toLowerOutput	String	one two three
toUpperOutput	String	ONE TWO THREE

trim

`trim (stringToTrim)`

Removes all leading and trailing white-space characters from the specified string.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
stringToTrim	Yes	string	The value to trim.

Return value

The string without leading and trailing white-space characters.

Examples

The following example template trims the white-space characters from the parameter.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "testString": {
            "type": "string",
            "defaultValue": "    one two three   "
        }
    },
    "resources": [],
    "outputs": {
        "return": {
            "type": "string",
            "value": "[trim(parameters('testString'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
return	String	one two three

uniqueString

`uniqueString (baseString, ...)`

Creates a deterministic hash string based on the values provided as parameters.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
baseString	Yes	string	The value used in the hash function to create a unique string.
additional parameters as needed	No	string	You can add as many strings as needed to create the value that specifies the level of uniqueness.

Remarks

This function is helpful when you need to create a unique name for a resource. You provide parameter values that limit the scope of uniqueness for the result. You can specify whether the name is unique down to subscription, resource group, or deployment.

The returned value isn't a random string, but rather the result of a hash function. The returned value is 13 characters long. It isn't globally unique. You may want to combine the value with a prefix from your naming convention to create a name that is meaningful. The following example shows the format of the returned value. The actual value varies by the provided parameters.

```
tcvhiyu5h2o5o
```

The following examples show how to use uniqueString to create a unique value for commonly used levels.

Unique scoped to subscription

```
"[uniqueString(subscription().subscriptionId)]"
```

Unique scoped to resource group

```
"[uniqueString(resourceGroup().id)]"
```

Unique scoped to deployment for a resource group

```
"[uniqueString(resourceGroup().id, deployment().name)]"
```

The following example shows how to create a unique name for a storage account based on your resource group. Inside the resource group, the name isn't unique if constructed the same way.

```
"resources": [{
    "name": "[concat('storage', uniqueString(resourceGroup().id))]",
    "type": "Microsoft.Storage/storageAccounts",
    ...
```

If you need to create a new unique name each time you deploy a template, and don't intend to update the resource, you can use the utcNow function with uniqueString. You could use this approach in a test environment. For an example, see utcNow.

Return value

A string containing 13 characters.

Examples

The following example template returns results from uniquestring:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "resources": [],
    "outputs": {
        "uniqueRG": {
            "value": "[uniqueString(resourceGroup().id)]",
            "type" : "string"
        },
        "uniqueDeploy": {
            "value": "[uniqueString(resourceGroup().id, deployment().name)]",
            "type" : "string"
        }
    }
}
```

uri

`uri (baseUri, relativeUri)`

Creates an absolute URI by combining the baseUri and the relativeUri string.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
baseUri	Yes	string	The base uri string.
relativeUri	Yes	string	The relative uri string to add to the base uri string.

The value for the **baseUri** parameter can include a specific file, but only the base path is used when constructing the URI. For example, passing `http://contoso.com/resources/azuredeploy.json` as the baseUri parameter results in a base URI of `http://contoso.com/resources/`.

Return value

A string representing the absolute URI for the base and relative values.

Examples

The following example shows how to construct a link to a nested template based on the value of the parent template.

```
"templateLink": "[uri(deployment().properties.templateLink.uri, 'nested/azuredeploy.json')]"
```

The following example template shows how to use uri, uriComponent, and uriComponentToString:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "variables": {
        "uriFormat": "[uri('http://contoso.com/resources/', 'nested/azuredeploy.json')]",
        "uriEncoded": "[uriComponent(variables('uriFormat'))]"
    },
    "resources": [
    ],
    "outputs": {
        "uriOutput": {
            "type": "string",
            "value": "[variables('uriFormat')]"
        },
        "componentOutput": {
            "type": "string",
            "value": "[variables('uriEncoded')]"
        },
        "toStringOutput": {
            "type": "string",
            "value": "[uriComponentToString(variables('uriEncoded'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
uriOutput	String	http://contoso.com/resources/nested/azuredeploy.json
componentOutput	String	http%3A%2F%2Fcontoso.com%2Fresources%2Fnested%2Fazuredeploy.json

NAME	TYPE	VALUE
toStringOutput	String	http://contoso.com/resources/nested/azuredeploy.json

uriComponent

`uricomponent(stringToEncode)`

Encodes a URI.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
stringToEncode	Yes	string	The value to encode.

Return value

A string of the URI encoded value.

Examples

The following example template shows how to use uri, uriComponent, and uriComponentToString:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "variables": {
        "uriFormat": "[uri('http://contoso.com/resources/', 'nested/azuredeploy.json')]",
        "uriEncoded": "[uriComponent(variables('uriFormat'))]"
    },
    "resources": [
    ],
    "outputs": {
        "uriOutput": {
            "type": "string",
            "value": "[variables('uriFormat')]"
        },
        "componentOutput": {
            "type": "string",
            "value": "[variables('uriEncoded')]"
        },
        "toStringOutput": {
            "type": "string",
            "value": "[uriComponentToString(variables('uriEncoded'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
uriOutput	String	http://contoso.com/resources/nested/azuredeploy.json
componentOutput	String	http%3A%2F%2Fcontoso.com%2Fresources%2Fnested%2Fazuredeploy.json

NAME	TYPE	VALUE
toStringOutput	String	http://contoso.com/resources/nested/azuredeploy.json

uriComponentToString

`uriComponentToString(uriEncodedString)`

Returns a string of a URI encoded value.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
uriEncodedString	Yes	string	The URI encoded value to convert to a string.

Return value

A decoded string of URI encoded value.

Examples

The following example template shows how to use uri, uriComponent, and uriComponentToString:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "variables": {
        "uriFormat": "[uri('http://contoso.com/resources/', 'nested/azuredeploy.json')]",
        "uriEncoded": "[uriComponent(variables('uriFormat'))]"
    },
    "resources": [
    ],
    "outputs": {
        "uriOutput": {
            "type": "string",
            "value": "[variables('uriFormat')]"
        },
        "componentOutput": {
            "type": "string",
            "value": "[variables('uriEncoded')]"
        },
        "toStringOutput": {
            "type": "string",
            "value": "[uriComponentToString(variables('uriEncoded'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
uriOutput	String	http://contoso.com/resources/nested/azuredeploy.json
componentOutput	String	http%3A%2F%2Fcontoso.com%2Fresources%2Fnested%2Fazuredeploy.json

NAME	TYPE	VALUE
toStringOutput	String	http://contoso.com/resources/nested/azuredeploy.json

utcNow

`utcNow(format)`

Returns the current (UTC) datetime value in the specified format. If no format is provided, the ISO 8601 (yyyyMMddTHHmmssZ) format is used. **This function can only be used in the default value for a parameter.**

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
format	No	string	The URI encoded value to convert to a string. Use either standard format strings or custom format strings.

Remarks

You can only use this function within an expression for the default value of a parameter. Using this function anywhere else in a template returns an error. The function isn't allowed in other parts of the template because it returns a different value each time it's called. Deploying the same template with the same parameters wouldn't reliably produce the same results.

If you use the option to redeploy an earlier successful deployment, and the earlier deployment includes a parameter that uses utcNow, the parameter isn't reevaluated. Instead, the parameter value from the earlier deployment is automatically reused in the rollback deployment.

Be careful redeploying a template that relies on the utcNow function for a default value. When you redeploy and don't provide a value for the parameter, the function is reevaluated. If you want to update an existing resource rather than create a new one, pass in the parameter value from the earlier deployment.

Return value

The current UTC datetime value.

Examples

The following example template shows different formats for the datetime value.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "utcValue": {
            "type": "string",
            "defaultValue": "[utcNow()]"
        },
        "utcShortValue": {
            "type": "string",
            "defaultValue": "[utcNow('d')]"
        },
        "utcCustomValue": {
            "type": "string",
            "defaultValue": "[utcNow('M d')]"
        }
    },
    "resources": [
    ],
    "outputs": {
        "utcOutput": {
            "type": "string",
            "value": "[parameters('utcValue')]"
        },
        "utcShortOutput": {
            "type": "string",
            "value": "[parameters('utcShortValue')]"
        },
        "utcCustomOutput": {
            "type": "string",
            "value": "[parameters('utcCustomValue')]"
        }
    }
}
```

The output from the preceding example varies for each deployment but will be similar to:

NAME	TYPE	VALUE
utcOutput	string	20190305T175318Z
utcShortOutput	string	03/05/2019
utcCustomOutput	string	3 5

The next example shows how to use a value from the function when setting a tag value.

```json
{
    "$schema": "https://schema.management.azure.com/schemas/2018-05-01/subscriptionDeploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "utcShort": {
            "type": "string",
            "defaultValue": "[utcNow('d')]"
        },
        "rgName": {
            "type": "string"
        }
    },
    "resources": [
        {
            "type": "Microsoft.Resources/resourceGroups",
            "apiVersion": "2018-05-01",
            "name": "[parameters('rgName')]",
            "location": "westeurope",
            "tags":{
                "createdDate": "[parameters('utcShort')]"
            },
            "properties":{}
        }
    ],
    "outputs": {
        "utcShort": {
            "type": "string",
            "value": "[parameters('utcShort')]"
        }
    }
}
```

Next steps

- For a description of the sections in an Azure Resource Manager template, see Authoring Azure Resource Manager templates.
- To merge multiple templates, see Using linked templates with Azure Resource Manager.
- To iterate a specified number of times when creating a type of resource, see Create multiple instances of resources in Azure Resource Manager.
- To see how to deploy the template you've created, see Deploy an application with Azure Resource Manager template.

Deletion of Azure resources for complete mode deployments

7/17/2019 • 11 minutes to read • Edit Online

This article describes how resource types handle deletion when not in a template that is deployed in complete mode.

The resource types marked with `Yes` are deleted when the type isn't in the template deployed with complete mode.

The resource types marked with `No` aren't automatically deleted when not in the template; however, they're deleted if the parent resource is deleted. For a full description of the behavior, see Azure Resource Manager deployment modes.

Microsoft.AAD

RESOURCE TYPE	COMPLETE MODE DELETION
DomainServices	Yes
DomainServices/oucontainer	No

microsoft.aadiam

RESOURCE TYPE	COMPLETE MODE DELETION
diagnosticSettings	No
diagnosticSettingsCategories	No

Microsoft.Addons

RESOURCE TYPE	COMPLETE MODE DELETION
supportProviders	No

Microsoft.ADHybridHealthService

RESOURCE TYPE	COMPLETE MODE DELETION
aadsupportcases	No
addsservices	No
agents	No

RESOURCE TYPE	COMPLETE MODE DELETION
anonymousapiusers	No
configuration	No
logs	No
reports	No
services	No

Microsoft.Advisor

RESOURCE TYPE	COMPLETE MODE DELETION
configurations	No
generateRecommendations	No
recommendations	No
suppressions	No

Microsoft.AlertsManagement

RESOURCE TYPE	COMPLETE MODE DELETION
actionRules	No
alerts	No
alertsList	No
alertsSummary	No
alertsSummaryList	No
smartDetectorAlertRules	No
smartDetectorRuntimeEnvironments	No
smartGroups	No

Microsoft.AnalysisServices

RESOURCE TYPE	COMPLETE MODE DELETION
servers	Yes

Microsoft.ApiManagement

RESOURCE TYPE	COMPLETE MODE DELETION
reportFeedback	No
service	Yes
validateServiceName	No

Microsoft.Attestation

RESOURCE TYPE	COMPLETE MODE DELETION
attestationProviders	No

Microsoft.Authorization

RESOURCE TYPE	COMPLETE MODE DELETION
classicAdministrators	No
denyAssignments	No
elevateAccess	No
locks	No
permissions	No
policyAssignments	No
policyDefinitions	No
policySetDefinitions	No
providerOperations	No
roleAssignments	No
roleDefinitions	No

Microsoft.Automation

RESOURCE TYPE	COMPLETE MODE DELETION
automationAccounts	Yes
automationAccounts/configurations	Yes

RESOURCE TYPE	COMPLETE MODE DELETION
automationAccounts/jobs	No
automationAccounts/runbooks	Yes
automationAccounts/softwareUpdateConfigurations	No
automationAccounts/webhooks	No

Microsoft.Azure.Geneva

RESOURCE TYPE	COMPLETE MODE DELETION
environments	No
environments/accounts	No
environments/accounts/namespaces	No
environments/accounts/namespaces/configurations	No

Microsoft.AzureActiveDirectory

RESOURCE TYPE	COMPLETE MODE DELETION
b2cDirectories	Yes

Microsoft.AzureStack

RESOURCE TYPE	COMPLETE MODE DELETION
registrations	Yes
registrations/customerSubscriptions	No
registrations/products	No

Microsoft.Batch

RESOURCE TYPE	COMPLETE MODE DELETION
batchAccounts	Yes

Microsoft.Billing

RESOURCE TYPE	COMPLETE MODE DELETION
billingAccounts	No

RESOURCE TYPE	COMPLETE MODE DELETION
billingAccounts/billingProfiles	No
billingAccounts/billingProfiles/billingSubscriptions	No
billingAccounts/billingProfiles/invoices	No
billingAccounts/billingProfiles/invoices/pricesheet	No
billingAccounts/billingProfiles/operationStatus	No
billingAccounts/billingProfiles/paymentMethods	No
billingAccounts/billingProfiles/policies	No
billingAccounts/billingProfiles/pricesheet	No
billingAccounts/billingProfiles/products	No
billingAccounts/billingProfiles/transactions	No
billingAccounts/billingSubscriptions	No
billingAccounts/departments	No
billingAccounts/eligibleOffers	No
billingAccounts/enrollmentAccounts	No
billingAccounts/invoices	No
billingAccounts/invoiceSections	No
billingAccounts/invoiceSections/billingSubscriptions	No
billingAccounts/invoiceSections/billingSubscriptions/transfer	No
billingAccounts/invoiceSections/importRequests	No
billingAccounts/invoiceSections/initiateImportRequest	No
billingAccounts/invoiceSections/initiateTransfer	No
billingAccounts/invoiceSections/operationStatus	No
billingAccounts/invoiceSections/products	No
billingAccounts/invoiceSections/transfers	No
billingAccounts/products	No

RESOURCE TYPE	COMPLETE MODE DELETION
billingAccounts/projects	No
billingAccounts/projects/billingSubscriptions	No
billingAccounts/projects/importRequests	No
billingAccounts/projects/initiateImportRequest	No
billingAccounts/projects/operationStatus	No
billingAccounts/projects/products	No
billingAccounts/transactions	No
billingPeriods	No
BillingPermissions	No
billingProperty	No
BillingRoleAssignments	No
BillingRoleDefinitions	No
CreateBillingRoleAssignment	No
departments	No
enrollmentAccounts	No
importRequests	No
importRequests/acceptImportRequest	No
importRequests/declineImportRequest	No
invoices	No
transfers	No
transfers/acceptTransfer	No
transfers/declineTransfer	No
transfers/operationStatus	No
usagePlans	No

Microsoft.BingMaps

RESOURCE TYPE	COMPLETE MODE DELETION
mapApis	Yes
updateCommunicationPreference	No

Microsoft.BizTalkServices

RESOURCE TYPE	COMPLETE MODE DELETION
BizTalk	Yes

Microsoft.Blueprint

RESOURCE TYPE	COMPLETE MODE DELETION
blueprintAssignments	No
blueprintAssignments/assignmentOperations	No
blueprintAssignments/operations	No
blueprints	No
blueprints/artifacts	No
blueprints/versions	No
blueprints/versions/artifacts	No

Microsoft.BotService

RESOURCE TYPE	COMPLETE MODE DELETION
botServices	Yes
botServices/channels	No
botServices/connections	No

Microsoft.Cache

RESOURCE TYPE	COMPLETE MODE DELETION
Redis	Yes
RedisConfigDefinition	No

Microsoft.Capacity

RESOURCE TYPE	COMPLETE MODE DELETION
appliedReservations	No
calculatePrice	No
catalogs	No
commercialReservationOrders	No
reservationOrders	No
reservationOrders/calculateRefund	No
reservationOrders/merge	No
reservationOrders/reservations	No
reservationOrders/reservations/revisions	No
reservationOrders/return	No
reservationOrders/split	No
reservationOrders/swap	No
reservations	No
resources	No
validateReservationOrder	No

Microsoft.Cdn

RESOURCE TYPE	COMPLETE MODE DELETION
edgenodes	No
profiles	Yes
profiles/endpoints	Yes
profiles/endpoints/customdomains	No
profiles/endpoints/origins	No
validateProbe	No

Microsoft.CertificateRegistration

RESOURCE TYPE	COMPLETE MODE DELETION
certificateOrders	Yes
certificateOrders/certificates	No
validateCertificateRegistrationInformation	No

Microsoft.ClassicCompute

RESOURCE TYPE	COMPLETE MODE DELETION
capabilities	No
domainNames	No
domainNames/capabilities	No
domainNames/internalLoadBalancers	No
domainNames/serviceCertificates	No
domainNames/slots	No
domainNames/slots/roles	No
moveSubscriptionResources	No
operatingSystemFamilies	No
operatingSystems	No
quotas	No
resourceTypes	No
validateSubscriptionMoveAvailability	No
virtualMachines	No
virtualMachines/diagnosticSettings	No

Microsoft.ClassicInfrastructureMigrate

RESOURCE TYPE	COMPLETE MODE DELETION
classicInfrastructureResources	No

Microsoft.ClassicNetwork

RESOURCE TYPE	COMPLETE MODE DELETION
capabilities	No
expressRouteCrossConnections	No
expressRouteCrossConnections/peerings	No
gatewaySupportedDevices	No
networkSecurityGroups	No
quotas	No
reservedIps	No
virtualNetworks	No
virtualNetworks/remoteVirtualNetworkPeeringProxies	No
virtualNetworks/virtualNetworkPeerings	No

Microsoft.ClassicStorage

RESOURCE TYPE	COMPLETE MODE DELETION
capabilities	No
disks	No
images	No
osImages	No
osPlatformImages	No
publicImages	No
quotas	No
storageAccounts	No
storageAccounts/services	No
storageAccounts/services/diagnosticSettings	No
storageAccounts/vmImages	No
vmImages	No

Microsoft.CognitiveServices

RESOURCE TYPE	COMPLETE MODE DELETION
accounts	Yes

Microsoft.Commerce

RESOURCE TYPE	COMPLETE MODE DELETION
RateCard	No
UsageAggregates	No

Microsoft.Compute

RESOURCE TYPE	COMPLETE MODE DELETION
availabilitySets	Yes
disks	Yes
images	Yes
restorePointCollections	Yes
restorePointCollections/restorePoints	No
sharedVMImages	Yes
sharedVMImages/versions	Yes
snapshots	Yes
virtualMachines	Yes
virtualMachines/diagnosticSettings	No
virtualMachines/extensions	Yes
virtualMachineScaleSets	Yes
virtualMachineScaleSets/extensions	No
virtualMachineScaleSets/networkInterfaces	No
virtualMachineScaleSets/publicIPAddresses	No
virtualMachineScaleSets/virtualMachines	No
virtualMachineScaleSets/virtualMachines/networkInterfaces	No

Microsoft.Consumption

RESOURCE TYPE	COMPLETE MODE DELETION
AggregatedCost	No
Balances	No
Budgets	No
Charges	No
CostTags	No
credits	No
events	No
Forecasts	No
lots	No
Marketplaces	No
Pricesheets	No
products	No
ReservationDetails	No
ReservationRecommendations	No
ReservationSummaries	No
ReservationTransactions	No
Tags	No
Terms	No
UsageDetails	No

Microsoft.ContainerInstance

RESOURCE TYPE	COMPLETE MODE DELETION
containerGroups	Yes
serviceAssociationLinks	No

Microsoft.ContainerRegistry

RESOURCE TYPE	COMPLETE MODE DELETION
registries	Yes
registries/builds	No
registries/builds/cancel	No
registries/builds/getLogLink	No
registries/buildTasks	Yes
registries/buildTasks/steps	No
registries/eventGridFilters	No
registries/getBuildSourceUploadUrl	No
registries/GetCredentials	No
registries/importImage	No
registries/queueBuild	No
registries/regenerateCredential	No
registries/regenerateCredentials	No
registries/replications	Yes
registries/runs	No
registries/runs/cancel	No
registries/scheduleRun	No
registries/tasks	Yes
registries/updatePolicies	No
registries/webhooks	Yes
registries/webhooks/getCallbackConfig	No
registries/webhooks/ping	No

Microsoft.ContainerService

RESOURCE TYPE	COMPLETE MODE DELETION
containerServices	Yes

RESOURCE TYPE	COMPLETE MODE DELETION
managedClusters	Yes

Microsoft.ContentModerator

RESOURCE TYPE	COMPLETE MODE DELETION
applications	Yes
updateCommunicationPreference	No

Microsoft.CortanaAnalytics

RESOURCE TYPE	COMPLETE MODE DELETION
accounts	Yes

Microsoft.CostManagement

RESOURCE TYPE	COMPLETE MODE DELETION
Alerts	No
BillingAccounts	No
Connectors	Yes
Departments	No
Dimensions	No
EnrollmentAccounts	No
Query	No
register	No
Reportconfigs	No
Reports	No

Microsoft.CustomerInsights

RESOURCE TYPE	COMPLETE MODE DELETION
hubs	Yes
hubs/authorizationPolicies	No

RESOURCE TYPE	COMPLETE MODE DELETION
hubs/connectors	No
hubs/connectors/mappings	No
hubs/interactions	No
hubs/kpi	No
hubs/links	No
hubs/profiles	No
hubs/roleAssignments	No
hubs/roles	No
hubs/suggestTypeSchema	No
hubs/views	No
hubs/widgetTypes	No

Microsoft.DataBox

RESOURCE TYPE	COMPLETE MODE DELETION
jobs	Yes

Microsoft.DataBoxEdge

RESOURCE TYPE	COMPLETE MODE DELETION
DataBoxEdgeDevices	Yes

Microsoft.Databricks

RESOURCE TYPE	COMPLETE MODE DELETION
workspaces	Yes
workspaces/virtualNetworkPeerings	No

Microsoft.DataCatalog

RESOURCE TYPE	COMPLETE MODE DELETION
catalogs	Yes

Microsoft.DataConnect

RESOURCE TYPE	COMPLETE MODE DELETION
connectionManagers	Yes

Microsoft.DataFactory

RESOURCE TYPE	COMPLETE MODE DELETION
dataFactories	Yes
dataFactories/diagnosticSettings	No
dataFactorySchema	No
factories	Yes
factories/integrationRuntimes	No

Microsoft.DataLakeAnalytics

RESOURCE TYPE	COMPLETE MODE DELETION
accounts	Yes
accounts/dataLakeStoreAccounts	No
accounts/storageAccounts	No
accounts/storageAccounts/containers	No

Microsoft.DataLakeStore

RESOURCE TYPE	COMPLETE MODE DELETION
accounts	Yes
accounts/eventGridFilters	No
accounts/firewallRules	No

Microsoft.DataMigration

RESOURCE TYPE	COMPLETE MODE DELETION
services	Yes
services/projects	Yes

Microsoft.DBforMariaDB

RESOURCE TYPE	COMPLETE MODE DELETION
servers	Yes
servers/recoverableServers	No
servers/virtualNetworkRules	No

Microsoft.DBforMySQL

RESOURCE TYPE	COMPLETE MODE DELETION
servers	Yes
servers/recoverableServers	No
servers/virtualNetworkRules	No

Microsoft.DBforPostgreSQL

RESOURCE TYPE	COMPLETE MODE DELETION
servers	Yes
servers/advisors	No
servers/queryTexts	No
servers/recoverableServers	No
servers/topQueryStatistics	No
servers/virtualNetworkRules	No
servers/waitStatistics	No

Microsoft.Devices

RESOURCE TYPE	COMPLETE MODE DELETION
IotHubs	Yes
IotHubs/eventGridFilters	No
ProvisioningServices	Yes
usages	No

Microsoft.DevSpaces

RESOURCE TYPE	COMPLETE MODE DELETION
controllers	Yes

Microsoft.DevTestLab

RESOURCE TYPE	COMPLETE MODE DELETION
labs	Yes
labs/serviceRunners	Yes
labs/virtualMachines	Yes
schedules	Yes

Microsoft.DocumentDB

RESOURCE TYPE	COMPLETE MODE DELETION
databaseAccountNames	No
databaseAccounts	Yes

Microsoft.DomainRegistration

RESOURCE TYPE	COMPLETE MODE DELETION
domains	Yes
domains/domainOwnershipIdentifiers	No
generateSsoRequest	No
topLevelDomains	No
validateDomainRegistrationInformation	No

Microsoft.DynamicsLcs

RESOURCE TYPE	COMPLETE MODE DELETION
lcsprojects	No
lcsprojects/clouddeployments	No
lcsprojects/connectors	No

Microsoft.EventGrid

RESOURCE TYPE	COMPLETE MODE DELETION
domains	Yes
domains/topics	No
eventSubscriptions	No
extensionTopics	No
topics	Yes
topicTypes	No

Microsoft.EventHub

RESOURCE TYPE	COMPLETE MODE DELETION
clusters	Yes
namespaces	Yes
namespaces/authorizationrules	No
namespaces/disasterrecoveryconfigs	No
namespaces/eventhubs	No
namespaces/eventhubs/authorizationrules	No
namespaces/eventhubs/consumergroups	No

Microsoft.Features

RESOURCE TYPE	COMPLETE MODE DELETION
features	No
providers	No

Microsoft.Gallery

RESOURCE TYPE	COMPLETE MODE DELETION
enroll	No
galleryitems	No

RESOURCE TYPE	COMPLETE MODE DELETION
generateartifactaccessuri	No
myareas	No
myareas/areas	No
myareas/areas/areas	No
myareas/areas/areas/galleryitems	No
myareas/areas/galleryitems	No
myareas/galleryitems	No
register	No
resources	No
retrieveresourcesbyid	No

Microsoft.GuestConfiguration

RESOURCE TYPE	COMPLETE MODE DELETION
guestConfigurationAssignments	No
software	No

Microsoft.HanaOnAzure

RESOURCE TYPE	COMPLETE MODE DELETION
hanaInstances	Yes

Microsoft.HDInsight

RESOURCE TYPE	COMPLETE MODE DELETION
clusters	Yes
clusters/applications	No

Microsoft.ImportExport

RESOURCE TYPE	COMPLETE MODE DELETION
jobs	Yes

Microsoft.InformationProtection

RESOURCE TYPE	COMPLETE MODE DELETION
labelGroups	No
labelGroups/labels	No
labelGroups/labels/conditions	No
labelGroups/labels/subLabels	No
labelGroups/labels/subLabels/conditions	No

microsoft.insights

RESOURCE TYPE	COMPLETE MODE DELETION
actiongroups	Yes
activityLogAlerts	Yes
alertrules	Yes
automatedExportSettings	No
autoscalesettings	Yes
baseline	No
calculatebaseline	No
components	Yes
components/events	No
components/pricingPlans	No
components/query	No
diagnosticSettings	No
diagnosticSettingsCategories	No
eventCategories	No
eventtypes	No
extendedDiagnosticSettings	No
logDefinitions	No

RESOURCE TYPE	COMPLETE MODE DELETION
logprofiles	No
logs	No
metricAlerts	Yes
migrateToNewPricingModel	No
myWorkbooks	No
queries	No
rollbackToLegacyPricingModel	No
scheduledqueryrules	Yes
vmInsightsOnboardingStatuses	No
webtests	Yes
workbooks	Yes

Microsoft.Intune

RESOURCE TYPE	COMPLETE MODE DELETION
diagnosticSettings	No
diagnosticSettingsCategories	No

Microsoft.IoTCentral

RESOURCE TYPE	COMPLETE MODE DELETION
IoTApps	Yes

Microsoft.IoTSpaces

RESOURCE TYPE	COMPLETE MODE DELETION
Graph	Yes

Microsoft.KeyVault

RESOURCE TYPE	COMPLETE MODE DELETION
deletedVaults	No

RESOURCE TYPE	COMPLETE MODE DELETION
vaults	Yes
vaults/accessPolicies	No
vaults/secrets	No

Microsoft.Kusto

RESOURCE TYPE	COMPLETE MODE DELETION
clusters	Yes
clusters/databases	No
clusters/databases/dataconnections	No
clusters/databases/eventhubconnections	No

Microsoft.LabServices

RESOURCE TYPE	COMPLETE MODE DELETION
labaccounts	Yes
users	No

Microsoft.LocationBasedServices

RESOURCE TYPE	COMPLETE MODE DELETION
accounts	Yes

Microsoft.LocationServices

RESOURCE TYPE	COMPLETE MODE DELETION
accounts	Yes

Microsoft.LogAnalytics

RESOURCE TYPE	COMPLETE MODE DELETION
logs	No

Microsoft.Logic

RESOURCE TYPE	COMPLETE MODE DELETION
integrationAccounts	Yes
workflows	Yes

Microsoft.MachineLearning

RESOURCE TYPE	COMPLETE MODE DELETION
commitmentPlans	Yes
webServices	Yes
Workspaces	Yes

Microsoft.MachineLearningExperimentation

RESOURCE TYPE	COMPLETE MODE DELETION
accounts	Yes
accounts/workspaces	Yes
accounts/workspaces/projects	Yes
teamAccounts	Yes
teamAccounts/workspaces	Yes
teamAccounts/workspaces/projects	Yes

Microsoft.MachineLearningModelManagement

RESOURCE TYPE	COMPLETE MODE DELETION
accounts	Yes

Microsoft.MachineLearningServices

RESOURCE TYPE	COMPLETE MODE DELETION
workspaces	Yes
workspaces/computes	No

Microsoft.ManagedIdentity

RESOURCE TYPE	COMPLETE MODE DELETION
Identities	No
userAssignedIdentities	Yes

Microsoft.Management

RESOURCE TYPE	COMPLETE MODE DELETION
getEntities	No
managementGroups	No
resources	No
startTenantBackfill	No
tenantBackfillStatus	No

Microsoft.Maps

RESOURCE TYPE	COMPLETE MODE DELETION
accounts	Yes
accounts/eventGridFilters	No

Microsoft.Marketplace

RESOURCE TYPE	COMPLETE MODE DELETION
offers	No
offerTypes	No
offerTypes/publishers	No
offerTypes/publishers/offers	No
offerTypes/publishers/offers/plans	No
offerTypes/publishers/offers/plans/agreements	No
offerTypes/publishers/offers/plans/configs	No
offerTypes/publishers/offers/plans/configs/importImage	No
privategalleryitems	No

RESOURCE TYPE	COMPLETE MODE DELETION
products	No

Microsoft.MarketplaceApps

RESOURCE TYPE	COMPLETE MODE DELETION
classicDevServices	Yes
updateCommunicationPreference	No

Microsoft.MarketplaceOrdering

RESOURCE TYPE	COMPLETE MODE DELETION
agreements	No
offertypes	No

Microsoft.Media

RESOURCE TYPE	COMPLETE MODE DELETION
mediaservices	Yes
mediaservices/accountFilters	No
mediaservices/assets	No
mediaservices/assets/assetFilters	No
mediaservices/contentKeyPolicies	No
mediaservices/eventGridFilters	No
mediaservices/liveEventOperations	No
mediaservices/liveEvents	Yes
mediaservices/liveEvents/liveOutputs	No
mediaservices/liveOutputOperations	No
mediaservices/streamingEndpointOperations	No
mediaservices/streamingEndpoints	Yes
mediaservices/streamingLocators	No

RESOURCE TYPE	COMPLETE MODE DELETION
mediaservices/streamingPolicies	No
mediaservices/transforms	No
mediaservices/transforms/jobs	No

Microsoft.Migrate

RESOURCE TYPE	COMPLETE MODE DELETION
projects	Yes

Microsoft.Network

RESOURCE TYPE	COMPLETE MODE DELETION
applicationGateways	Yes
applicationSecurityGroups	Yes
azureFirewallFqdnTags	No
azureFirewalls	Yes
bgpServiceCommunities	No
connections	Yes
ddosCustomPolicies	Yes
ddosProtectionPlans	Yes
dnsOperationStatuses	No
dnszones	Yes
dnszones/A	No
dnszones/AAAA	No
dnszones/all	No
dnszones/CAA	No
dnszones/CNAME	No
dnszones/MX	No
dnszones/NS	No

RESOURCE TYPE	COMPLETE MODE DELETION
dnszones/PTR	No
dnszones/recordsets	No
dnszones/SOA	No
dnszones/SRV	No
dnszones/TXT	No
expressRouteCircuits	Yes
expressRouteServiceProviders	No
frontdoors	Yes
frontdoorWebApplicationFirewallPolicies	Yes
getDnsResourceReference	No
interfaceEndpoints	Yes
internalNotify	No
loadBalancers	Yes
localNetworkGateways	Yes
natGateways	Yes
networkIntentPolicies	Yes
networkInterfaces	Yes
networkProfiles	Yes
networkSecurityGroups	Yes
networkWatchers	Yes
networkWatchers/connectionMonitors	Yes
networkWatchers/lenses	Yes
networkWatchers/pingMeshes	Yes
privateLinkServices	Yes
publicIPAddresses	Yes

RESOURCE TYPE	COMPLETE MODE DELETION
publicIPPrefixes	Yes
routeFilters	Yes
routeTables	Yes
serviceEndpointPolicies	Yes
trafficManagerGeographicHierarchies	No
trafficmanagerprofiles	Yes
trafficmanagerprofiles/heatMaps	No
virtualHubs	Yes
virtualNetworkGateways	Yes
virtualNetworks	Yes
virtualNetworkTaps	Yes
virtualWans	Yes
vpnGateways	Yes
vpnSites	Yes
webApplicationFirewallPolicies	Yes

Microsoft.NotificationHubs

RESOURCE TYPE	COMPLETE MODE DELETION
namespaces	Yes
namespaces/notificationHubs	Yes

Microsoft.OperationalInsights

RESOURCE TYPE	COMPLETE MODE DELETION
devices	No
linkTargets	No
storageInsightConfigs	No
workspaces	Yes

RESOURCE TYPE	COMPLETE MODE DELETION
workspaces/dataSources	No
workspaces/linkedServices	No
workspaces/query	No

Microsoft.OperationsManagement

RESOURCE TYPE	COMPLETE MODE DELETION
managementassociations	No
managementconfigurations	Yes
solutions	Yes
views	Yes

Microsoft.PolicyInsights

RESOURCE TYPE	COMPLETE MODE DELETION
policyEvents	No
policyStates	No
policyTrackedResources	No
remediations	No

Microsoft.Portal

RESOURCE TYPE	COMPLETE MODE DELETION
consoles	No
dashboards	Yes
userSettings	No

Microsoft.PowerBI

RESOURCE TYPE	COMPLETE MODE DELETION
workspaceCollections	Yes

Microsoft.PowerBIDedicated

RESOURCE TYPE	COMPLETE MODE DELETION
capacities	Yes

Microsoft.ProjectOxford

RESOURCE TYPE	COMPLETE MODE DELETION
accounts	Yes

Microsoft.RecoveryServices

RESOURCE TYPE	COMPLETE MODE DELETION
backupProtectedItems	No
vaults	Yes

Microsoft.Relay

RESOURCE TYPE	COMPLETE MODE DELETION
namespaces	Yes
namespaces/authorizationrules	No
namespaces/hybridconnections	No
namespaces/hybridconnections/authorizationrules	No
namespaces/wcfrelays	No
namespaces/wcfrelays/authorizationrules	No

Microsoft.ResourceGraph

RESOURCE TYPE	COMPLETE MODE DELETION
resources	No
subscriptionsStatus	No

Microsoft.ResourceHealth

RESOURCE TYPE	COMPLETE MODE DELETION
availabilityStatuses	No
childAvailabilityStatuses	No

RESOURCE TYPE	COMPLETE MODE DELETION
childResources	No
events	No
impactedResources	No
notifications	No

Microsoft.Resources

RESOURCE TYPE	COMPLETE MODE DELETION
deployments	No
deployments/operations	No
links	No
notifyResourceJobs	No
providers	No
resourceGroups	No
resources	No
subscriptions	No
subscriptions/providers	No
subscriptions/resourceGroups	No
subscriptions/resourcegroups/resources	No
subscriptions/resources	No
subscriptions/tagnames	No
subscriptions/tagNames/tagValues	No
tenants	No

Microsoft.SaaS

RESOURCE TYPE	COMPLETE MODE DELETION
applications	Yes
saasresources	No

Microsoft.Scheduler

RESOURCE TYPE	COMPLETE MODE DELETION
flows	Yes
jobcollections	Yes

Microsoft.Search

RESOURCE TYPE	COMPLETE MODE DELETION
resourceHealthMetadata	No
searchServices	Yes

Microsoft.Security

RESOURCE TYPE	COMPLETE MODE DELETION
advancedThreatProtectionSettings	No
alerts	No
allowedConnections	No
appliances	No
applicationWhitelistings	No
AutoProvisioningSettings	No
Compliances	No
dataCollectionAgents	No
discoveredSecuritySolutions	No
externalSecuritySolutions	No
InformationProtectionPolicies	No
jitNetworkAccessPolicies	No
monitoring	No
monitoring/antimalware	No
monitoring/baseline	No
monitoring/patch	No

RESOURCE TYPE	COMPLETE MODE DELETION
policies	No
pricings	No
securityContacts	No
securitySolutions	No
securitySolutionsReferenceData	No
securityStatus	No
securityStatus/endpoints	No
securityStatus/subnets	No
securityStatus/virtualMachines	No
securityStatuses	No
securityStatusesSummaries	No
settings	No
tasks	No
topologies	No
workspaceSettings	No

Microsoft.SecurityGraph

RESOURCE TYPE	COMPLETE MODE DELETION
diagnosticSettings	No
diagnosticSettingsCategories	No

Microsoft.ServiceBus

RESOURCE TYPE	COMPLETE MODE DELETION
namespaces	Yes
namespaces/authorizationrules	No
namespaces/disasterrecoveryconfigs	No
namespaces/eventgridfilters	No

RESOURCE TYPE	COMPLETE MODE DELETION
namespaces/queues	No
namespaces/queues/authorizationrules	No
namespaces/topics	No
namespaces/topics/authorizationrules	No
namespaces/topics/subscriptions	No
namespaces/topics/subscriptions/rules	No
premiumMessagingRegions	No

Microsoft.ServiceFabric

RESOURCE TYPE	COMPLETE MODE DELETION
clusters	Yes
clusters/applications	No

Microsoft.ServiceFabricMesh

RESOURCE TYPE	COMPLETE MODE DELETION
applications	Yes
gateways	Yes
networks	Yes
secrets	Yes
volumes	Yes

Microsoft.SignalRService

RESOURCE TYPE	COMPLETE MODE DELETION
SignalR	Yes

Microsoft.Solutions

RESOURCE TYPE	COMPLETE MODE DELETION
applianceDefinitions	Yes

RESOURCE TYPE	COMPLETE MODE DELETION
appliances	Yes
applicationDefinitions	Yes
applications	Yes
jitRequests	Yes

Microsoft.SQL

RESOURCE TYPE	COMPLETE MODE DELETION
managedInstances	Yes
managedInstances/databases	Yes
managedInstances/databases/backupShortTermRetentionPolicies	No
managedInstances/databases/schemas/tables/columns/sensitivityLabels	No
managedInstances/databases/vulnerabilityAssessments	No
managedInstances/databases/vulnerabilityAssessments/rules/baselines	No
managedInstances/encryptionProtector	No
managedInstances/keys	No
managedInstances/restorableDroppedDatabases/backupShortTermRetentionPolicies	No
managedInstances/vulnerabilityAssessments	No
servers	Yes
servers/administrators	No
servers/communicationLinks	No
servers/databases	Yes
servers/encryptionProtector	No
servers/firewallRules	No
servers/keys	No

RESOURCE TYPE	COMPLETE MODE DELETION
servers/restorableDroppedDatabases	No
servers/serviceobjectives	No
servers/tdeCertificates	No

Microsoft.SqlVirtualMachine

RESOURCE TYPE	COMPLETE MODE DELETION
SqlVirtualMachineGroups	Yes
SqlVirtualMachineGroups/AvailabilityGroupListeners	No
SqlVirtualMachines	Yes

Microsoft.Storage

RESOURCE TYPE	COMPLETE MODE DELETION
storageAccounts	Yes
storageAccounts/blobServices	No
storageAccounts/fileServices	No
storageAccounts/queueServices	No
storageAccounts/services	No
storageAccounts/tableServices	No
usages	No

Microsoft.StorageSync

RESOURCE TYPE	COMPLETE MODE DELETION
storageSyncServices	Yes
storageSyncServices/registeredServers	No
storageSyncServices/syncGroups	No
storageSyncServices/syncGroups/cloudEndpoints	No
storageSyncServices/syncGroups/serverEndpoints	No

RESOURCE TYPE	COMPLETE MODE DELETION
storageSyncServices/workflows	No

Microsoft.StorSimple

RESOURCE TYPE	COMPLETE MODE DELETION
managers	Yes

Microsoft.StreamAnalytics

RESOURCE TYPE	COMPLETE MODE DELETION
streamingjobs	Yes
streamingjobs/diagnosticSettings	No

Microsoft.Subscription

RESOURCE TYPE	COMPLETE MODE DELETION
CreateSubscription	No
SubscriptionDefinitions	No
SubscriptionOperations	No

microsoft.support

RESOURCE TYPE	COMPLETE MODE DELETION
supporttickets	No

Microsoft.TerraformOSS

RESOURCE TYPE	COMPLETE MODE DELETION
providerRegistrations	Yes
resources	Yes

Microsoft.TimeSeriesInsights

RESOURCE TYPE	COMPLETE MODE DELETION
environments	Yes

RESOURCE TYPE	COMPLETE MODE DELETION
environments/accessPolicies	No
environments/eventsources	Yes
environments/referenceDataSets	Yes

microsoft.visualstudio

RESOURCE TYPE	COMPLETE MODE DELETION
account	Yes
account/extension	Yes
account/project	Yes

Microsoft.Web

RESOURCE TYPE	COMPLETE MODE DELETION
apiManagementAccounts	No
apiManagementAccounts/apiAcls	No
apiManagementAccounts/apis	No
apiManagementAccounts/apis/apiAcls	No
apiManagementAccounts/apis/connectionAcls	No
apiManagementAccounts/apis/connections	No
apiManagementAccounts/apis/connections/connectionAcls	No
apiManagementAccounts/apis/localizedDefinitions	No
apiManagementAccounts/connectionAcls	No
apiManagementAccounts/connections	No
billingMeters	No
certificates	Yes
connectionGateways	Yes
connections	Yes
customApis	Yes

RESOURCE TYPE	COMPLETE MODE DELETION
deletedSites	No
functions	No
hostingEnvironments	Yes
hostingEnvironments/multiRolePools	No
hostingEnvironments/multiRolePools/instances	No
hostingEnvironments/workerPools	No
hostingEnvironments/workerPools/instances	No
publishingUsers	No
recommendations	No
resourceHealthMetadata	No
runtimes	No
serverFarms	Yes
serverFarms/workers	No
sites	Yes
sites/domainOwnershipIdentifiers	No
sites/hostNameBindings	No
sites/instances	No
sites/instances/extensions	No
sites/premieraddons	Yes
sites/recommendations	No
sites/resourceHealthMetadata	No
sites/slots	Yes
sites/slots/hostNameBindings	No
sites/slots/instances	No
sites/slots/instances/extensions	No

RESOURCE TYPE	COMPLETE MODE DELETION
sourceControls	No
validate	No
verifyHostingEnvironmentVnet	No

Microsoft.WindowsDefenderATP

RESOURCE TYPE	COMPLETE MODE DELETION
diagnosticSettings	No
diagnosticSettingsCategories	No

Microsoft.WindowsIoT

RESOURCE TYPE	COMPLETE MODE DELETION
DeviceServices	Yes

Microsoft.WorkloadMonitor

RESOURCE TYPE	COMPLETE MODE DELETION
components	No
componentsSummary	No
monitorInstances	No
monitorInstancesSummary	No
monitors	No
notificationSettings	No

Next steps

To get the same data as a file of comma-separated values, download complete-mode-deletion.csv.

Resource providers for Azure services

6/25/2019 • 2 minutes to read • Edit Online

This article shows how resource provider namespaces map to Azure services.

Match resource provider to service

RESOURCE PROVIDER NAMESPACE	AZURE SERVICE
Microsoft.AAD	Azure Active Directory Domain Services
microsoft.aadiam	Azure Active Directory
Microsoft.Addons	core
Microsoft.ADHybridHealthService	Azure Active Directory
Microsoft.Advisor	Azure Advisor
Microsoft.AlertsManagement	Azure Monitor
Microsoft.AnalysisServices	Azure Analysis Services
Microsoft.ApiManagement	API Management
Microsoft.AppConfiguration	core
Microsoft.Authorization	Azure Resource Manager
Microsoft.Automation	Automation
Microsoft.AzureActiveDirectory	Azure Active Directory B2C
Microsoft.AzureStack	core
Microsoft.Batch	Batch
Microsoft.Billing	Billing
Microsoft.BingMaps	Bing Maps
Microsoft.BizTalkServices	BizTalk Services
Microsoft.Blockchain	Azure Blockchain Service
Microsoft.Blueprint	Azure Blueprints
Microsoft.BotService	Azure Bot Service

RESOURCE PROVIDER NAMESPACE	AZURE SERVICE
Microsoft.Cache	Azure Cache for Redis
Microsoft.Capacity	core
Microsoft.Cdn	Content Delivery Network
Microsoft.CertificateRegistration	App Service Certificates
Microsoft.ClassicCompute	Classic deployment model virtual machine
Microsoft.ClassicInfrastructureMigrate	Classic deployment model migration
Microsoft.ClassicNetwork	Classic deployment model virtual network
Microsoft.ClassicStorage	Classic deployment model storage
Microsoft.ClassicSubscription	Classic deployment model
Microsoft.CognitiveServices	Cognitive Services
Microsoft.Commerce	core
Microsoft.Compute	Virtual Machines
Microsoft.Consumption	Cost Management
Microsoft.ContainerInstance	Container Instances
Microsoft.ContainerRegistry	Container Registry
Microsoft.ContainerService	Azure Kubernetes Service (AKS)
Microsoft.ContentModerator	Azure Content Moderator
Microsoft.CostManagement	Cost Management
Microsoft.CustomerInsights	Customer Insights
Microsoft.CustomerLockbox	Customer Lockbox for Microsoft Azure
Microsoft.CustomProviders	Azure Custom Providers
Microsoft.DataBox	Azure Data Box
Microsoft.DataBoxEdge	Azure Data Box Edge
Microsoft.Databricks	Azure Databricks
Microsoft.DataCatalog	Data Catalog

RESOURCE PROVIDER NAMESPACE	AZURE SERVICE
Microsoft.DataFactory	Data Factory
Microsoft.DataLakeAnalytics	Data Lake Analytics
Microsoft.DataLakeStore	Azure Data Lake Store
Microsoft.DataMigration	Azure Database Migration Service
Microsoft.DBforMariaDB	Azure Database for MariaDB
Microsoft.DBforMySQL	Azure Database for MySQL
Microsoft.DBforPostgreSQL	Azure Database for PostgreSQL
Microsoft.DeploymentManager	Azure Deployment Manager
Microsoft.Devices	IoT Hub IoT Hub Device Provisioning Service
Microsoft.DevSpaces	Azure Dev Spaces
Microsoft.DevTestLab	Azure Lab Services
Microsoft.DocumentDB	Azure Cosmos DB
Microsoft.DomainRegistration	App Service
Microsoft.EnterpriseKnowledgeGraph	Enterprise Knowledge Graph
Microsoft.EventGrid	Event Grid
Microsoft.EventHub	Event Hubs
Microsoft.Features	Azure Resource Manager
Microsoft.Genomics	Microsoft Genomics
Microsoft.GuestConfiguration	Azure Policy
Microsoft.HanaOnAzure	SAP HANA on Azure
Microsoft.HardwareSecurityModules	Azure Dedicated HSM
Microsoft.HDInsight	HDInsight
Microsoft.HealthcareApis	Azure API for FHIR
Microsoft.ImportExport	Azure Import/Export
microsoft.insights	Azure Monitor

RESOURCE PROVIDER NAMESPACE	AZURE SERVICE
Microsoft.Intune	Intune
Microsoft.IoTCentral	IoT Central
Microsoft.IoTSpaces	Azure Digital Twins
Microsoft.KeyVault	Key Vault
Microsoft.Kusto	Azure Data Explorer
Microsoft.LabServices	Azure Lab Services
Microsoft.LocationBasedServices	Azure Maps
Microsoft.LocationServices	core
Microsoft.LogAnalytics	Azure Monitor
Microsoft.Logic	Logic Apps
Microsoft.MachineLearning	Machine Learning Studio
Microsoft.MachineLearningCompute	Machine Learning Service
Microsoft.MachineLearningModelManagement	Machine Learning Service
Microsoft.MachineLearningServices	Machine Learning Service
Microsoft.ManagedIdentity	Managed identities for Azure resources
Microsoft.ManagedLab	Azure Lab Services
Microsoft.Management	Management Groups
Microsoft.Maps	Azure Maps
Microsoft.Marketplace	core
Microsoft.MarketplaceApps	core
Microsoft.MarketplaceOrdering	core
Microsoft.Media	Media Services
Microsoft.Migrate	Azure Migrate
Microsoft.MixedReality	Azure Spatial Anchors
Microsoft.NetApp	Azure NetApp Files

RESOURCE PROVIDER NAMESPACE	AZURE SERVICE
Microsoft.Network	Virtual Network Load Balancer Application Gateway Azure DNS ExpressRoute VPN Gateway Traffic Manager Network Watcher Azure Firewall Azure Front Door Service Azure Bastion
Microsoft.NotificationHubs	Notification Hubs
Microsoft.OffAzure	Azure Migrate
Microsoft.OperationalInsights	Azure Monitor
Microsoft.OperationsManagement	Azure Monitor
Microsoft.PolicyInsights	Azure Policy
Microsoft.Portal	Azure portal
Microsoft.PowerBI	Power BI
Microsoft.PowerBIDedicated	Power BI Embedded
Microsoft.RecoveryServices	Site Recovery
Microsoft.Relay	Azure Relay
Microsoft.ResourceGraph	Azure Resource Graph
Microsoft.ResourceHealth	core
Microsoft.Resources	Azure Resource Manager
Microsoft.SaaS	core
Microsoft.Scheduler	Scheduler
Microsoft.Search	Azure Search
Microsoft.Security	Security Center
Microsoft.ServiceBus	Service Bus
Microsoft.ServiceFabric	Service Fabric
Microsoft.ServiceFabricMesh	Service Fabric Mesh

RESOURCE PROVIDER NAMESPACE	AZURE SERVICE
Microsoft.SignalRService	Azure SignalR Service
Microsoft.SiteRecovery	Site Recovery
Microsoft.Solutions	Azure Managed Applications
Microsoft.Sql	Azure SQL Database
Microsoft.SqlVirtualMachine	SQL Server on Azure Virtual Machines
Microsoft.Storage	Storage
Microsoft.StorageSync	Storage
Microsoft.StorSimple	StorSimple
Microsoft.StreamAnalytics	Stream Analytics
Microsoft.Subscription	core
microsoft.support	core
Microsoft.TimeSeriesInsights	Time Series Insights
microsoft.visualstudio	Azure DevOps
Microsoft.VMwareCloudSimple	Azure VMware Solution by CloudSimple
Microsoft.Web	App Service Functions
Microsoft.WindowsDefenderATP	Windows Defender Advanced Threat Protection
Microsoft.WindowsIoT	Windows 10 IoT Core Services
Microsoft.WorkloadMonitor	Azure Monitor

Next steps

For more information about resource providers, see Azure resource providers and types

Manage personal data associated with Azure Resource Manager

6/18/2019 • 2 minutes to read • Edit Online

To avoid exposing sensitive information, delete any personal information you may have provided in deployments, resource groups, or tags. Azure Resource Manager provides operations that let you manage personal data you may have provided in deployments, resource groups, or tags.

NOTE

This article provides steps for how to delete personal data from the device or service and can be used to support your obligations under the GDPR. If you're looking for general info about GDPR, see the GDPR section of the Service Trust portal.

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

Delete personal data in deployment history

For deployments, Resource Manager retains parameter values and status messages in the deployment history. These values persist until you delete the deployment from the history. To see if you have provided personal data in these values, list the deployments. If you find personal data, delete the deployments from the history.

To list **deployments** in the history, use:

- List By Resource Group
- Get-AzResourceGroupDeployment
- az group deployment list

To delete **deployments** from the history, use:

- Delete
- Remove-AzResourceGroupDeployment
- az group deployment delete

Delete personal data in resource group names

The name of the resource group persists until you delete the resource group. To see if you have provided personal data in the names, list the resource groups. If you find personal data, move the resources to a new resource group, and delete the resource group with personal data in the name.

To list **resource groups**, use:

- List
- Get-AzResourceGroup
- az group list

To delete **resource groups**, use:

- Delete
- Remove-AzResourceGroup
- az group delete

Delete personal data in tags

Tags names and values persist until you delete or modify the tag. To see if you have provided personal data in the tags, list the tags. If you find personal data, delete the tags.

To list **tags**, use:

- List
- Get-AzTag
- az tag list

To delete **tags**, use:

- Delete
- Remove-AzTag
- az tag delete

Next steps

- For an overview of Azure Resource Manager, see the What is Resource Manager?