

Contents

[Azure Resource Manager Documentation](#)

[Overview](#)

[What is Resource Manager?](#)

[Resource Manager and classic deployment](#)

[Quickstarts](#)

[Create templates - portal](#)

[Create templates - VS Code](#)

[Create templates - Visual Studio](#)

[Tutorials](#)

[Create encrypted storage account](#)

[Create multiple resources](#)

[Concepts](#)

[Resource providers and types](#)

[Management groups](#)

[Subscription governance](#)

[Templates for cloud consistency](#)

[How to](#)

[Create templates](#)

[Template sections](#)

[Parameters](#)

[Variables](#)

[Functions](#)

[Resources](#)

[Outputs](#)

[Linked and nested templates](#)

[Define dependency between resources](#)

[Create multiple instances](#)

[Update resource](#)

[Deploy](#)

Azure PowerShell

- [Deploy template](#)
- [Deploy private template with SAS token](#)
- [Export template and redeploy](#)

Azure CLI

- [Deploy template](#)
- [Deploy private template with SAS token](#)
- [Export template and redeploy](#)

Azure portal

- [Deploy resources](#)
- [Export template](#)

REST API

- [Multiple resource groups or subscriptions](#)
- [Continuous integration with Visual Studio Team Services](#)
- [Pass secure values during deployment](#)

Manage

- [Azure PowerShell](#)
- [Azure CLI](#)
- [Azure portal](#)
- [Use tags to organize resources](#)
- [Move resources to new group or subscription](#)
- [Create EA subscriptions](#)
- [Grant access to create EA subscriptions](#)
- [Create management groups](#)
- [Manage your management groups](#)

Control Access

- [Create service principal](#)
- [Azure PowerShell](#)
- [Azure CLI](#)
- [Azure portal](#)
- [Authentication API to access subscriptions](#)
- [Lock resources](#)

Audit

[View activity logs](#)

[View deployment operations](#)

Troubleshoot

[Troubleshoot deployment errors](#)

[AccountNameInvalid](#)

[InvalidTemplate](#)

[Linux deployment issues](#)

[NoRegisteredProviderFound](#)

[NotFound](#)

[ParentResourceNotFound](#)

[Provisioning and allocation issues for Linux](#)

[Provisioning and allocation issues for Windows](#)

[RequestDisallowedByPolicy](#)

[ReservedResourceName](#)

[ResourceQuotaExceeded](#)

[SkuNotAvailable](#)

[Windows deployment issues](#)

Reference

[Template reference](#)

[AKS](#)

[Managed Clusters](#)

[Analysis Services](#)

[Servers](#)

[API Management](#)

[Service](#)

[APIs](#)

[Diagnostics](#)

[Issues](#)

[Operations](#)

[Policies](#)

[Releases](#)

- Schemas
- Tag Descriptions
- Tags
- API Version Sets
- Authorization Servers
- Backends
- Certificates
- Diagnostics
 - Loggers
- Groups
 - Users
- Identity Providers
- Loggers
- Notifications
 - Recipient Emails
 - Recipient Users
- OpenID Connect Providers
- Policies
- Products
 - APIs
 - Groups
 - Policies
 - Tags
- Properties
- Subscriptions
- Tags
- Templates
- Users

- Certificates
- Compilation Jobs
- Configurations
- Connections
- Connection Types
- Credentials
- Jobs
- Job Schedules
- Modules
- Node Configurations
- Runbooks
- Schedules
- Software Update Configurations
- Source Controls
 - Source Controls Sync Jobs
- Variables
- Webhooks

Batch

- Batch Accounts
- Certificates
- Pools

Batch AI

- Clusters
- File Servers
- Jobs

Cache

- Redis
- Firewall Rules
- Linked Servers

CDN

- Profiles
- Endpoints

[Custom Domains](#)

[Certificate Registration](#)

[Certificate Orders](#)

[Certificates](#)

[Cognitive Services](#)

[Accounts](#)

[Compute](#)

[Availability Sets](#)

[Disks](#)

[Galleries](#)

[Images](#)

[Versions](#)

[Images](#)

[Snapshots](#)

[Virtual Machines](#)

[Extensions](#)

[Virtual Machine Scale Sets](#)

[Extensions](#)

[Virtual Machines](#)

[Container Instance](#)

[Container Groups](#)

[Container Registry](#)

[Registries](#)

[Replications](#)

[WebHooks](#)

[Container Service](#)

[Container Services](#)

[Cosmos DB](#)

[Database Accounts](#)

[Customer Insights](#)

[Hubs](#)

[Authorization Policies](#)

- Connectors
 - Mappings
- Interactions
- KPI
- Links
- Predictions
- Profiles
- Relationship Links
- Relationships
- Role Assignments
- Views

Data Catalog

- Catalogs

Data Factory

- Factories
- Datasets
- Integration Runtimes
- Linked Services
- Pipelines
- Triggers

Data Lake Analytics

- Accounts
- Compute Policies
- Data Lake Store Accounts
- Firewall Rules
- Storage Accounts

Data Lake Store

- Accounts
- Firewall Rules
- TrustedID Providers

Devices

- IoT Hubs

- Certificates
- Consumer Groups
- Provisioning Services
 - Certificates
- DevTest Labs
 - Labs
 - Artifact Sources
 - Costs
 - Custom Images
 - Formulas
 - Notification Channels
 - Policies
 - Schedules
 - Service Runners
 - Users
 - Disks
 - Environments
 - Secrets
 - Virtual Machines
 - Schedules
 - Virtual Networks
 - Schedules
 - Domain Registration
 - Domains
 - Domain Ownership Identifiers
 - Event Grid
 - Event Subscriptions
 - Topics
 - Event Hub
 - Namespaces
 - Authorization Rules
 - Disaster Recovery Configs

- [Event Hubs](#)
 - [Authorization Rules](#)
 - [Consumer Groups](#)
- [HDInsight](#)
 - [Clusters](#)
 - [Applications](#)
 - [Extensions](#)
 - [Import Export](#)
 - [Jobs](#)
 - [Insights](#)
 - [Action Groups](#)
 - [Activity Log Alerts](#)
 - [Alert Rules](#)
 - [Autoscale Settings](#)
 - [Components](#)
 - [Web Tests](#)
 - [Key Vault](#)
 - [Vaults](#)
 - [Location Based Services](#)
 - [Accounts](#)
 - [Logic](#)
 - [Integration Accounts](#)
 - [Agreements](#)
 - [Certificates](#)
 - [Maps](#)
 - [Partners](#)
 - [Schemas](#)
 - [Workflows](#)
 - [Machine Learning](#)
 - [Commitment Plans](#)
 - [Web Services](#)
 - [Media](#)

[Media Services](#)

[MySQL](#)

[Servers](#)

[Configurations](#)

[Databases](#)

[Firewall Rules](#)

[Network](#)

[Application Gateways](#)

[Application Security Groups](#)

[Connections](#)

[DNS Zones](#)

[A](#)

[AAAA](#)

[CAA](#)

[CNAME](#)

[MX](#)

[NS](#)

[PTR](#)

[SOA](#)

[SRV](#)

[TXT](#)

[Express Route Circuits](#)

[Authorizations](#)

[Peerings](#)

[Load Balancers](#)

[Inbound Nat Rules](#)

[Local Network Gateways](#)

[Network Interfaces](#)

[Network Security Groups](#)

[Security Rules](#)

[Network Watchers](#)

[Packet Captures](#)

[Public IP Addresses](#)

[Route Filters](#)

[Route Filters Rules](#)

[Route Tables](#)

[Routes](#)

[Traffic Manager Profiles](#)

[Virtual Network Gateways](#)

[Virtual Networks](#)

[Subnets](#)

[Virtual Network Peerings](#)

[Notification Hubs](#)

[Namespaces](#)

[Authorization Rules](#)

[Notification Hubs](#)

[Authorization Rules](#)

[Operational Insights](#)

[Workspaces](#)

[Data Sources](#)

[Linked Services](#)

[Machine Groups](#)

[Saved Searches](#)

[Storage Insight](#)

[PostgreSQL](#)

[Servers](#)

[Configurations](#)

[Databases](#)

[Firewall Rules](#)

[Power BI](#)

[Workspace Collections](#)

[Power BI Dedicated](#)

[Capacities](#)

[Recovery Services](#)

Vaults

Certificates

Backup Protection Intent

Replication Alert Settings

Replication Fabrics

 Replication Network Mappings

 Replication Protection Containers

 Replication Storage Classification Mappings

 Replication vCenters

Replication Policies

Replication Recovery Plans

Resources

 Deployments

Relay

 Namespaces

 Authorization Rules

 Hybrid Connections

 Authorization Rules

 WCF Relays

 Authorization Rules

Scheduler

 Job Collections

 Jobs

Search

 Search Services

Server Management

 Gateways

 Nodes

 Sessions

 PS Sessions

Service Bus

 Namespaces

- [Authorization Rules](#)
- [Disaster Recovery Configs](#)
- [Queues](#)
 - [Authorization Rules](#)
- [Topics](#)
 - [Authorization Rules](#)
 - [Subscriptions](#)
- [Service Fabric](#)
 - [Clusters](#)
- [SQL](#)
 - [Managed Instances](#)
 - [Databases](#)
 - [Auditing Settings](#)
 - [Backup Long Term Retention Policies](#)
 - [Transparent Data Encryption](#)
 - [Elastic Pools](#)
 - [Encryption Protector](#)
 - [Failover Groups](#)
 - [Firewall Rules](#)
 - [Keys](#)
 - [Virtual Network Rules](#)
- [Storage](#)
 - [Storage Accounts](#)
- [StorSimple](#)
 - [Managers](#)
 - [Access Control Records](#)
 - [Bandwidth Settings](#)
 - [Storage Account Credentials](#)
 - [Devices](#)
 - [Backup Policies](#)

[Volume Containers](#)

[Stream Analytics](#)

[Streaming Jobs](#)

[Functions](#)

[Inputs](#)

[Outputs](#)

[Transformations](#)

[Web](#)

[Certificates](#)

[Connection Gateways](#)

[Connections](#)

[Custom APIs](#)

[Hosting Environments](#)

[Worker Pools](#)

[Server Farms](#)

[Gateways](#)

[Routes](#)

[Sites](#)

[Deployments](#)

[Domain Ownership Identifiers](#)

[Functions](#)

[Host Name Bindings](#)

[Hybrid Connection](#)

[Hybrid Connection Namespaces Relays](#)

[Premier Add Ons](#)

[Public Certificates](#)

[Site Extensions](#)

[Slots](#)

[Deployments](#)

[Domain Ownership Identifiers](#)

[Functions](#)

[Host Name Bindings](#)

- Hybrid Connection
- Hybrid Connection Namespaces Relays
- Premier Add Ons
- Public Certificates
- Site Extensions
- Virtual Network Connections
- Virtual Network Connections
- Gateways

Template functions

- Array and object functions
- Comparison functions
- Deployment functions
- Logical functions
- Numeric functions
- Resource functions
- String functions

PowerShell

Azure CLI

.NET

Java

Python

REST

Resources

- Azure Roadmap
- Pricing calculator
- Service updates
- Stack Overflow
- Manage personal data
- Throttling requests
- Track asynchronous operations

Videos

Learn how to use Resource Manager to deploy, monitor, and manage solution resources as a group. Tutorials, API references, and other documentation show you how to set up resource groups and create templates for consistent and repeatable deployment.

[Learn about Azure Resource Manager](#)

[Azure Resource Manager Video Library](#)

[Get Started with Azure Resource Manager](#)

[Get Started creating dependencies in Azure Resource Manager templates](#)

[Get Started with multiple resource instances in Azure Resource Manager](#)

Reference

Command-Line

[PowerShell](#)

[Azure CLI](#)

Languages

[.NET](#)

[Java](#)

[Python](#)

[Template format](#)

[Template functions](#)

[UI definition functions](#)

[UI definition elements](#)

REST

[REST API](#)

Azure Resource Manager overview

7/18/2018 • 15 minutes to read • [Edit Online](#)

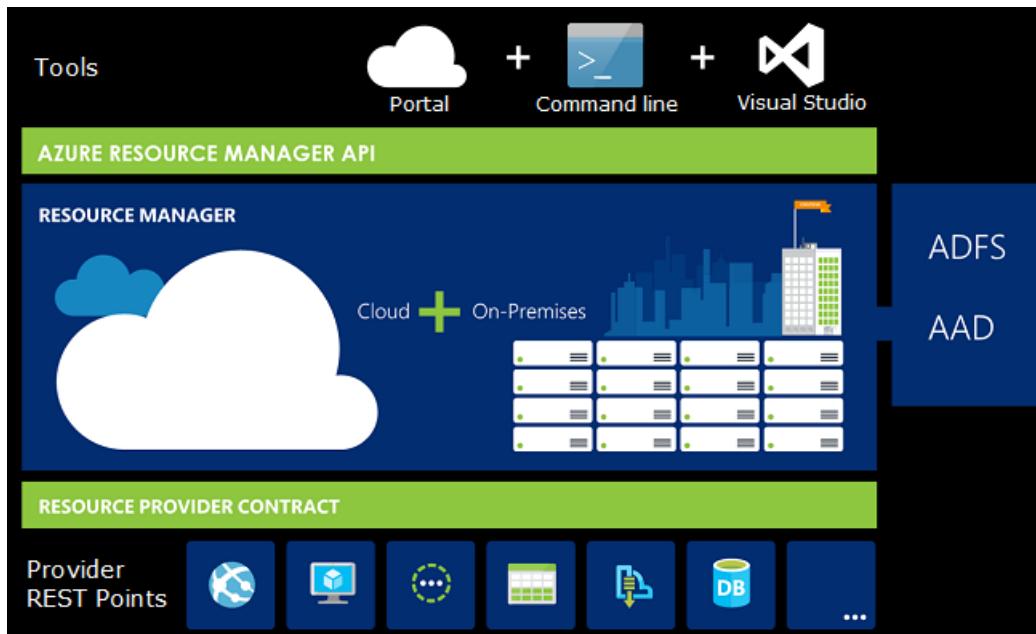
The infrastructure for your application is typically made up of many components – maybe a virtual machine, storage account, and virtual network, or a web app, database, database server, and third-party services. You don't see these components as separate entities, instead you see them as related and interdependent parts of a single entity. You want to deploy, manage, and monitor them as a group. Azure Resource Manager enables you to work with the resources in your solution as a group. You can deploy, update, or delete all the resources for your solution in a single, coordinated operation. You use a template for deployment and that template can work for different environments such as testing, staging, and production. Resource Manager provides security, auditing, and tagging features to help you manage your resources after deployment.

Consistent management layer

Resource Manager provides a consistent management layer to perform tasks through Azure PowerShell, Azure CLI, Azure portal, REST API, and client SDKs. All capabilities that are available in the Azure portal are also available through Azure PowerShell, Azure CLI, the Azure REST APIs, and client SDKs. Functionality initially released through APIs will be represented in the portal within 180 days of initial release.

Choose the tools and APIs that work best for you - they have the same capability and provide consistent results.

The following image shows how all the tools interact with the same 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 resource providers.



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. Some common resources are a virtual machine, storage account, web app, database, and virtual network, but there are many more.
- **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. See [Resource groups](#).

- **resource provider** - A service that supplies the resources you can deploy and manage through Resource Manager. Each resource provider offers operations for working with the resources that are deployed. Some common resource providers are Microsoft.Compute, which supplies the virtual machine resource, Microsoft.Storage, which supplies the storage account resource, and Microsoft.Web, which supplies resources related to web apps. See [Resource providers](#).
- **Resource Manager template** - A JavaScript Object Notation (JSON) file that defines one or more resources to deploy to a resource group. It also defines the dependencies between the deployed resources. 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.

Resource Manager provides a new way to deploy and manage your solutions. If you used the earlier deployment model and want to learn about the changes, see [Understanding Resource Manager deployment and classic deployment](#).

Guidance

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

1. Define and deploy your infrastructure through the declarative syntax in Resource Manager templates, rather than through imperative commands.
2. Define all deployment and configuration steps in the template. You should have no manual steps for setting up your solution.
3. Run imperative commands to manage your resources, such as to start or stop an app or machine.
4. 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 that you can use across global Azure, Azure sovereign clouds, and Azure Stack, see [Develop Azure Resource Manager templates for cloud consistency](#).

Resource groups

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

1. 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.
2. Each resource can only exist in one resource group.
3. You can add or remove a resource to a resource group at any time.
4. You can move a resource from one resource group to another group. For more information, see [Move resources to new resource group or subscription](#).
5. A resource group can contain resources that reside in different regions.
6. A resource group can be used to scope access control for administrative actions.
7. 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.

Resource providers

Each resource provider offers a set of resources and operations for working with an Azure service. 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}**. For example, the key vault type 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](#).

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. 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. You can 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.

To learn about the format of the template and how you construct it, see [Create your first Azure Resource Manager template](#). 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 (see the image for [Consistent management layer](#)). 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": {}
  }
]

```

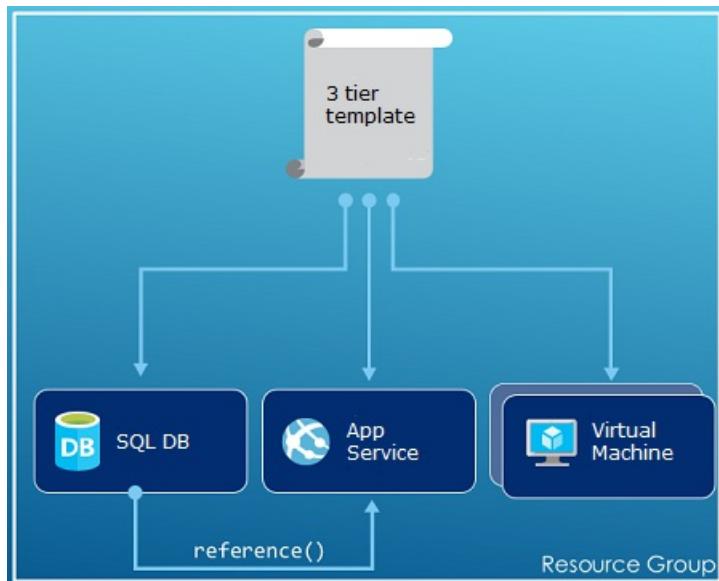
It converts the definition to the following REST API operation, which is sent to the Microsoft.Storage resource provider:

```

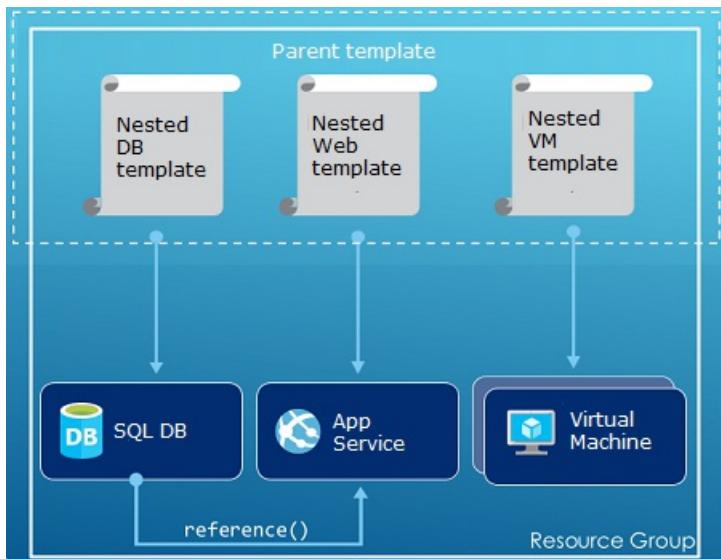
PUT
https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts/mystorageaccount?api-version=2016-01-01
REQUEST BODY
{
  "location": "westus",
  "properties": {},
  "sku": {
    "name": "Standard_LRS"
  },
  "kind": "Storage"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

```

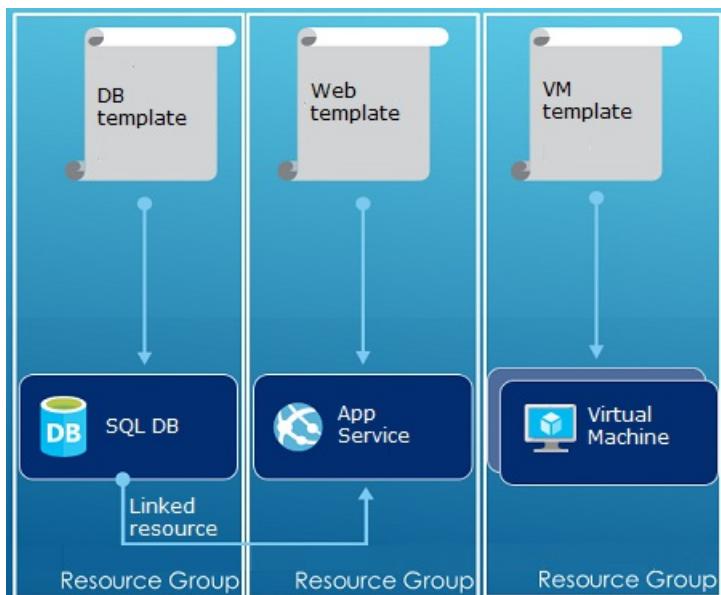
How you define templates and resource groups is entirely up to you and how you want to manage your solution. For example, you can deploy your three tier application through a single template to a single resource group.



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 specifies creating a resource but that resource already exists, Azure Resource Manager performs an update instead of creating a new asset. Azure Resource Manager updates the existing asset to the same state as it would be as new.

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](#).

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. For the commands 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](#)

Tags

Resource Manager provides a tagging feature that enables you to categorize resources according to your requirements for managing or billing. Use tags when you have a complex collection of resource groups and resources, and need to visualize those assets in the way that makes the most sense to you. For example, you could tag resources that serve a similar role in your organization or belong to the same department. Without tags, users in your organization can create multiple resources that may be difficult to later identify and manage. For example, you may wish to delete all the resources for a particular project. If those resources aren't tagged for the project, you have to manually find them. Tagging can be an important way for you to reduce unnecessary costs in your subscription.

Resources do not need to reside in the same resource group to share a tag. You can create your own tag taxonomy to ensure that all users in your organization use common tags rather than users inadvertently applying slightly different tags (such as "dept" instead of "department").

The following example shows a tag applied to a virtual machine.

```
"resources": [
  {
    "type": "Microsoft.Compute/virtualMachines",
    "apiVersion": "2015-06-15",
    "name": "SimpleWindowsVM",
    "location": "[resourceGroup().location]",
    "tags": {
      "costCenter": "Finance"
    },
    ...
  }
]
```

To retrieve all the resources with a tag value, use the following PowerShell cmdlet:

```
Find-AzureRmResource -TagName costCenter -TagValue Finance
```

Or, the following Azure CLI 2.0 command:

```
az resource list --tag costCenter=Finance
```

You can also view tagged resources through the Azure portal.

The [usage report](#) for your subscription includes tag names and values, which enables you to break out costs by tags. For more information about tags, see [Using tags to organize your Azure resources](#).

Access control

Resource Manager enables you to control who has access to specific actions for your organization. It natively integrates role-based access control (RBAC) into the management platform and applies that access control to all services in your resource group.

There are two main concepts to understand when working with role-based access control:

- Role definitions - describe a set of permissions and can be used in many assignments.
- Role assignments - associate a definition with an identity (user or group) for a particular scope (subscription, resource group, or resource). The assignment is inherited by lower scopes.

You can add users to pre-defined platform and resource-specific roles. For example, you can take advantage of the pre-defined role called Reader that permits users to view resources but not change them. You add users in your organization that need this type of access to the Reader role and apply the role to the subscription, resource group, or resource.

Azure provides the following four platform roles:

1. Owner - can manage everything, including access
2. Contributor - can manage everything except access
3. Reader - can view everything, but can't make changes
4. User Access Administrator - can manage user access to Azure resources

Azure also provides several resource-specific roles. Some common ones are:

1. Virtual Machine Contributor - can manage virtual machines but not grant access to them, and can't manage the virtual network or storage account to which they're connected
2. Network Contributor - can manage all network resources, but not grant access to them
3. Storage Account Contributor - Can manage storage accounts, but not grant access to them
4. SQL Server Contributor - Can manage SQL servers and databases, but not their security-related policies
5. Website Contributor - Can manage websites, but not the web plans to which they're connected

For the full list of roles and permitted actions, see [RBAC: Built in Roles](#). For more information about role-based access control, see [Azure Role-based Access Control](#).

In some cases, you want to run code or script that accesses resources, but you don't want to run it under a user's credentials. Instead, you want to create an identity called a service principal for the application and assign the appropriate role for the service principal. Resource Manager enables you to create credentials for the application and programmatically authenticate the application. To learn about creating service principals, see one of following topics:

- [Use Azure PowerShell to create a service principal to access resources](#)
- [Use Azure CLI to create a service principal to access resources](#)
- [Use portal to create Azure Active Directory application and service principal that can access resources](#)

You can also explicitly lock critical resources to prevent users from deleting or modifying them. For more information, see [Lock resources with Azure Resource Manager](#).

Activity logs

Resource Manager logs all operations that create, modify, or delete a resource. You can use the activity logs to find an error when troubleshooting or to monitor how a user in your organization modified a resource. You can filter the logs by many different values including which user initiated the operation. For information about working with the activity logs, see [View activity logs to manage Azure resources](#).

Customized policies

Resource Manager enables you to create customized policies for managing your resources. The types of policies you create can include diverse scenarios. You can enforce a naming convention on resources, limit which types and instances of resources can be deployed, or limit which regions can host a type of resource. You can require a tag value on resources to organize billing by departments. You create policies to help reduce costs and maintain consistency in your subscription.

You define policies with JSON and then apply those policies either across your subscription or within a resource group. Policies are different than role-based access control because they're applied to resource types.

The following example shows a policy that ensures tag consistency by specifying that all resources include a costCenter tag.

```
{  
  "if": {  
    "not" : {  
      "field" : "tags",  
      "containsKey" : "costCenter"  
    }  
  },  
  "then" : {  
    "effect" : "deny"  
  }  
}
```

There are many more types of policies you can create. For more information, see [What is Azure Policy?](#).

SDKs

Azure SDKs are available for multiple languages and platforms. Each of these language implementations is available through its ecosystem package manager and GitHub.

Here are the Open Source SDK repositories.

- [Azure SDK for .NET](#)
- [Azure Management Libraries for Java](#)
- [Azure SDK for Node.js](#)
- [Azure SDK for PHP](#)
- [Azure SDK for Python](#)
- [Azure SDK for Ruby](#)

For information about using these languages with your resources, see:

- [Azure for .NET developers](#)
- [Azure for Java developers](#)
- [Azure for Node.js developers](#)
- [Azure for Python developers](#)

NOTE

If the SDK doesn't provide the required functionality, you can also call to the [Azure REST API](#) directly.

Next steps

- For a simple introduction to working with templates, see [Export an Azure Resource Manager template from existing resources](#).
- For a more thorough walkthrough of creating a template, see [Create your first Azure Resource Manager template](#).
- To understand the functions you can use in a template, see [Template functions](#)
- For information about using Visual Studio with Resource Manager, see [Creating and deploying Azure resource groups through Visual Studio](#).

- For information on migration of resources from Classic to ARM, see [Migrate from Classic to Azure Resource Manager](#)

Here's a video demonstration of this overview:

Azure Resource Manager vs. classic deployment: Understand deployment models and the state of your resources

5/21/2018 • 10 minutes to read • [Edit Online](#)

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](#).

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-AzureRmResource -ResourceGroupName ExampleGroup -ResourceType Microsoft.ClassicCompute/virtualMachines
```

It returns the virtual machine:

```
Name      : ExampleClassicVM
ResourceId   :
/subscriptions/{guid}/resourceGroups/ExampleGroup/providers/Microsoft.ClassicCompute/virtualMachines/ExampleClassicVM
ResourceName  : ExampleClassicVM
ResourceType   : Microsoft.ClassicCompute/virtualMachines
ResourceGroupName : ExampleGroup
Location      : westus
SubscriptionId : {guid}
```

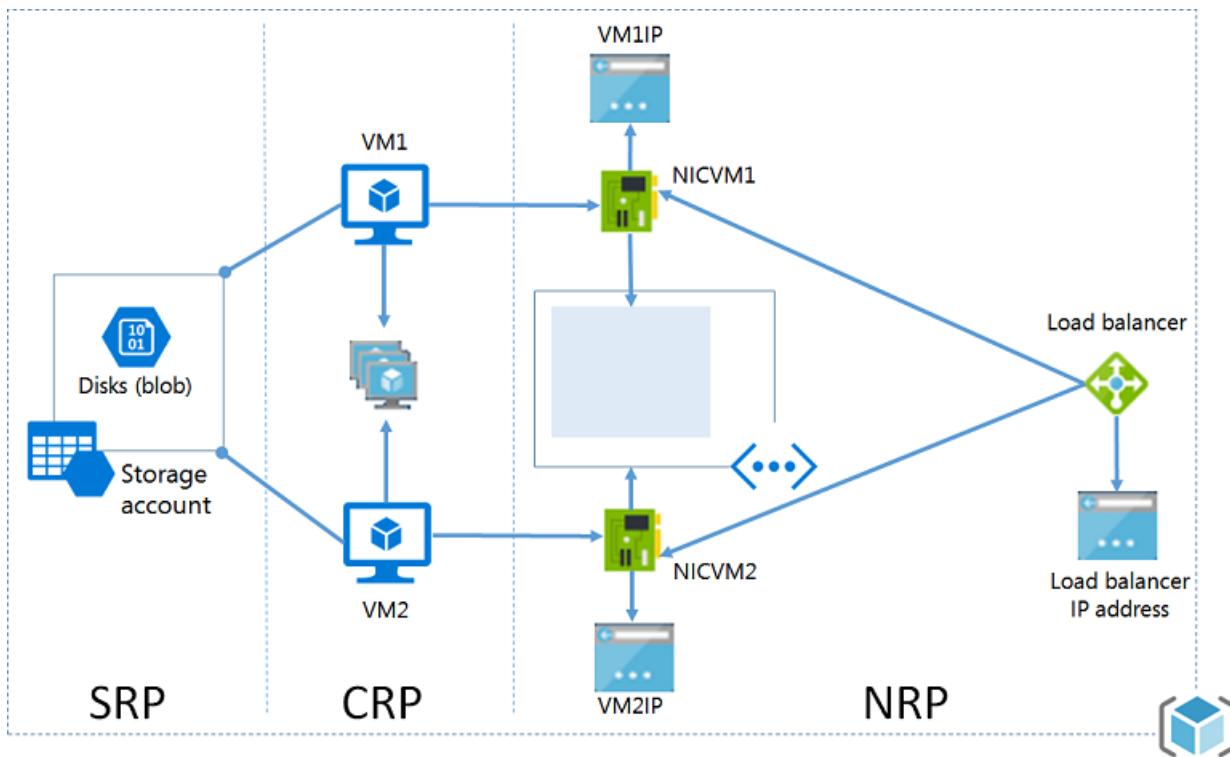
However, the Resource Manager cmdlet **Get-AzureRmVM** only returns virtual machines deployed through Resource Manager. The following command does not return the virtual machine created through classic deployment.

```
Get-AzureRmVM -ResourceGroupName ExampleGroup
```

Only resources created through Resource Manager support tags. You cannot apply tags to classic resources.

Changes for compute, network, and storage

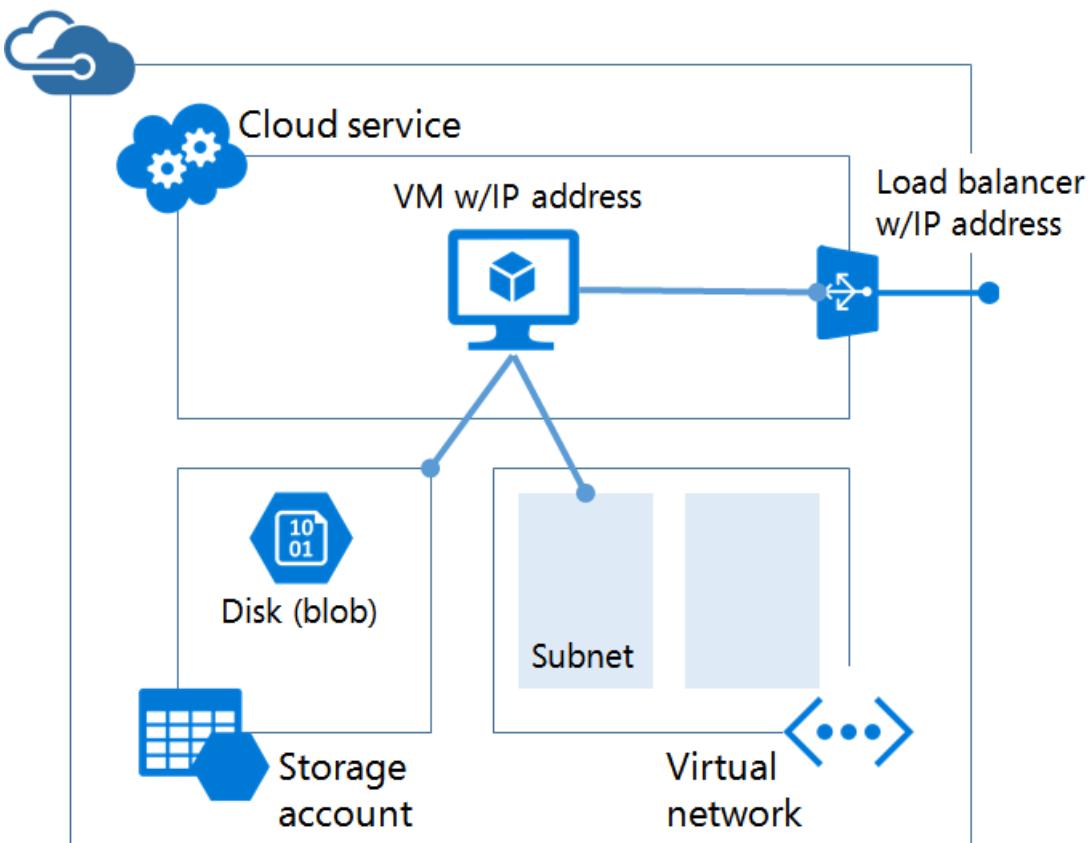
The following diagram displays compute, network, and storage resources deployed through Resource Manager.



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: <code>mycoffeeshop.cloudapp.net</code>	DNS Names are optional parameters that can be specified on a Public IP Address resource. The FQDN is in the following format - <code><domainlabel>. <region>.cloudapp.azure.com</code>

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](#).

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

7/18/2018 • 4 minutes to read • [Edit Online](#)

Learn how to create your first Azure Resource Manager template by generating one using the Azure portal, and how to edit and deploy the template from the portal.

Resource Manager templates are JSON files that define the resources you need to deploy for your solution. To create a template, you don't have to always start from scratch. In this tutorial, you learn how to generate a template from the Azure portal. You can then customize the template and deploy it.

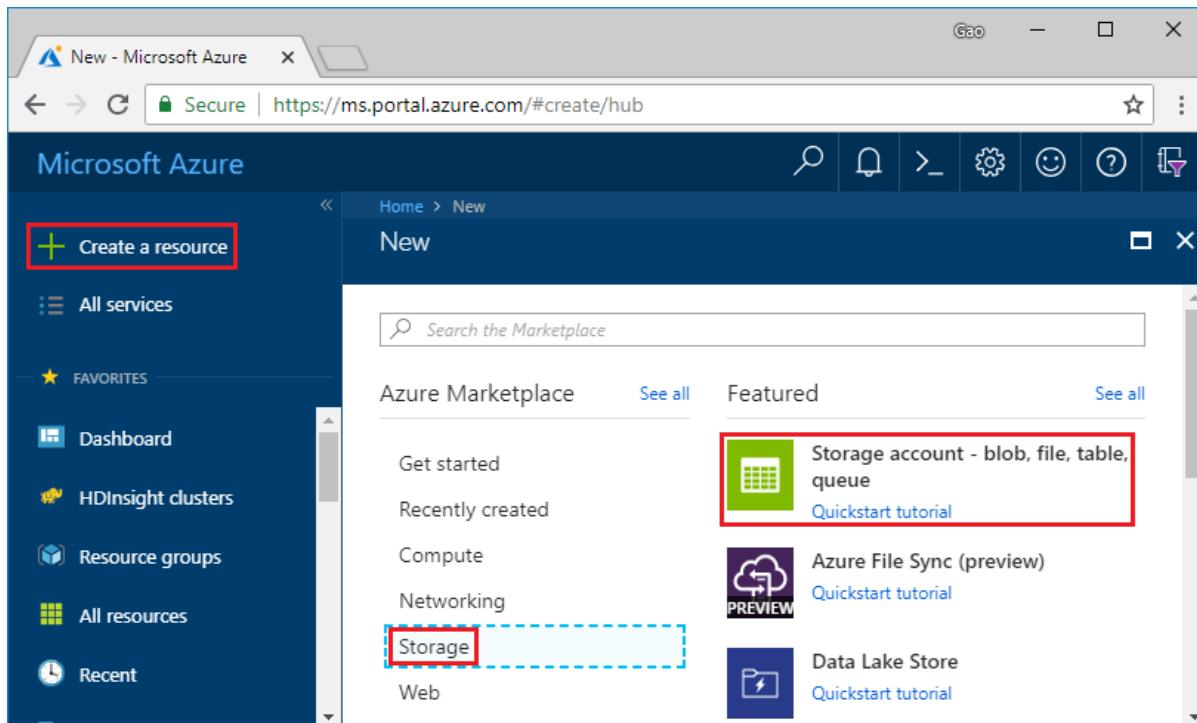
The instructions in this tutorial create an Azure Storage account. You can use the same process to create other Azure resources.

If you don't have an Azure subscription, [create a free account](#) before you begin.

Generate a template using the portal

In this section, you create a storage account using the Azure portal. Before you deploy the storage account, you have the option to explore the template generated by the portal based on your configurations. You can save the template and reuse it in the future.

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



3. Enter the following information. Make sure to select **Automation options** instead of **Create** in the next step, so you can see the template before it is deployed.
 - **Name:** give your storage account a unique name. On the screenshot, the name is *mystorage0626*.
 - **Resource group:** create a new Azure resource group with the name of your choice. On the screenshot, the resource group name is *mystorage0626rg*.

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

Create storage account X

The cost of your storage account depends on the usage and the options you choose below.
[Learn more](#)

* Name i
mystorage0626 ✓
.core.windows.net

Deployment model i
Resource manager Classic

Account kind i
Storage (general purpose v1)

* Location
East US

Replication i
Locally-redundant storage (LRS)

Performance i
Standard Premium

* Secure transfer required i
Disabled Enabled

* Subscription
<Azure subscription Name>

* Resource group
 Create new Use existing
mystorage0626rg ✓

Virtual networks
Configure virtual networks i
Disabled Enabled

Create Automation options

NOTE

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

4. Select **Automation options** at the bottom of the screen. The portal shows the template on the **Template** tab:

The screenshot shows the Azure Resource Manager Template editor interface. At the top, there are buttons for Download, Add to library, and Deploy. Below the header, there's a brief description of what the template does. The main area has tabs for Template, Parameters, CLI, PowerShell, .NET, and Ruby, with the Template tab selected. On the left, a navigation tree shows 'Parameters (5)', 'Variables (0)', and 'Resources (1)'. The 'Resources' node is expanded, showing a single item: '[parameters('name')]' (Microsoft.Storage). The main pane displays the JSON template code, which defines five parameters and a single storage account resource.

```
1 {
2     "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
3     "contentVersion": "1.0.0.0",
4     "parameters": {
5         "name": {
6             "type": "string"
7         },
8         "location": {
9             "type": "string"
10        },
11        "accountType": {
12            "type": "string"
13        },
14        "kind": {
15            "type": "string"
16        },
17        "httpsTrafficOnlyEnabled": {
18            "type": "bool"
19        }
20    },
21    "resources": [
22        {
23            "apiVersion": "2018-02-01",
24            "name": "[parameters('name')]",
25            "location": "[parameters('location')]",
26            "type": "Microsoft.Storage/storageAccounts",
27            "sku": {
28                "name": "[parameters('accountType')]"
29            },
30            "kind": "[parameters('kind')]",
31            "properties": {
32                "supportsHttpsTrafficOnly": "[parameters('httpsTrafficOnlyEnabled')]",
33                "encryption": {
34                    "services": {
35                        "blob": {
36                            "enabled": true
37                        },
38                        "file": {
39                            "enabled": true
40                        }
41                    },
42                    "keySource": "Microsoft.Storage"
43                },
44                "dependsOn": []
45            }
46        }
47    ]
48 }
```

The main pane shows the template. It is a JSON file with four top-level elements. For more information, see [Understand the structure and syntax of Azure Resource Manager Templates](#)

Under the **Parameter** element, there are five parameters defined. To see the values you provide during deployment, select the **Parameters** tab.

The screenshot shows the Azure Resource Manager template editor. At the top, there are three menu items: 'Download', 'Add to library', and 'Deploy'. Below the menu, there is a help section about template deployment. The tabs at the top are 'Template' (selected), 'Parameters' (highlighted with a red box), 'CLI', 'PowerShell', '.NET', and 'Ruby'. The main area contains the JSON template code:

```

1  {
2   "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
3   "contentversion": "1.0.0.0",
4   "parameters": {
5     "name": {
6       "value": "mystorage0626"
7     },
8     "location": {
9       "value": "eastus"
10    },
11    "accountType": {
12      "value": "Standard_LRS"
13    },
14    "kind": {
15      "value": "Storage"
16    },
17    "httpsTrafficOnlyEnabled": {
18      "value": false
19    }
20  }
21 }

```

These values are what you configured in the previous section. Using both the template and the parameters files, you can create an Azure storage account.

- On the top of the tabs, there are three menu items:

- Download:** Download the template and the parameters file to your local computer.
- Add to library:** Add the template to the library to be reused in the future.
- Deploy:** Deploy the Azure storage account to Azure.

In this tutorial, you use the **Add to library** option.

- Select **Add to library**.

- Enter **Name** and **Description**, and then select **Save**.

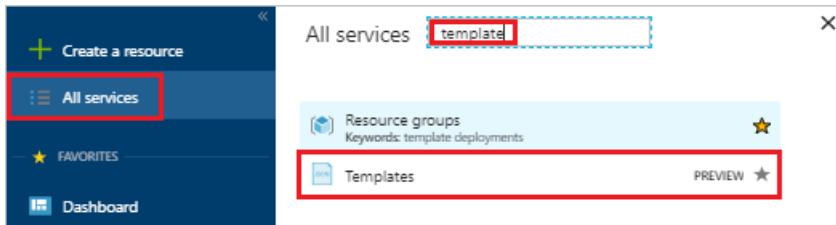
NOTE

The template library feature is in preview. Most people choose to save their templates to local computer or a public storage such as Github.

Edit and deploy the template

In this section, you open the saved template from the template library, edit the template inside the portal, and deploy the revised template. To edit a more complex template, consider using Visual Studio Code which provides richer edit functionalities.

- In the Azure portal, select **All services** from the left menu, enter **template** in the filter box, and then select **Template (PREVIEW)**.



- Select the template you saved in the last section. The name used on the screenshot is *mystorage0626*.
- Select **Edit**, and then select **Template added**.

The screenshot shows the Azure portal interface for managing templates. On the left, there's a list of templates with a 'createvm' item and 'mystorage0626' selected, which is also highlighted with a red box. In the center, the template details are displayed: DESCRIPTION (My first storage account), PUBLISHER (empty), and MODIFIED (7/11/2018). At the bottom, there's a 'View Template' link. On the right, the 'Edit Template' screen is open, showing a preview of the JSON template. A red box highlights the 'Edit' button at the top. Another red box highlights the message 'Template added' under the 'ARM Template' section.

4. Add a **variables** element, and then add one variable as shown in the following screenshot:

```
"variables": {  
    "storageAccountName": "[concat(uniquestring(resourceGroup().id), 'standardsa')]"  
},
```

ARM Template

mystorageU626

```
1 [
2     "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
3     "contentVersion": "1.0.0.0",
4     "parameters": {
5         "name": {
6             "type": "string"
7         },
8         "location": {
9             "type": "string"
10        },
11        "accountType": {
12            "type": "string"
13        },
14        "kind": {
15            "type": "string"
16        },
17        "httpsTrafficOnlyEnabled": {
18            "type": "bool"
19        }
20    },
21    "variables": {
22        "storageAccountName": "[concat(uniqueString(resourceGroup().id), 'standardsa')]"
23    },
24    "resources": [
25        {
26            "apiVersion": "2018-02-01",
27            "name": "[variables('storageAccountName')]",
28            "location": "[parameters('location')]",
29            "type": "Microsoft.Storage/storageAccounts",
30            "sku": {
31                "name": "[parameters('accountType')]"
32            },
33            "kind": "[parameters('kind')]",
34            "properties": {
35                "supportsHttpsTrafficonly": "[parameters('httpsTrafficOnlyEnabled')]",
36                "encryption": {
37                    "services": {
38                        "blob": {
39                            "enabled": true
40                        },
41                        "file": {
42                            "enabled": true
43                        }
44                    },
45                    "keySource": "Microsoft.Storage"
46                }
47            },
48            "dependsOn": []
49        }
50    ]
51 ]
```

OK

Two functions are used here: `concat()` and `uniqueString()`. The `uniqueString()` is helpful for creating a unique name for a resource.

5. Remove the **name** parameter highlighted in the previous screenshot.
6. 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"
        },
        "httpsTrafficOnlyEnabled": {
            "type": "bool"
        }
    },
    "variables": {
        "storageAccountName": "[concat(uniquestring(resourceGroup().id), 'standardsa')]"
    },
    "resources": [
        {
            "apiVersion": "2018-02-01",
            "name": "[variables('storageAccountName')]",
            "location": "[parameters('location')]",
            "type": "Microsoft.Storage/storageAccounts",
            "sku": {
                "name": "[parameters('accountType')]"
            },
            "kind": "[parameters('kind')]",
            "properties": {
                "supportsHttpsTrafficOnly": "[parameters('httpsTrafficOnlyEnabled')]",
                "encryption": {
                    "services": {
                        "blob": {
                            "enabled": true
                        },
                        "file": {
                            "enabled": true
                        }
                    },
                    "keySource": "Microsoft.Storage"
                }
            },
            "dependsOn": []
        }
    ]
}
```

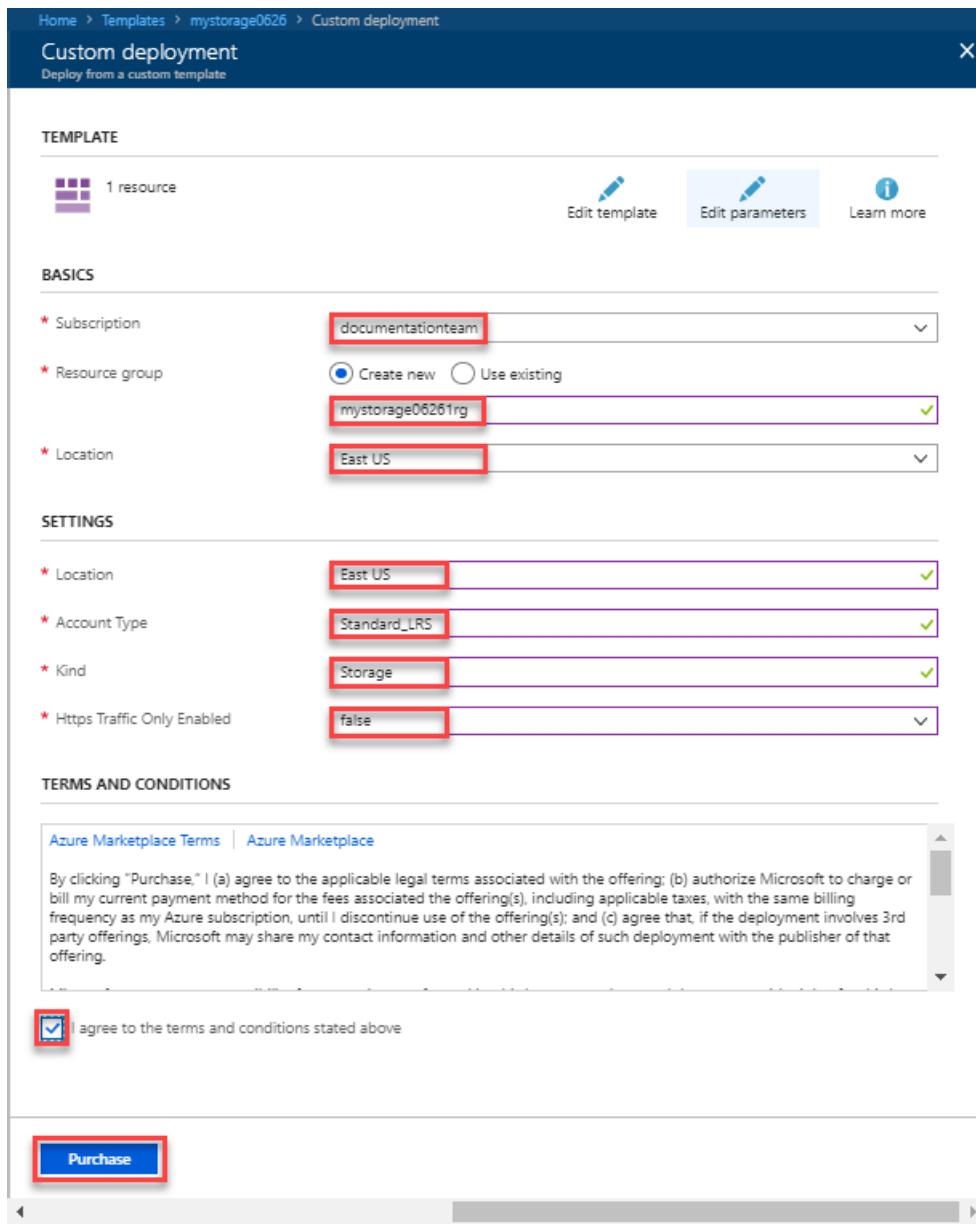
7. Select **OK**, and then select **Save** to save the changes.

8. Select **Deploy**.

9. Enter the following values:

- **Subscription:** select your Azure subscription.
- **Resource group:** name your resource group with a unique name.
- **Location:** select a location for the resource group.
- **Location:** select a location for the storage account. You can use the same location as the resource group.
- **Account Type:** Enter **Standard_LRS** for this quickstart.
- **Kind:** Enter **Storage** for this quickstart.
- **Https Traffic Only Enabled.** Select **false** for this quickstart.
- **I agree to the terms and conditions stated above:** (select)

Here is a screenshot of a sample deployment:



10. Select **Purchase**.

11. Select the bell icon (notifications) from the top of the screen to see the deployment status.

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.

Create templates by using Visual Studio Code

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

7/20/2018 • 3 minutes to read • [Edit Online](#)

Learn how to create Azure Resource Manager templates by using Visual Studio Code and the Azure Resource Manager Tools extension. 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](#).

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/azuredetect.json
```

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

Edit the template

To learn how to edit a template using Visual Studio Code, you add one more element into the outputs section.

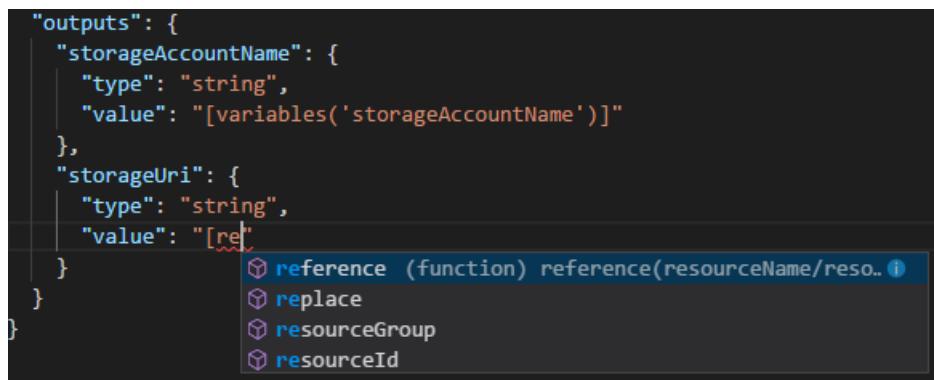
1. From Visual Studio Code, add one more output to the exported template:

```
"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.



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

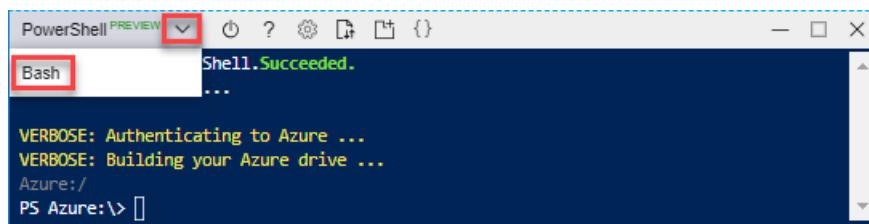
Deploy the template

There are many methods for deploying templates. In this quickstart, you use the Cloud shell from the Azure portal. The Cloud shell supports both Azure CLI and Azure PowerShell. The instructions provided here use CLI.

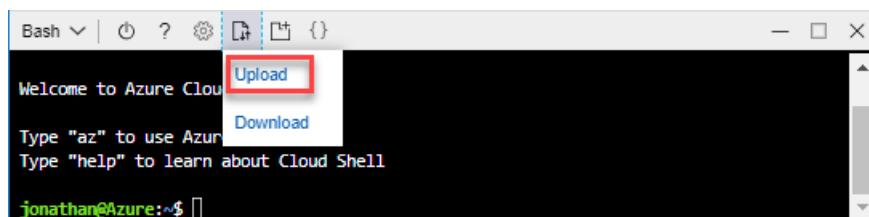
1. Sign in to the [Azure portal](#)
2. Select **Cloud Shell** from the upper right corner as shown in the following image:



3. Select the down arrow and then select **Bash** to switch to CLI from PowerShell.



4. Select **Restart** to restart the shell.
5. Select **Upload/download files**, and then select **Upload**.



6. Select the file you saved earlier in the quickstart. The default name is **azuredeploy.json**.
 7. From the Cloud shell, run the **ls** command to verify the file is uploaded successfully. You can also use the **cat** command to verify the template content.

8. From the Cloud shell, run the following commands:

```
az group create --name <ResourceGroupName> --location <AzureLocation>  
  
az group deployment create --name <DeploymentName> --resource-group <ResourceGroupName> --template-file  
<TemplateFileName>
```

Here is the screenshot of a sample deployment:

```
Bash | ⚡ ? 🌐 🔍 ⌂ {  
jonathan@Azure:~$ az group create --name myresourcegroup0709 --location eastus2  
{  
  "id": "/subscriptions/  
  "location": "eastus2",  
  "managedBy": null,  
  "name": "myresourcegroup0709",  
  "properties": {  
    "provisioningState": "Succeeded"  
  },  
  "tags": null  
}  
jonathan@Azure:~$ az group deployment create --name mydeploy0709 --resource-group myresourcegroup0709 --template-file azuredeploy.json  
{  
  "id": "/subscriptions/  
  "location": null,  
  "name": "mydeploy0709",  
  "properties": {  
    "correlationId": "43ead051-8908-41ce-bfa1-9676854391f4",  
    "debugSetting": null,  
    "dependencies": [],  
    "duration": "PT28.1781993S",  
    "mode": "Incremental",  
    "onErrorDeployment": null,  
    "outputResources": [  
      {  
        "id": "/subscriptions/  
        "resourceGroup": "myresourcegroup0709"  
      }  
    ],  
    "outputs": {  
      "storageAccountName": {  
        "type": "String",  
        "value": "3tqebj3slyfyestandardsa"  
      },  
      "storageUri": {  
        "type": "String",  
        "value": "https://3tqebj3slyfyestandardsa.blob.core.windows.net/"  
      }  
    },  
    "parameters": {  
      "location": {  
        "value": "eastus2"  
      }  
    }  
  }  
}  
jonathan@Azure:~$ az group deployment show --name mydeploy0709 --resource-group myresourcegroup0709  
{"id": "/subscriptions/  
  "location": "eastus2",  
  "name": "mydeploy0709",  
  "properties": {  
    "correlationId": "43ead051-8908-41ce-bfa1-9676854391f4",  
    "debugSetting": null,  
    "dependencies": [],  
    "duration": "PT28.1781993S",  
    "mode": "Incremental",  
    "onErrorDeployment": null,  
    "outputResources": [  
      {  
        "id": "/subscriptions/  
        "resourceGroup": "myresourcegroup0709"  
      }  
    ],  
    "outputs": {  
      "storageAccountName": {  
        "type": "String",  
        "value": "3tqebj3slyfyestandardsa"  
      },  
      "storageUri": {  
        "type": "String",  
        "value": "https://3tqebj3slyfyestandardsa.blob.core.windows.net/"  
      }  
    }  
  }  
}  
jonathan@Azure:~$ az group deployment history show --name mydeploy0709 --resource-group myresourcegroup0709  
{"id": "/subscriptions/  
  "location": "eastus2",  
  "name": "mydeploy0709",  
  "properties": {  
    "correlationId": "43ead051-8908-41ce-bfa1-9676854391f4",  
    "debugSetting": null,  
    "dependencies": [],  
    "duration": "PT28.1781993S",  
    "mode": "Incremental",  
    "onErrorDeployment": null,  
    "outputResources": [  
      {  
        "id": "/subscriptions/  
        "resourceGroup": "myresourcegroup0709"  
      }  
    ],  
    "outputs": {  
      "storageAccountName": {  
        "type": "String",  
        "value": "3tqebj3slyfyestandardsa"  
      },  
      "storageUri": {  
        "type": "String",  
        "value": "https://3tqebj3slyfyestandardsa.blob.core.windows.net/"  
      }  
    }  
  }  
}
```

On the screenshot, these values are used:

- <**ResourceGroupName**>: myresourcegroup0709. There are two appearances of the parameter. Make sure to use the same value.
 - <**AzureLocation**>: eastus2
 - <**DeployName**>: mydeployment0709
 - <**TemplateFile**>: azuredeploy.json

From the screenshot output, the storage account name is `3tqebj3slyfye``standardsa`.

9. Run the following PowerShell command to list the newly created storage account:

```
az storage account show --resource-group <ResourceGroupName> --name <StorageAccountName>
```

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 a template using Visual Studio Code, and how to deploy the template using the Azure portal Cloud shell. In the next tutorial, you learn more about how to develop a template, and how to use template reference.

[Create an encrypted Storage account](#)

Creating and deploying Azure resource groups through Visual Studio

7/12/2018 • 10 minutes to read • [Edit Online](#)

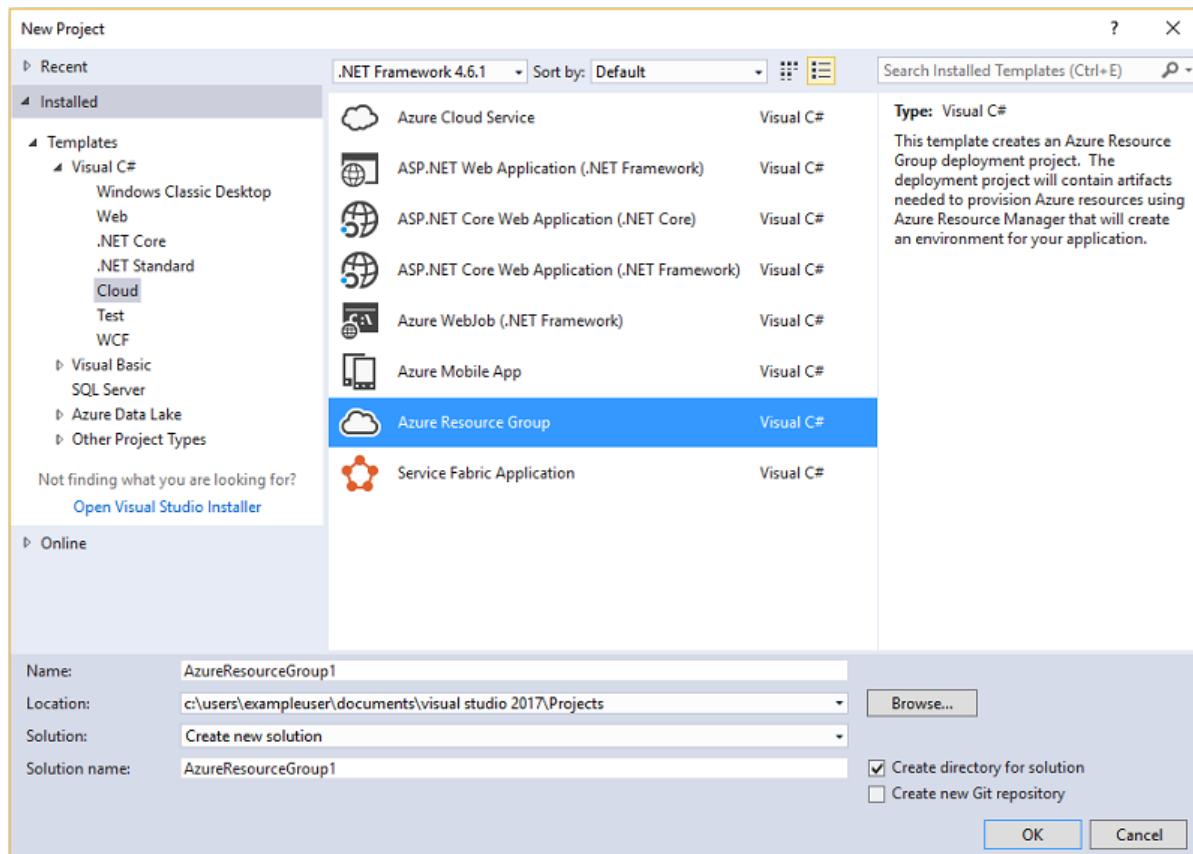
With Visual Studio, you can create a project that deploys your infrastructure and code to Azure. For example, you can define the web host, web site, and database for your app, and deploy that infrastructure along with the code. Visual Studio provides many different starter templates for deploying common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

This article shows how to use [Visual Studio 2017 with the Azure development and ASP.NET workloads installed](#). If you use Visual Studio 2015 Update 2 and Microsoft Azure SDK for .NET 2.9, or Visual Studio 2013 with Azure SDK 2.9, your experience is largely the same.

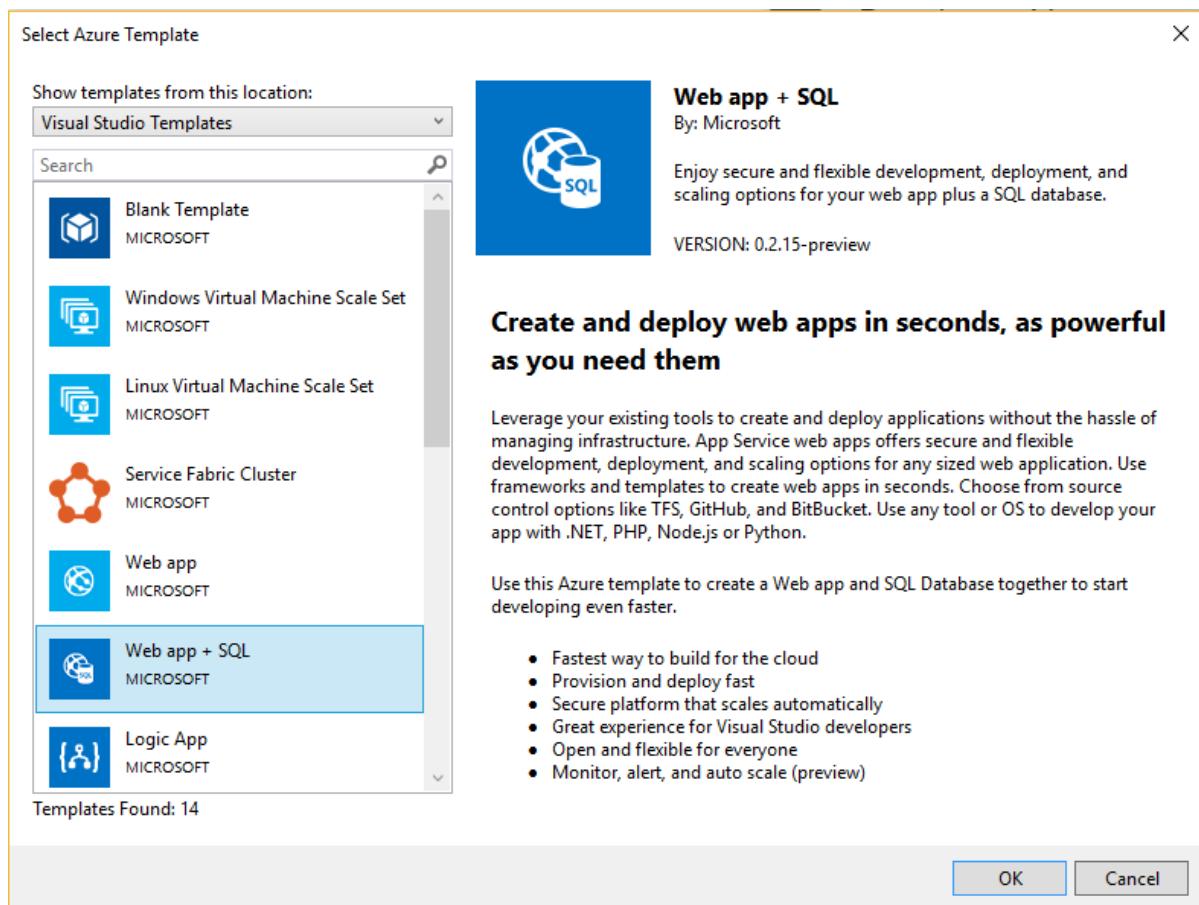
Create Azure Resource Group project

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

1. In Visual Studio, choose **File**, **New Project**, choose either **C#** or **Visual Basic** (which language you choose has no impact on the later stages as these projects have only JSON and PowerShell content). Then choose **Cloud**, and **Azure Resource Group** project.



2. 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 + SQL** template.



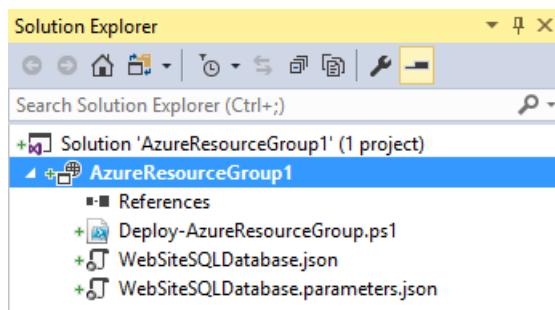
The template you pick is just a starting point; you can add and remove resources to fulfill your scenario.

NOTE

Visual Studio retrieves a list of available templates online. The list may change.

Visual Studio creates a resource group deployment project for the web app and SQL database.

3. To see what you created, look at the node in the deployment project.



Since you chose the Web app + SQL template for this example, you see the following files:

FILE NAME	DESCRIPTION
Deploy-AzureResourceGroup.ps1	A PowerShell script that runs PowerShell commands to deploy to Azure Resource Manager. Note Visual Studio uses this PowerShell script to deploy your template. Any changes you make to this script affect deployment in Visual Studio, so be careful.

FILE NAME	DESCRIPTION
WebSiteSQLDatabase.json	The Resource Manager template that defines the infrastructure you want to 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.
WebSiteSQLDatabase.parameters.json	A parameters file that has values needed by the template. You pass in parameter values to customize each deployment.

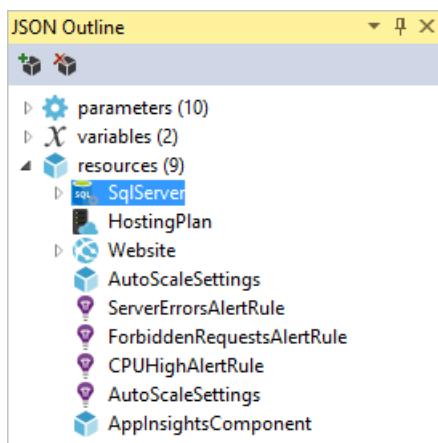
All resource group deployment projects have these basic files. Other projects may have additional files to support other functionality.

Customize the Resource Manager template

You can customize a deployment project by modifying the JSON templates that describe the resources you want to deploy. JSON stands for JavaScript Object Notation, and is a serialized data format that is easy to work with. The JSON files use a schema that you reference at the top of each file. If you want to understand the schema, you can download and analyze it. The schema defines what elements are valid, the types and formats of fields, and the possible values for a property. To learn about the elements of the Resource Manager template, see [Authoring Azure Resource Manager templates](#).

To work on your template, open **WebSiteSQLDatabase.json**.

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.



Selecting any of the elements in the outline takes you to that part of the template and highlights the corresponding JSON.

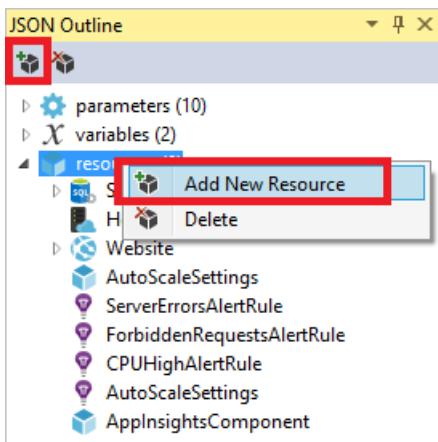
JSON Outline

```

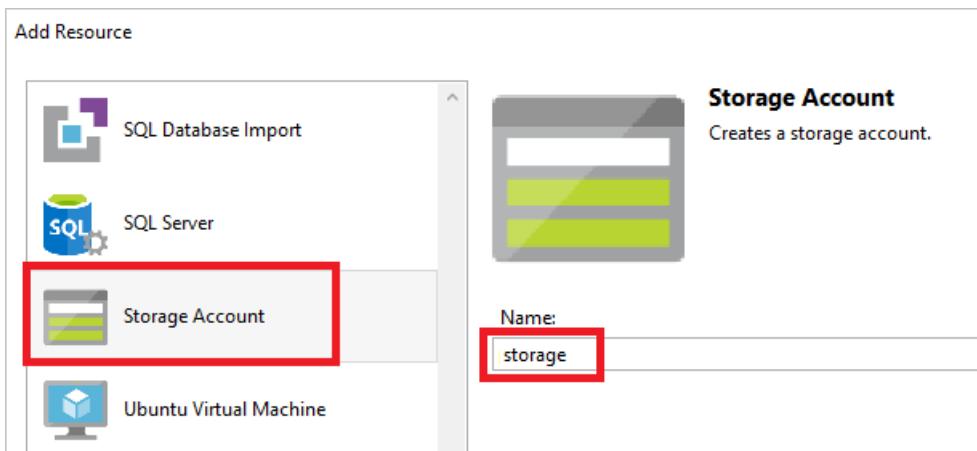
Schema: http://schema.management.azure.com/schemas/2015-01-01
{
  "name": "[parameters('hostingPlanName')]",
  "type": "Microsoft.Web/sites",
  "location": "[resourceGroup().location]",
  "dependsOn": [
    "[concat('Microsoft.Web/serverFarms/',"
  ],
  "tags": {
    "[concat('hidden-related:', resourceGroup().name, 'Website')]"
  },
  "properties": {
    "name": "[variables('webSiteName')]",
    "serverFarmId": "[resourceId('Microsoft.Web/serverFarms',"
  }
}

```

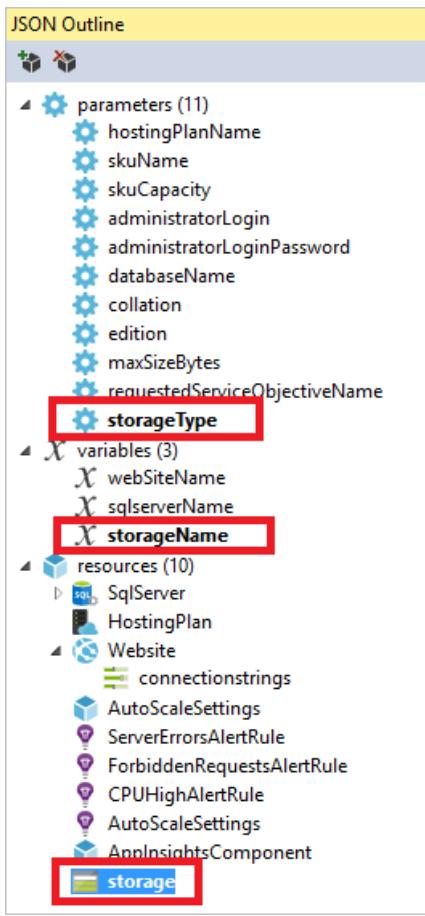
You can add a resource by either selecting the **Add Resource** button at the top of the JSON Outline window, or by right-clicking **resources** and selecting **Add New Resource**.



For this tutorial, 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.



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.



The **storageType** parameter 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.

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

Visual Studio also provides intellisense to help you understand what properties 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.

```
{
  "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')]",
    "numberOfWorkers": 1
  }
}, Microsoft.Web/serverfarms: The instance count, which is the number of workers per instance. Valid values are 1-10.
{
  "accessLevel": "Owner"
}
```

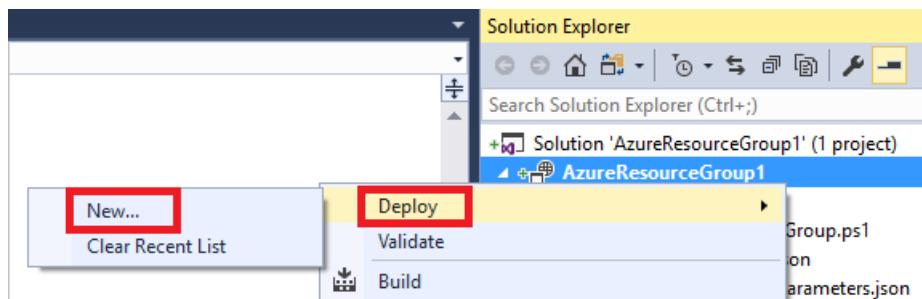
You can set **numberOfWorkers** to 1.

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

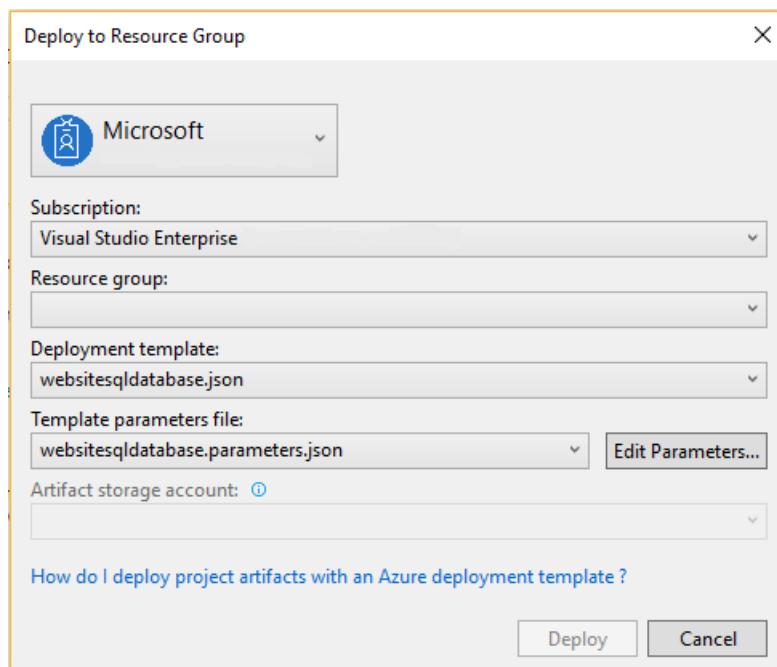
Deploy the Resource Group project to Azure

You're now ready to deploy your project. When you deploy an Azure Resource Group project, you deploy it to an Azure resource group. The resource group is a logical grouping of resources that share a common lifecycle.

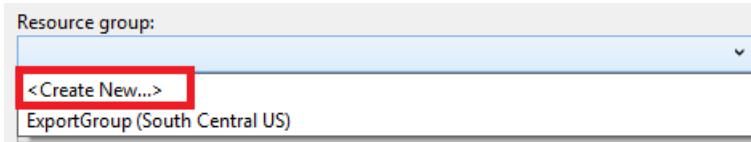
1. On the shortcut menu of the deployment project node, choose **Deploy > New**.



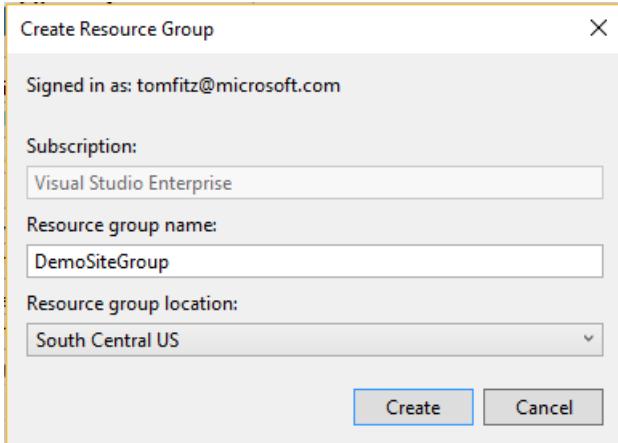
The **Deploy to Resource Group** dialog box appears.



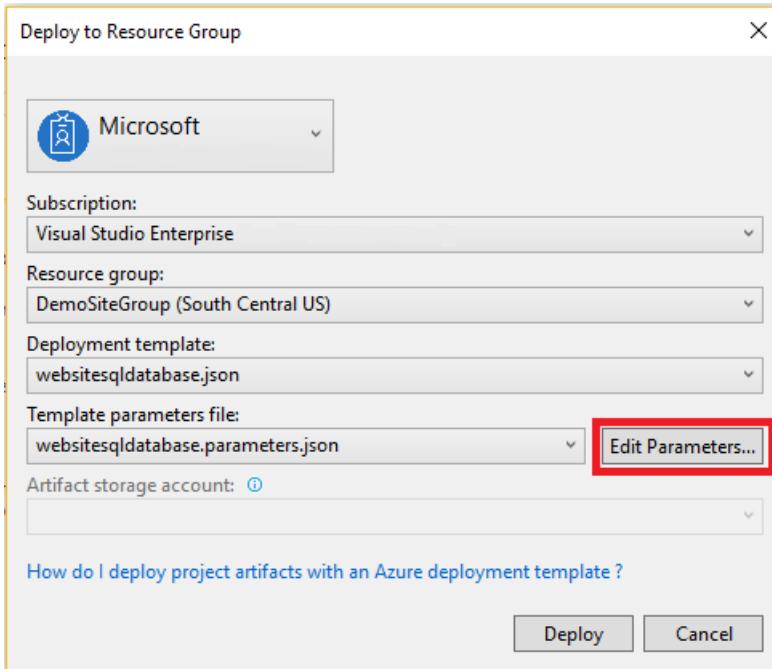
2. In the **Resource group** dropdown box, choose an existing resource group or create a new one. To create a resource group, open the **Resource Group** dropdown box and choose **Create New**.



The **Create Resource Group** dialog box appears. Give your group a name and location, and select the **Create** button.



3. Edit the parameters for the deployment by selecting the **Edit Parameters** button.



4. Provide values for the empty parameters and select the **Save** button. The empty parameters are **hostingPlanName**, **administratorLogin**, **administratorLoginPassword**, and **databaseName**.

hostingPlanName specifies a name for the [App Service plan](#) to create.

administratorLogin specifies the user name for the SQL Server administrator. Don't use common admin names like **sa** or **admin**.

The **administratorLoginPassword** specifies a password for SQL Server administrator. The **Save passwords as plain text in the parameters file** option isn't secure; therefore, don't select this option. Since the password isn't saved as plain text, you need to provide this password again during deployment.

databaseName specifies a name for the database to create.

Edit Parameters

The following parameter values will be used for this deployment:

Parameter Name	Value
hostingPlanName	DemoSitePlan
skuName	F1
skuCapacity	1
administratorLogin	DemoAdmin
administratorLoginPassword	*****
databaseName	DemoDatabase
collation	SQL_Latin1_General_CI_AS
edition	Basic
maxSizeBytes	1073741824
requestedServiceObjectiveName	Basic
storageType	Standard_LRS

Save passwords as plain text in the parameters file

Save **Cancel**

5. Choose the **Deploy** button to deploy the project to Azure. A PowerShell console opens outside of the Visual Studio instance. Enter the SQL Server administrator password in the PowerShell console when prompted. **Your PowerShell console may be hidden behind other items or minimized in the task bar.** Look for this console and select it to provide the password.

NOTE

Visual Studio may ask you to install the Azure PowerShell cmdlets. You need the Azure PowerShell cmdlets to successfully deploy resource groups. If prompted, install them. For more information, see [Install and configure Azure PowerShell](#).

6. The deployment may take a few minutes. 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 'websitesqldatabase.json' to resource group 'DemoSiteGroup'.
```

7. 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.

The screenshot shows the Microsoft Azure Resource groups page. On the left, there's a sidebar with icons for creating a new resource group, adding resources, and managing subscriptions. Below the sidebar is a search bar labeled 'Filter by name...'. The main area displays a list of '9 items' under the heading 'NAME'. The first two items, 'MyGroup' and 'DemoSiteGroup', are highlighted with red boxes.

NAME
MyGroup
DemoSiteGroup
ExportGroup

8. 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 provide a unique name.

The screenshot shows the details of the 'DemoSiteGroup' resource group. On the left, there's a navigation menu with options like Overview, Activity log, Access control (IAM), Tags, Quickstart, Resource costs, Deployments, Properties, Locks, and Automation script. The 'Overview' tab is selected and highlighted in blue. The main content area shows deployment details: 'Subscription name (change) Visual Studio Enterprise' and '1 Succeeded'. Below this is a list of deployed resources, each with a small icon and a unique name ending in a random string of characters. The storage account is named 'storagealkkpuznxyv6'.

NAME
DemoSitePlan
sqlserveralkkpuznxyv6
DemoDatabase
storagealkkpuznxyv6
webSitealkkpuznxyv6
webSitealkkpuznxyv6

9. If you make changes and want to redeploy your project, choose the existing resource group from the shortcut menu of Azure resource group project. On the shortcut menu, choose **Deploy**, and then choose the resource group you deployed.

The screenshot shows the context menu for the 'ResourceGroup1' project in Visual Studio. The 'Deploy' option is highlighted with a red box. A submenu is open under 'Deploy', showing a list of resource groups. The 'DemoSiteGroup' entry is also highlighted with a red box.

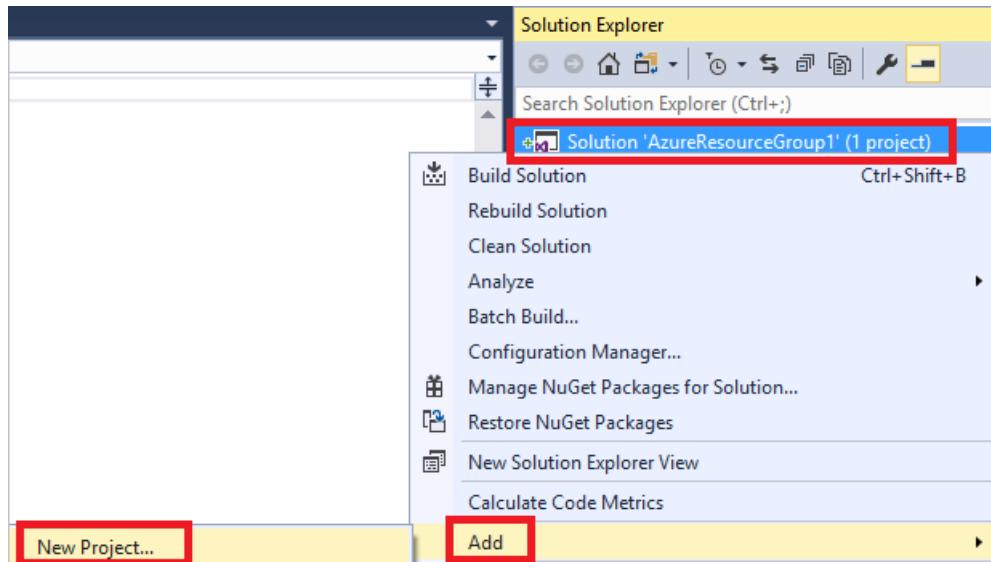
- Deploy
- Validate
- Build

- DemoSiteGroup

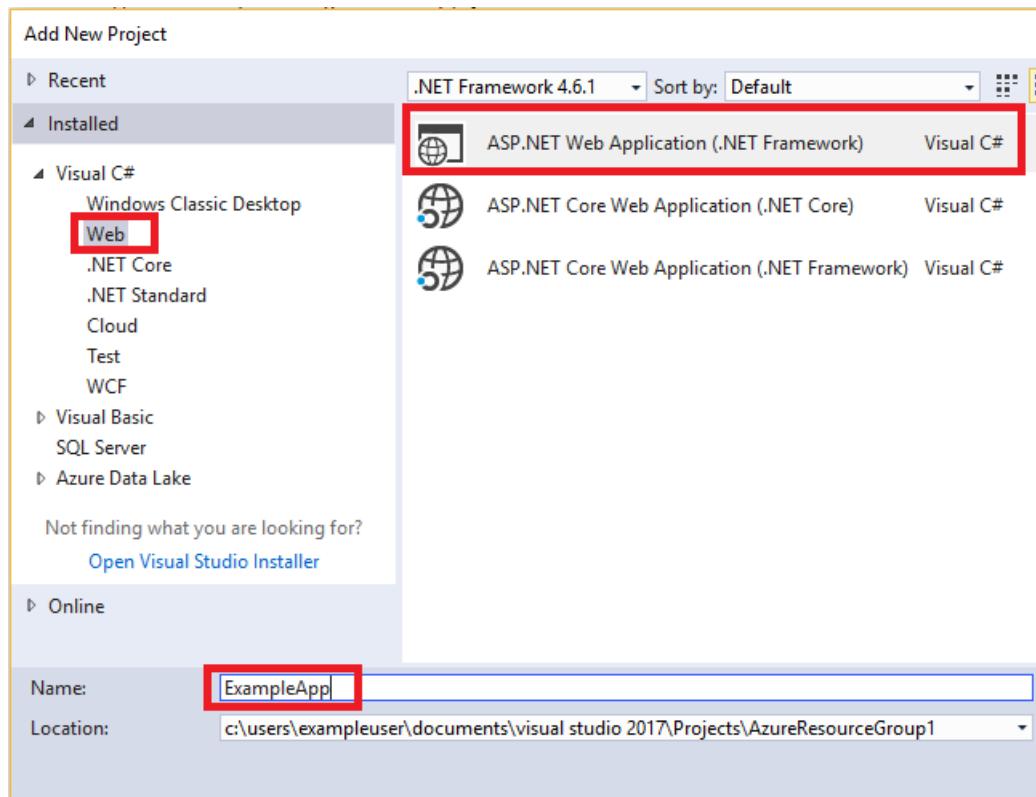
Deploy code with your infrastructure

At this point, you've deployed the infrastructure for your app, but there's no actual code deployed with the project. This article shows how to deploy a web app and SQL Database tables during deployment. If you're deploying a Virtual Machine instead of a web app, you want to run some code on the machine as part of deployment. The process for deploying code for a web app or for setting up a Virtual Machine is almost the same.

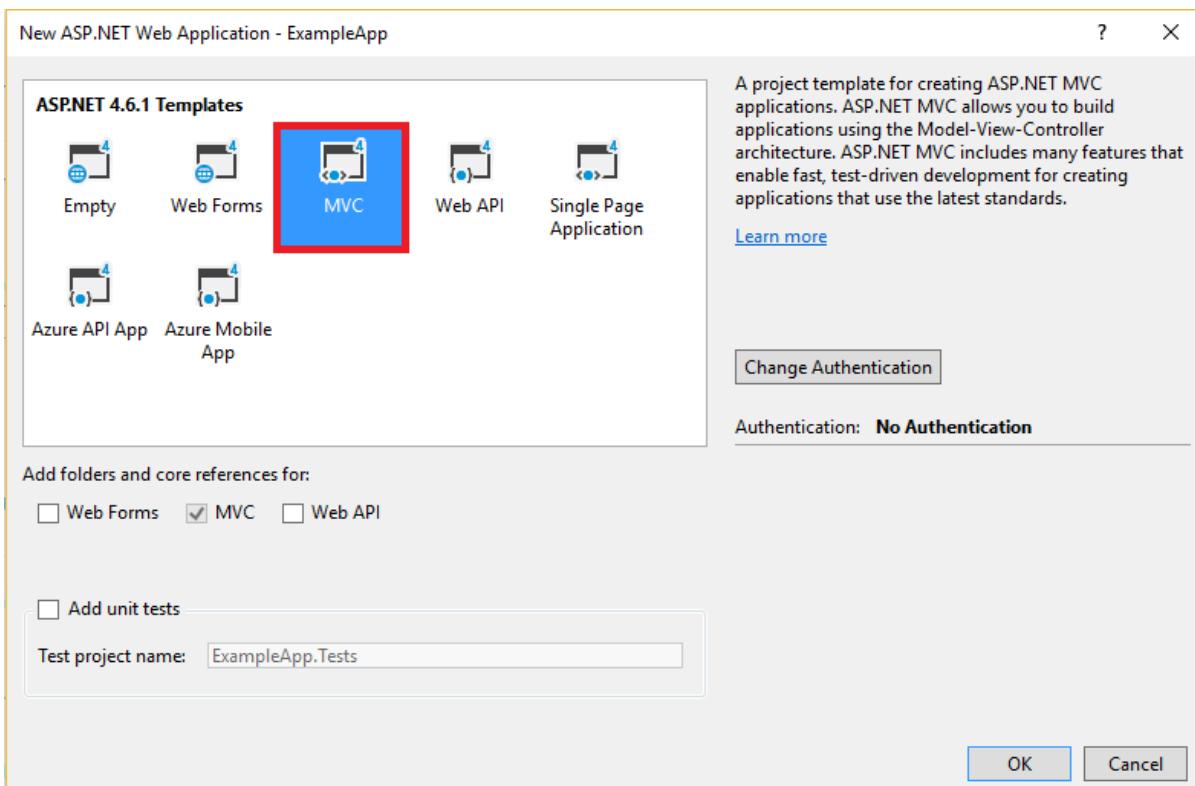
1. Add a project to your Visual Studio solution. Right-click the solution, and select **Add > New Project**.



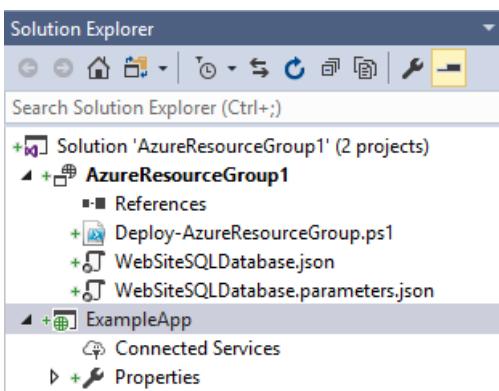
2. Add an **ASP.NET Web Application**.



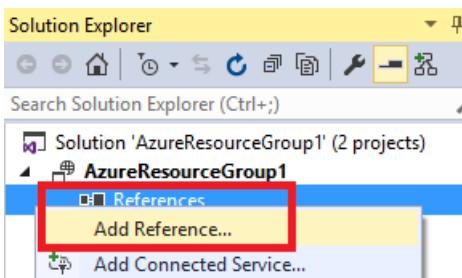
3. Select **MVC**.



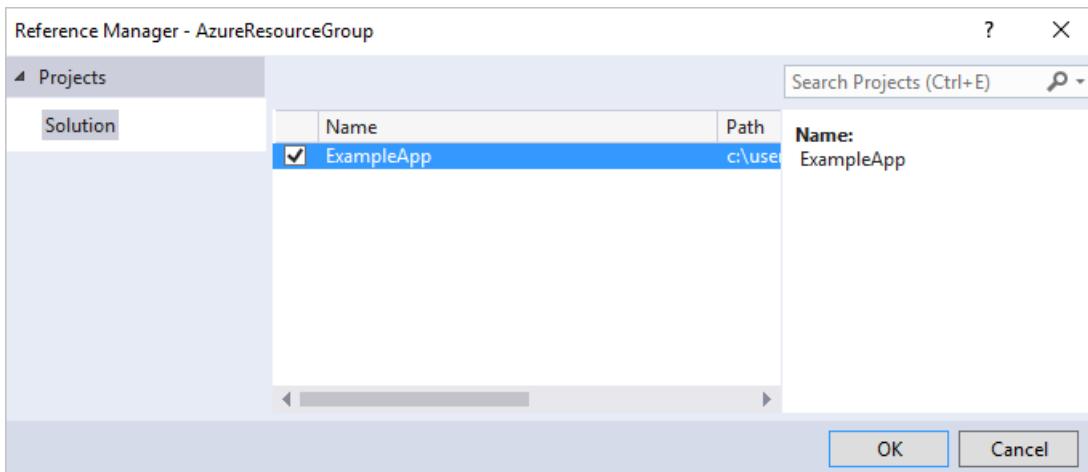
4. After Visual Studio creates your web app, you see both projects in the solution.



5. Now, you need to make sure your resource group project is aware of the new project. Go back to your resource group project (AzureResourceGroup1). Right-click **References** and select **Add Reference...**.



6. 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 set three key properties. You see these properties in the **Properties** window for the reference.

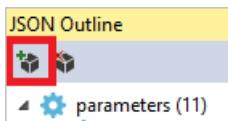
Additional Properties	PackageLocation=..\AzureResourceGroup1\obj\Debug\ProjectReferences\\ExampleApp\package.zip
Include File Path	obj\Debug\ProjectReferences\\ExampleApp\package.zip
Include Targets	Build;Package

The properties are:

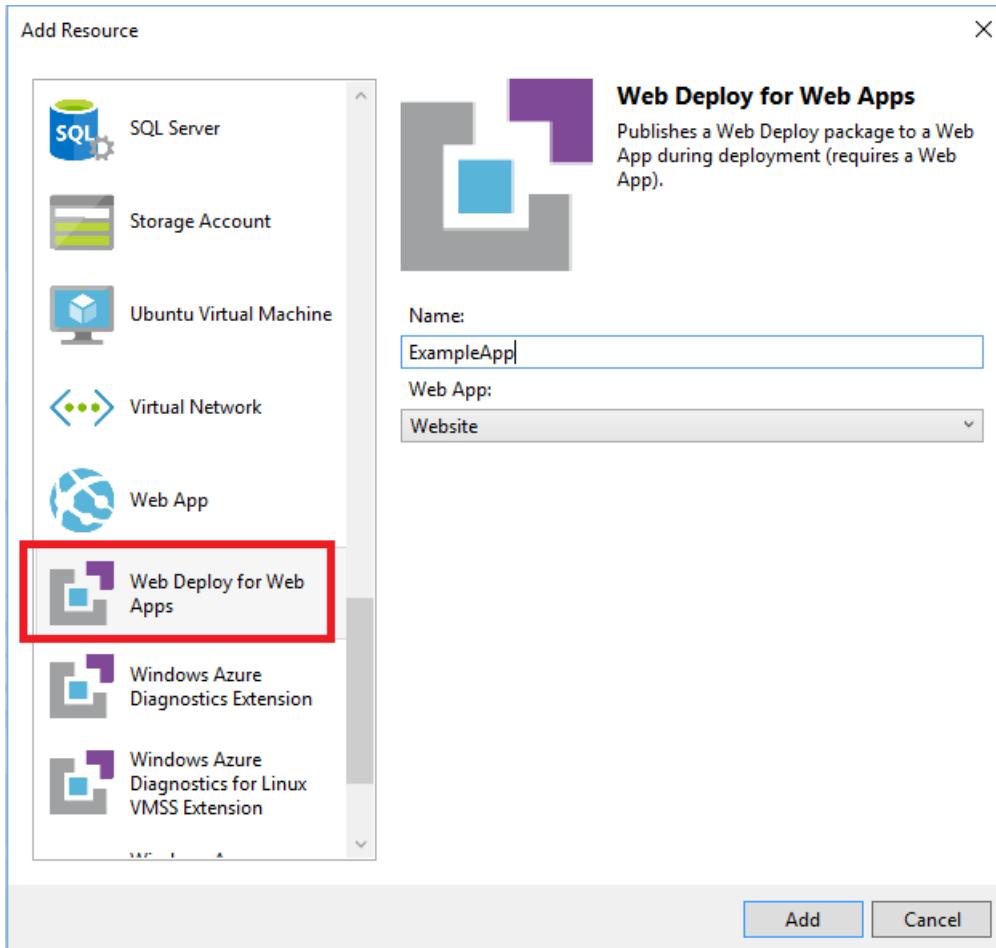
- The **Additional Properties** has the web deployment package staging location that is pushed to the Azure Storage. Note the folder (ExampleApp) and file (package.zip). You need to know these values because you provide them as parameters when deploying the app.
- The **Include File Path** has the path where the package is created. The **Include Targets** has the command that deployment executes.
- The default value of **Build;Package** enables the deployment to build and create a web deployment package (package.zip).

You don't need a publish profile as the deployment gets the necessary information from the properties to create the package.

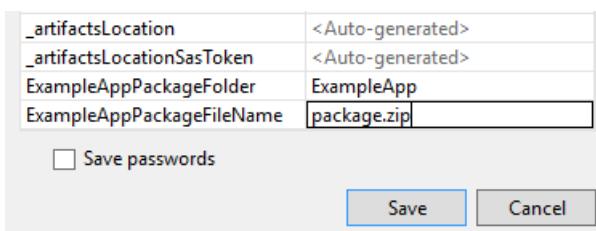
7. Go back to WebSiteSQLDatabase.json and add a resource to the template.



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



9. Redeploy your resource group project to the resource group. This time there are some new parameters. You don't need to provide values for **_artifactsLocation** or **_artifactsLocationSasToken** because Visual Studio automatically generates those values. However, you have to set the folder and file name to the path that contains the deployment package (shown as **ExampleAppPackageFolder** and **ExampleAppPackageName** in the following image). Provide the values you saw earlier in the reference properties (**ExampleApp** and **package.zip**).



For the **Artifact storage account**, select the one deployed with this resource group.

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

Essentials ^	
Resource group	URL
DemoSiteGroup	http://websitelg52msdchkveo.azurewebsite...
Status	App Service plan/pricing tier
Running	DemoSitePlan (Free)
Location	FTP/Deployment username
West US	webSitelg52msdchkveo\tomfitz
Subscription name	FTP hostname
Windows Azure MSDN - Visual Studio Ulti...	ftp://waws-prod-bay-065.ftp.azurewebsites...
Subscription ID	FTPS hostname
	ftps://waws-prod-bay-065.ftp.azurewebsite...

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

The screenshot shows a deployed ASP.NET application. The URL is highlighted in red as <http://websitelg52msdchkveo.azurewebsite...>. The application's home page is displayed, featuring a large 'ASP.NET' title, a descriptive subtitle about the framework, and a 'Learn more »' button.

Add an operations dashboard to your deployment

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.

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

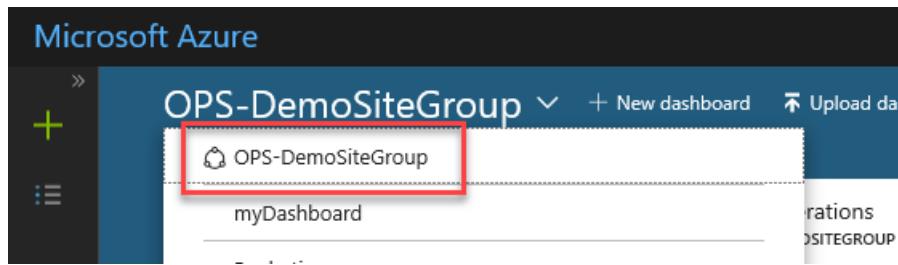
```
,{
  "properties": {
    "lenses": {
      "0": {
        "order": 0,
        "parts": {
          "0": {
            "position": {
              "x": 0,
              "y": 0,
              "colSpan": 4,
              "rowSpan": 6
            },
            "metadata": {
              "inputs": [
                {
                  "name": "resourceGroup",
                  "isOptional": true
                },
                {
                  "name": "id",
                  "value": "[resourceGroup().id]"
                }
              ]
            }
          }
        }
      }
    }
  }
}
```

```

        "isOptional": true
    }
],
"type": "Extension/HubsExtension/PartType/ResourceGroupPinnedPart"
}
},
"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]"
                }
            }
        }
    }
},
"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)]"
}
}
}

```

- Redeploy your resource group. Look at your dashboard on the Azure portal, and notice the shared dashboard has been added to your list of choices.



3. Select the dashboard.

Resources
DEMOBSITEGROUP

	DemoSitePlan App Service plan
	sqlservervrbvdxxvs3dxo SQL server
	DemoDatabase SQL database
	storagevrbvdxxvs3dxo Storage account
	webSitevrbvdxxvs3dxo Application Insights
	webSitevrbvdxxvs3dxo App Service

Operations
DEMOBSITEGROUP

Customizations

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 [A](#) 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](#).

Next steps

- To learn more about templates, see [Authoring Azure Resource Manager templates](#).

Tutorial: create an Azure Resource Manager template for deploying an encrypted storage account

7/20/2018 • 4 minutes to read • [Edit Online](#)

Learn how to find information to complete an Azure Resource Manager template.

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

This tutorial covers the following tasks:

- Open a Quickstart template
- Understand the template format
- Use parameters in template
- Use variables in 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](#).
- Resource Manager Tools extension. To install, see [Install the Resource Manager Tools extension](#).

Open a Quickstart template

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 format

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

```
1  {
2    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
3    "contentVersion": "1.0.0.0",
4    "parameters": { ...
25   },
26   "variables": { ...
28   },
29   "resources": [ ...
41   ],
42   "outputs": { ...
47   }
48 }
```

- **\$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.

Use parameters in template

Parameters enable you to customize the deployment by providing values that are tailored for a particular environment. You use the parameters defined in the template when setting values for the storage account.

```
{
  "$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": { ... },
  "resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "name": "[variables('storageAccountName')]",
      "apiVersion": "2016-01-01",
      "location": "[parameters('location')]",
      "sku": {
        "name": "[parameters('storageAccountType')]"
      },
      "kind": "Storage",
      "properties": {}
    }
  ],
  "outputs": { ... }
}
```

In this template, two parameters are defined. Notice a template function is used in location.defaultValue:

```
"defaultValue": "[resourceGroup().location]",
```

The resourceGroup() function returns an object that represents the current resource group. For a list of template functions, see [Azure Resource Manager template functions](#).

To use the parameters defined in the template:

```
"location": "[parameters('location')]",
"name": "[parameters('storageAccountType')]"
```

Use variables in template

Variables allow you to construct values that can be used throughout your template. Variables help reducing the complexity of the templates.

```
{
  "$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": "[variables('storageAccountName')]",
      "apiVersion": "2016-01-01",
      "location": "[parameters('location')]",
      "sku": {
        "name": "[parameters('storageAccountType')]"
      },
      "kind": "Storage",
      "properties": {}
    }
  ],
  "outputs": { ... }
}
```

This template defines one variable `storageAccountName`. In the definition, two template functions are used:

- **concat()**: concatenates strings. For more information, see [concat](#).
- **uniqueString()**: creates a deterministic hash string based on the values provided as parameters. Each Azure storage account must have an unique name across of all Azure. This function provides an unique string. For more string functions, see [String functions](#).

To use the variable defined in the template:

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

Edit the template

To find the storage account encryption-related configuration, you can use the template reference of Azure Storage account.

1. Browse to [Azure Templates](#).
2. From the TOC on the left, select **Reference->Storage->Storage Accounts**. The page contains the information for defining a Storage Account information.
3. Explore the encryption-related information.
4. Inside the properties element of the storage account resource definition, add the following json:

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

This part enables the encryption function of the blob storage service.

The final resources element looks like:

```

"resources": [
  {
    "type": "Microsoft.Storage/storageAccounts",
    "name": "[variables('storageAccountName')]",
    "apiVersion": "2016-01-01",
    "location": "[parameters('location')]",
    "sku": {
      "name": "[parameters('storageAccountType')]"
    },
    "kind": "Storage",
    "properties": {
      "encryption": {
        "keySource": "Microsoft.Storage",
        "services": {
          "blob": {
            "enabled": true
          }
        }
      }
    }
  }
],

```

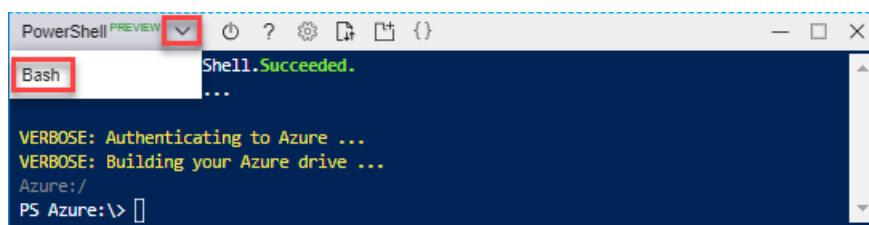
Deploy the template

There are many methods for deploying templates. In this tutorial, you use the Cloud shell from the Azure portal. The Cloud shell supports both Azure CLI and Azure PowerShell. The instructions provided here use CLI.

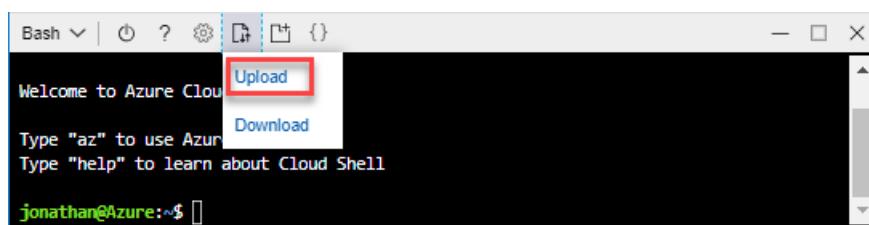
1. Sign in to the [Azure portal](#)
2. Select **Cloud Shell** from the upper right corner as shown in the following image:



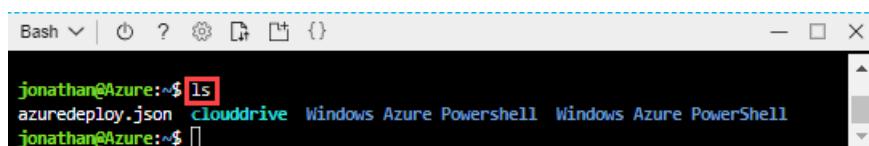
3. Select the down arrow and then select **Bash** if it is not Bash. You use Azure CLI in this tutorial.



4. Select **Restart** to restart the shell.
5. Select **Upload/download files**, and then select **Upload**.



6. Select the file you saved earlier in the tutorial. The default name is **azuredeploy.json**.
7. From the Cloud shell, run the **ls** command to verify the file is uploaded successfully. You can also use the **cat** command to verify the template content.



8. From the Cloud shell, run the following commands:

```
az group create --name <ResourceGroupName> --location <AzureLocation>  
  
az group deployment create --name <DeploymentName> --resource-group <ResourceGroupName> --template-file  
azuredeploy.json
```

Here is the screenshot of a sample deployment:

On the screenshot, these values are used:

- <**ResourceGroupName**>: myresourcegroup0719. There are two appearances of the parameter. Make sure to use the same value.
 - <**AzureLocation**>: eastus2
 - <**DeployName**>: mydeployment0719
 - <**TemplateFile**>: azuredeploy.json

From the screenshot output, the storage account name is `fhqbfslkdqdsstandardsa`.

9. Run the following PowerShell command to list the newly created storage account:

```
az storage account show --resource-group <ResourceGroupName> --name <StorageAccountName>
```

You shall see an output similar to the following screenshot which indicates encryption has been enabled for the blob storage.

```
jonathan@Azure:~$ az storage account show --resource-group myresourcegroup0719 --name fhqbfslkdqdsstandardsa
{
  "accessTier": null,
  "creationTime": "2018-07-19T21:50:43.531552+00:00",
  "customDomain": null,
  "enableHttpsTrafficOnly": false,
  "encryption": {
    "keySource": "Microsoft.Storage",
    "keyVaultProperties": null,
    "services": {
      "blob": {
        "enabled": true,
        "lastEnabledTime": "2018-07-19T21:50:43.672151+00:00"
      },
      "file": {
        "enabled": true,
        "lastEnabledTime": "2018-07-19T21:50:43.672151+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. The template used in this tutorial only contains one Azure resource. In the next tutorial, you develop a template with multiple resources. Some of the resources have dependent resources.

[Create multiple resources](#)

Tutorial: create Azure Resource Manager templates with dependent resources

7/20/2018 • 3 minutes to read • [Edit Online](#)

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

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](#).

- Open a quickstart template
- Explore the template
- Deploy the template
- Clean up resources

The instructions in this tutorial create a virtual machine, a virtual network, and some other dependent resources.

Prerequisites

To complete this article, you need:

- [Visual Studio Code](#).
- Resource Manager Tools extension. See [Install the 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 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:

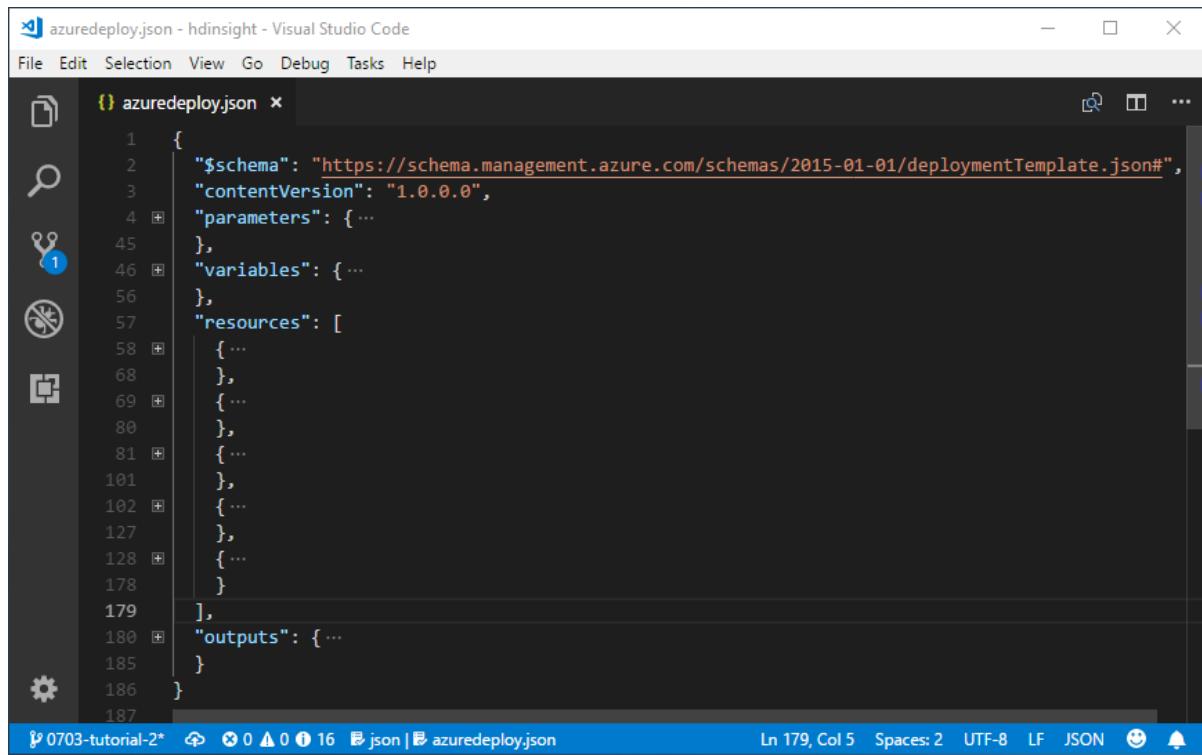
```
https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-vm-simple-windows/azuredeploy.json
```

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

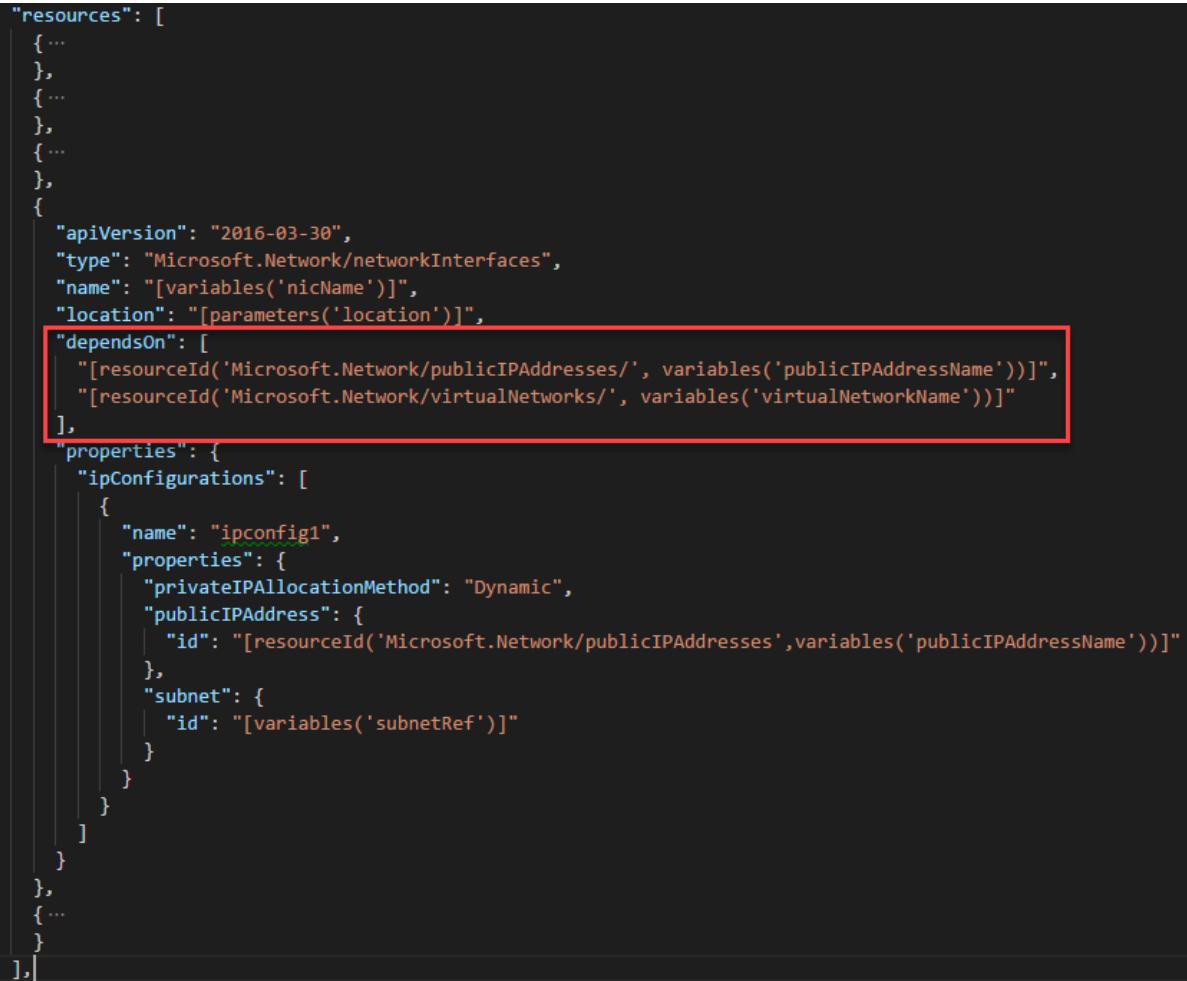
1. From Visual Studio Code, collapse the elements until you only see the first-level elements and the second-level elements inside **resources**:



```
1  {
2    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
3    "contentVersion": "1.0.0.0",
4    "parameters": { ... },
5    "variables": { ... },
6    "resources": [
7      { ... },
8      { ... },
9      { ... },
10     { ... },
11     { ... },
12     { ... },
13     { ... },
14     { ... },
15     { ... },
16     { ... },
17     { ... },
18     { ... },
19     { ... },
20     { ... },
21     { ... },
22     { ... },
23     { ... },
24     { ... },
25     { ... },
26     { ... },
27     { ... },
28     { ... },
29     { ... },
30     { ... },
31     { ... },
32     { ... },
33     { ... },
34     { ... },
35     { ... },
36     { ... },
37     { ... },
38     { ... },
39     { ... },
40     { ... },
41     { ... },
42     { ... },
43     { ... },
44     { ... },
45     { ... },
46     { ... },
47     { ... },
48     { ... },
49     { ... },
50     { ... },
51     { ... },
52     { ... },
53     { ... },
54     { ... },
55     { ... },
56     { ... },
57     { ... },
58     { ... },
59     { ... },
60     { ... },
61     { ... },
62     { ... },
63     { ... },
64     { ... },
65     { ... },
66     { ... },
67     { ... },
68     { ... },
69     { ... },
70     { ... },
71     { ... },
72     { ... },
73     { ... },
74     { ... },
75     { ... },
76     { ... },
77     { ... },
78     { ... },
79     { ... },
80     { ... },
81     { ... },
82     { ... },
83     { ... },
84     { ... },
85     { ... },
86     { ... },
87     { ... },
88     { ... },
89     { ... },
90     { ... },
91     { ... },
92     { ... },
93     { ... },
94     { ... },
95     { ... },
96     { ... },
97     { ... },
98     { ... },
99     { ... },
100    { ... },
101    { ... },
102    { ... },
103    { ... },
104    { ... },
105    { ... },
106    { ... },
107    { ... },
108    { ... },
109    { ... },
110    { ... },
111    { ... },
112    { ... },
113    { ... },
114    { ... },
115    { ... },
116    { ... },
117    { ... },
118    { ... },
119    { ... },
120    { ... },
121    { ... },
122    { ... },
123    { ... },
124    { ... },
125    { ... },
126    { ... },
127    { ... },
128    { ... },
129    { ... },
130    { ... },
131    { ... },
132    { ... },
133    { ... },
134    { ... },
135    { ... },
136    { ... },
137    { ... },
138    { ... },
139    { ... },
140    { ... },
141    { ... },
142    { ... },
143    { ... },
144    { ... },
145    { ... },
146    { ... },
147    { ... },
148    { ... },
149    { ... },
150    { ... },
151    { ... },
152    { ... },
153    { ... },
154    { ... },
155    { ... },
156    { ... },
157    { ... },
158    { ... },
159    { ... },
160    { ... },
161    { ... },
162    { ... },
163    { ... },
164    { ... },
165    { ... },
166    { ... },
167    { ... },
168    { ... },
169    { ... },
170    { ... },
171    { ... },
172    { ... },
173    { ... },
174    { ... },
175    { ... },
176    { ... },
177    { ... },
178    { ... },
179    { ... },
180    { ... },
181    { ... },
182    { ... },
183    { ... },
184    { ... },
185    { ... },
186    { ... },
187    { ... }
```

There are five resources defined by the template.

2. Expand the fourth element:



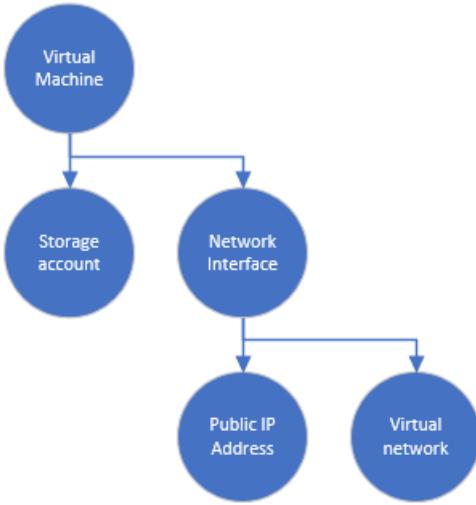
```
"resources": [
  { ... },
  { ... },
  { ... },
  { ... },
  { ... },
  {
    "apiVersion": "2016-03-30",
    "type": "Microsoft.Network/networkInterfaces",
    "name": "[variables('nicName')]",
    "location": "[parameters('location')]",
    "dependsOn": [
      "[resourceId('Microsoft.Network/publicIPAddresses/', 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. In this example, this resource is a networkInterface. It depends on two other resources:

- publicIPAddress

- virtualNetwork
3. Expand the fifth element. This resource is a virtual machine. It depends on two other resources:
- storageAccount
 - networkInterface

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

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

1. Sign in to the [Azure portal](#)
2. Select **Cloud Shell** from the upper right corner as shown in the following image:

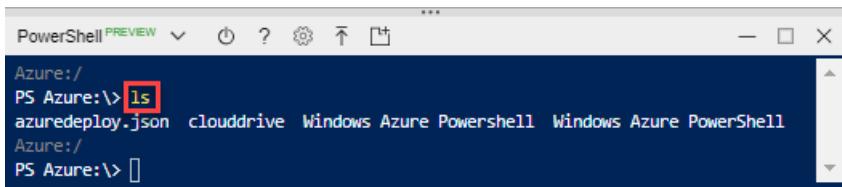


3. Select **PowerShell** from the upper left corner of the Cloud shell. You use PowerShell in this tutorial.
4. Select **Restart**
5. Select **Upload file** from the Cloud shell:

A screenshot of the Azure Cloud Shell PowerShell window. The title bar says 'PowerShell PREVIEW'. The main area shows the command output:
Requesting a Cloud Shell.**Succeeded**.
Connecting terminal...
VERBOSE: Authenticating to Azure ...
VERBOSE: Building your Azure drive ...
Azure:/
PS Azure:> []

6. Select the file 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 will be overwritten without any notification.
7. From the Cloud shell, run the following command to verify the file is uploaded successfully.

```
ls
```



Azure:/
PS Azure:\> ls
azuredeploy.json clouddrive Windows Azure Powershell Windows Azure PowerShell
Azure:/
PS Azure:\>

The file name shown on the screenshot is `azuredeploy.json`.

8. From the Cloud shell run the following command to verify the content of the JSON file:

```
cat azuredeploy.json
```

9. From the Cloud shell, run the following PowerShell commands:

```
$resourceGroupName = "<Enter the resource group name>"  
$location = "<Enter the Azure location>"  
$vmAdmin = "<Enter the admin username>"  
$vmPassword = "<Enter the password>"  
$dnsLabelPrefix = "<Enter the prefix>"  
  
New-AzureRmResourceGroup -Name $resourceGroupName -Location $location  
$vmPW = ConvertTo-SecureString -String $vmPassword -AsPlainText -Force  
New-AzureRmResourceGroupDeployment -Name mydeployment0710 -ResourceGroupName $resourceGroupName `  
-TemplateFile azuredeploy.json -adminUsername $vmAdmin -adminPassword $vmPW `  
-dnsLabelPrefix $dnsLabelPrefix
```

Here is the screenshot for a sample deployment:

```

PowerShell PREVIEW | ⌁ ? ⌂ ⌄
Azure:/ PS Azur: \> $resourceGroupName = "myresourcegroup0710"
Azure:/ PS Azur: \> $location = "eastus2"
Azure:/ PS Azur: \> $vmAdmin = "JohnDole"
Azure:/ PS Azur: \> $vmPassword = "Pass@word123"
Azure:/ PS Azur: \> $dnsLabelPrefix = "myvm0710"
Azure:/ PS Azur: \>
Azure:/ PS Azur: \> New-AzureRmResourceGroup -Name $resourceGroupName -Location $location

ResourceGroupName : myresourcegroup0710
Location        : eastus2
ProvisioningState : Succeeded
Tags            :
ResourceId      : /subscriptions/ <Azure Subscription ID> /myresourcegroup0710

Azure:/ PS Azur: \> $vmPW = ConvertTo-SecureString -String $vmPassword -AsPlainText -Force
Azure:/ PS Azur: \> New-AzureRmResourceGroupDeployment -Name mydeployment0710 -ResourceGroupName myresourcegroup0710 ` 
>> -TemplateFile azuredeploy.json -adminUsername $vmAdmin -adminPassword $vmPW ` 
>> -dnsLabelPrefix $dnsLabelPrefix

DeploymentName      : mydeployment0710
ResourceGroupName   : myresourcegroup0710
ProvisioningState   : Succeeded
Timestamp          : 7/10/18 10:18:03 PM
Mode               : Incremental
TemplateLink       :
Parameters         :
    Name          Type           Value
    ======  ======  ======
    adminUsername String        JohnDole
    adminPassword SecureString
    dnsLabelPrefix String        myvm0710
    windowsOSVersion String      2016-Datacenter
    location       String        eastus2

Outputs           :
    Name          Type           Value
    ======  ======  ======
    hostname      String        myvm0710.eastus2.cloudapp.azure.com

DeploymentLogLevel :

```

On the screenshot, these values are used:

- **\$resourceGroupName:** myresourcegroup0710.
- **\$location:** eastus2
- **<DeployName>:** mydeployment0710
- **<TemplateFile>:** azuredeploy.json
- **Template parameters:**
 - **adminUsername:** JohnDole
 - **adminPassword:** Pass@word123
 - **dnsLabelPrefix:** myvm0710

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

```
Get-AzureRmVM -Name SimpleWinVM -ResourceGroupName <ResourceGroupName>
```

The virtual machine name is hard-coded as **SimpleWinVM** inside 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 develop and deploy a template to create a virtual machine, a virtual network, and the dependent resources. To learn more about templates, see [Understand the structure and syntax of Azure Resource Manager Templates](#).

Resource providers and types

5/21/2018 • 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 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 perform these steps through the portal, PowerShell, or Azure CLI.

PowerShell

To see all resource providers in Azure, and the registration status for your subscription, use:

```
Get-AzureRmResourceProvider -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 perform the `/register/action` operation for the resource provider. This operation is included in the Contributor and Owner roles.

```
Register-AzureRmResourceProvider -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 cannot 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-AzureRmResourceProvider -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-AzureRmResourceProvider -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-AzureRmResourceProvider -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-AzureRmResourceProvider -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 perform 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 cannot 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:

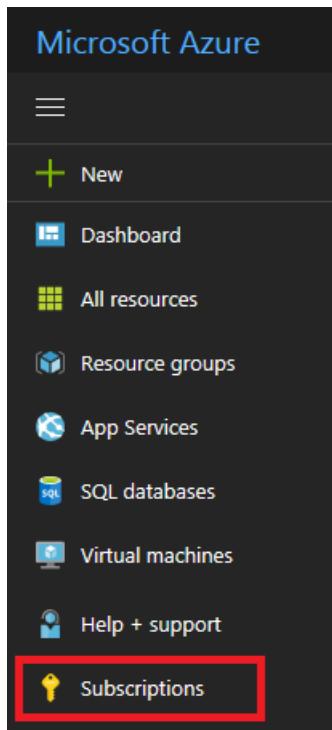
```
az provider show --namespace Microsoft.Batch --query "resourceTypes[?resourceType=='batchAccounts'].locations | [0]" --out table
```

Which returns:

```
Result
-----
West Europe
East US
East US 2
West US
...
```

Portal

To see all resource providers in Azure, and the registration status for your subscription, select **Subscriptions**.



Choose the subscription to view.

The image shows the 'Subscriptions' management interface. It features a search bar at the top with the placeholder 'Search to filter items...'. Below the search bar is a table with a single row labeled 'SUBSCRIPTION'. The row contains a blue circular icon with a white 'H' and the text 'Visual Studio Enterprise'. This row is also highlighted with a red rectangular box.

Select **Resource providers** and view the list of available resource providers.

Visual Studio Enterprise - Resource providers

Subscription

Search (Ctrl+ /)

Diagnose and solve problems

SETTINGS

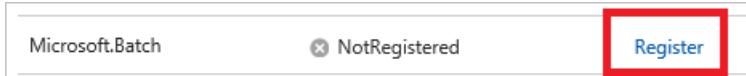
- Programmatic deployment
- Resource groups
- Resources
- Usage + quotas
- Policies
- Management certificates
- My permissions
- Resource providers**
- Properties

Refresh

Search to filter resource providers...

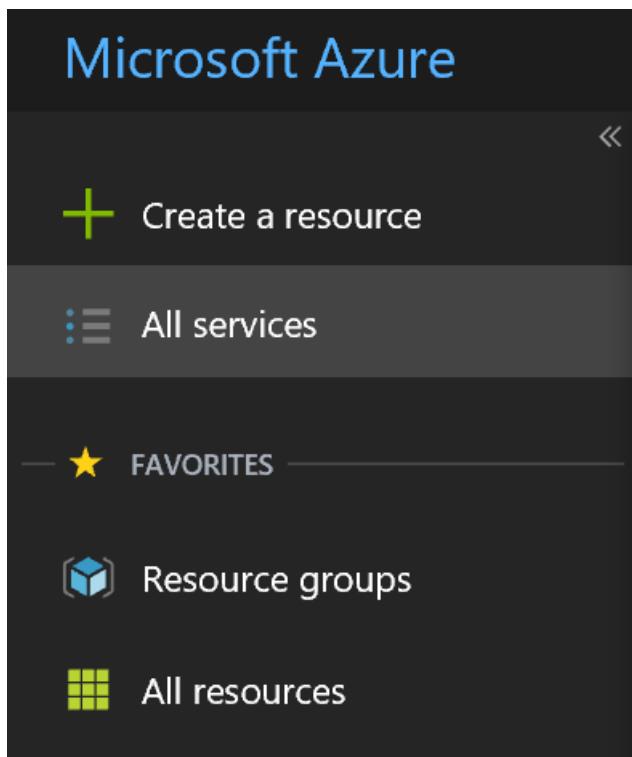
PROVIDER	STATUS	Re-register	Unregister
Microsoft.Cdn	Registered	Re-register	Unregister
Microsoft.ClassicStorage	Registered	Re-register	Unregister
microsoft.insights	Registered	Re-register	Unregister
Microsoft.Network	Registered	Re-register	Unregister
Microsoft.OperationalInsights	Registered	Re-register	Unregister
Microsoft.Security	Registered	Re-register	Unregister
Microsoft.Sql	Registered	Re-register	Unregister
Microsoft.Storage	Registered	Re-register	Unregister
Microsoft.Web	Registered	Re-register	Unregister
84codes.CloudAMQP	NotRegistered	Register	
AppDynamics.APM	NotRegistered	Register	
Aspera.Transfers	NotRegistered	Register	
Auth0.Cloud	NotRegistered	Register	

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 perform 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**.

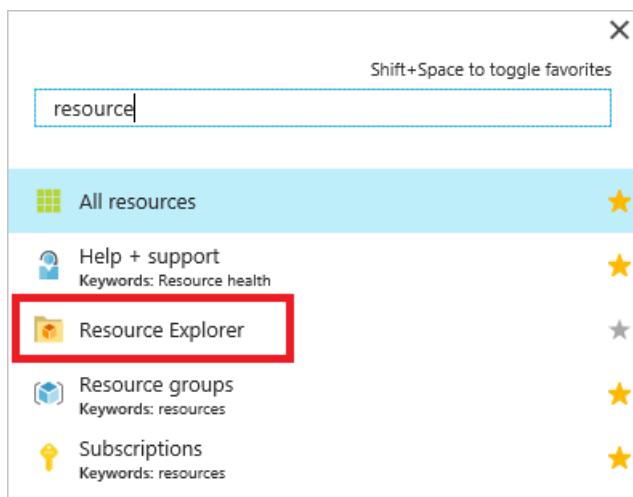


You cannot unregister a resource provider when you still have resource types from that resource provider in your subscription.

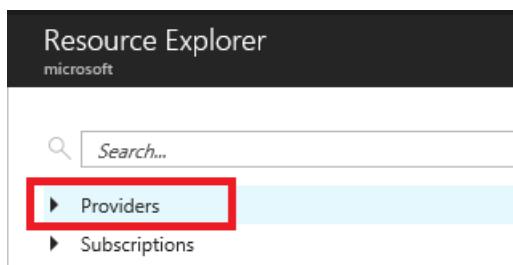
To see information for a particular resource provider, select **All services**.



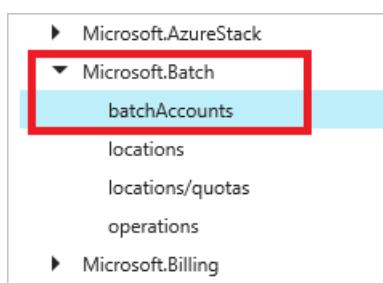
Search for **Resource Explorer** and select it from the available options.



Select **Providers**.



Select the resource provider and resource type that you want to view.



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.

```
batchAccounts (Response Time 58ms)

/providers/Microsoft.Batch?api-version=2014-04-01-preview

1 {
2     "namespace": "Microsoft.Batch",
3     "resourceTypes": [
4         {
5             "resourceType": "batchAccounts",
6             "locations": [
7                 "West Europe",
8                 "East US",
9                 "East US 2",
10                "West US",
11                "North Central US",
12            ]
13        }
14    ]
15 }
```

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.

```
37
38     "apiVersions": [
39         "2017-05-01",
40         "2017-01-01",
41         "2015-12-01",
42         "2015-09-01",
43         "2015-07-01",
44         "2014-05-01-privatepreview"
45     ]
46 },
```

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](#).

Organize your resources with Azure management groups

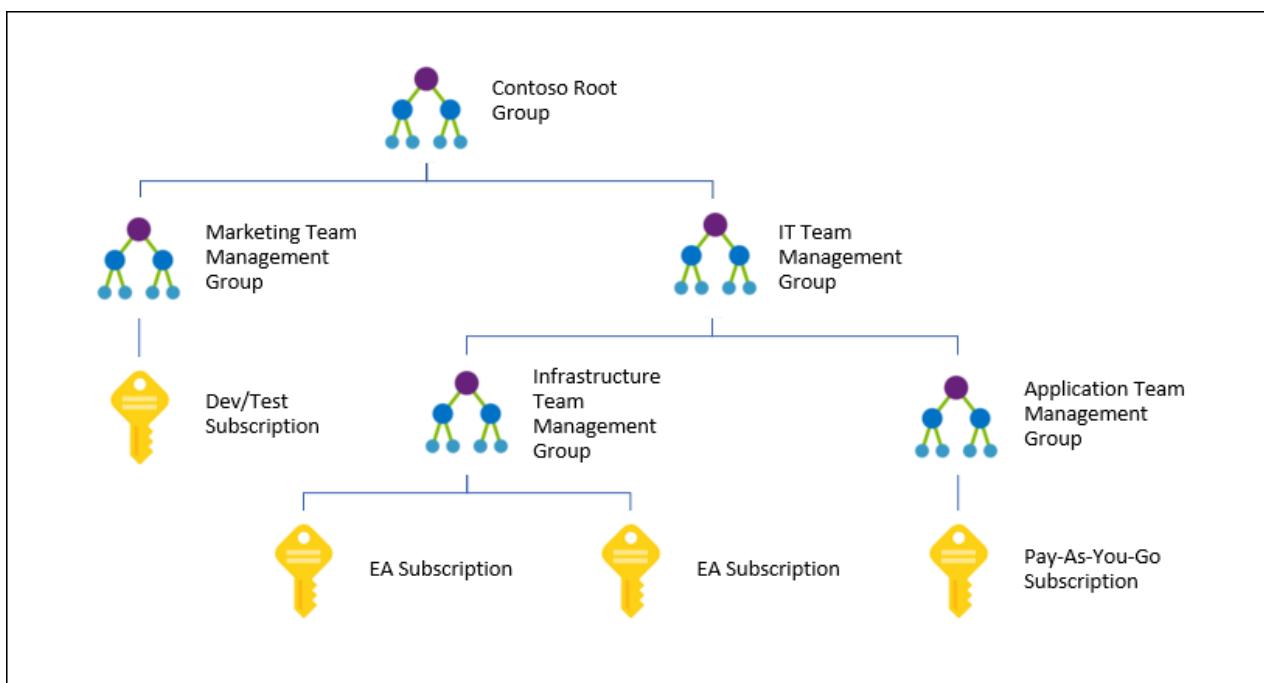
8/2/2018 • 5 minutes to read • [Edit Online](#)

If your organization has many subscriptions, you may need a way to efficiently manage access, policies, and compliance for those subscriptions. Azure management groups provide a level of scope above subscriptions. You organize subscriptions into containers called "management groups" and apply your governance conditions to the management groups. All subscriptions within a management group automatically inherit the conditions applied to the management group. Management groups give you enterprise-grade management at a large scale no matter what type of subscriptions you might have.

For example, you can apply policies to a management group that limits the regions available for virtual machine (VM) creation. This policy would be applied to all management groups, subscriptions, and resources under that management group by only allowing VMs to be created in that region.

Hierarchy of management groups and subscriptions

You can build a flexible structure of management groups and subscriptions to organize your resources into a hierarchy for unified policy and access management. The following diagram shows an example of creating a hierarchy for governance using management groups.



By creating a hierarchy like this example you can apply a policy, for example, VM locations limited to US West Region on the group "Infrastructure Team management group" to enable internal compliance and security policies. This policy will inherit onto both EA subscriptions under that management group and will apply to all VMs under those subscriptions. As this policy inherits from the management group to the subscriptions, this security policy cannot be altered by the resource or subscription owner allowing for improved governance.

Another scenario where you would use management groups is to provide user access to multiple subscriptions. By moving multiple subscriptions under that management group, you have the ability create one RBAC assignment on the management group, which will inherit that access to all the subscriptions. Without the need to script RBAC assignments over multiple subscriptions, one assignment on the management group can enable

users to have access to everything they need.

Important facts about management groups

- 10,000 management groups can be supported in a single directory.
- A management group tree can support up to six levels of depth.
 - This limit doesn't include the Root level or the subscription level.
- Each management group and subscription can only support one parent.
- Each management group can have multiple children.
- All subscriptions and management groups are contained within a single hierarchy in each directory. See [Important facts about the Root management group](#) for exceptions during the Preview.

Root management group for each directory

Each directory is given a single top-level management group called the "Root" management group. This root management group is built into the hierarchy to have all management groups and subscriptions fold up to it. This Root management group allows for global policies and RBAC assignments to be applied at the directory level. The [Directory Administrator needs to elevate themselves](#) to be the owner of this root group initially. Once the administrator is the owner of the group, they can assign any RBAC role to other directory users or groups to manage the hierarchy.

Important facts about the Root management group

- The root management group's name and ID are given by default. The display name can be updated at any time to show different within the Azure portal.
 - The name will be "Tenant root group".
 - The ID will be the Azure Active Directory ID.
- The root management group can't be moved or deleted, unlike other management groups.
- All subscriptions and management groups fold up to the one root management group within the directory.
 - All resources in the directory fold up to the root management group for global management.
 - New subscriptions are automatically defaulted to the root management group when created.
- All Azure customers can see the root management group, but not all customers have access to manage that root management group.
 - Everyone who has access to a subscription can see the context of where that subscription is in the hierarchy.
 - No one is given default access to the root management group. Directory Global Administrators are the only users that can elevate themselves to gain access. Once they have access, the directory administrators can assign any RBAC role to other users to manage.

NOTE

If your directory started using the management groups service before 6/25/2018, your directory might not be set up with all subscriptions in the hierarchy. The management group team is retroactively updating each directory that started using management groups in the Public Preview prior to that date within July/August 2018. All subscriptions in the directories will be made children under the root management group prior.

If you have questions on this retroactive process, contact: managementgroups@microsoft.com

Initial setup of management groups

When any user starts using management groups, there's an initial setup process that happens. The first step is the root management group is created in the directory. Once this group is created, all existing subscriptions that exist in the directory are made children of the root management group. The reason for this process is to make sure

there's only one management group hierarchy within a directory. The single hierarchy within the directory allows administrative customers to apply global access and policies that other customers within the directory can't bypass. Anything assigned on the root will apply across all management groups, subscriptions, resource groups, and resources within the directory by having one hierarchy within the directory.

IMPORTANT

Any assignment of user access or policy assignment on the root management group **applies to all resources within the directory**. Because of this, all customers should evaluate the need to have items defined on this scope. User access and policy assignments should be "Must Have" only at this scope.

Management group access

Azure management groups support [Azure Role-Based Access Control \(RBAC\)](#) for all resource accesses and role definitions. These permissions are inherited to child resources that exist in the hierarchy. Any built-in RBAC role can be assigned to a management group that will inherit down the hierarchy to the resources. For example, the RBAC role VM contributor can be assigned to a management group. This role has no action on the management group, but will inherit to all VMs under that management group.

The following chart shows the list of roles and the supported actions on management groups.

RBAC ROLE NAME	CREATE	RENAME	MOVE	DELETE	ASSIGN ACCESS	ASSIGN POLICY	READ
Owner	X	X	X	X	X	X	X
Contributor	X	X	X	X			X
MG Contributor*	X	X	X	X			X
Reader							X
MG Reader*							X
Resource Policy Contributor						X	
User Access Administrator					X		

*: MG Contributor and MG Reader only allow users to perform those actions on the management group scope.

Custom RBAC Role Definition and Assignment

Custom RBAC roles aren't supported on management groups at this time. See the [management group feedback forum](#) to view the status of this item.

Next steps

To learn more about management groups, see:

- [Create management groups to organize Azure resources](#)

- How to change, delete, or manage your management groups
- Install the Azure PowerShell module
- Review the REST API Spec
- Install the Azure CLI extension

Develop Azure Resource Manager templates for cloud consistency

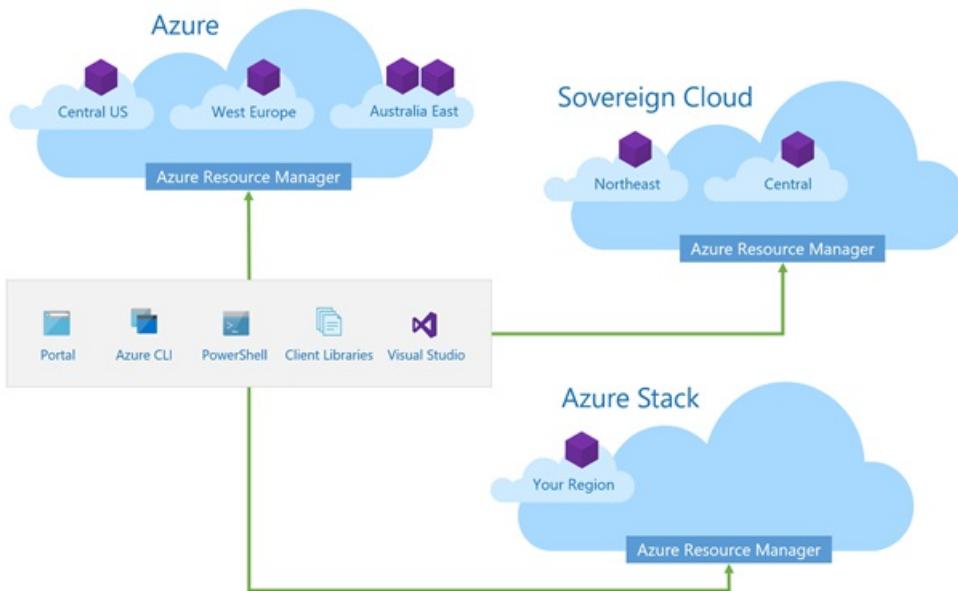
7/10/2018 • 25 minutes to read • [Edit Online](#)

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 Manager 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-AzureRmTemplateFunctions cmdlet:

```
# Import the module
Import-Module <path to local clone>\AzureRmTemplateFunctions.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')))]"  
        ]  
    }  
}
```

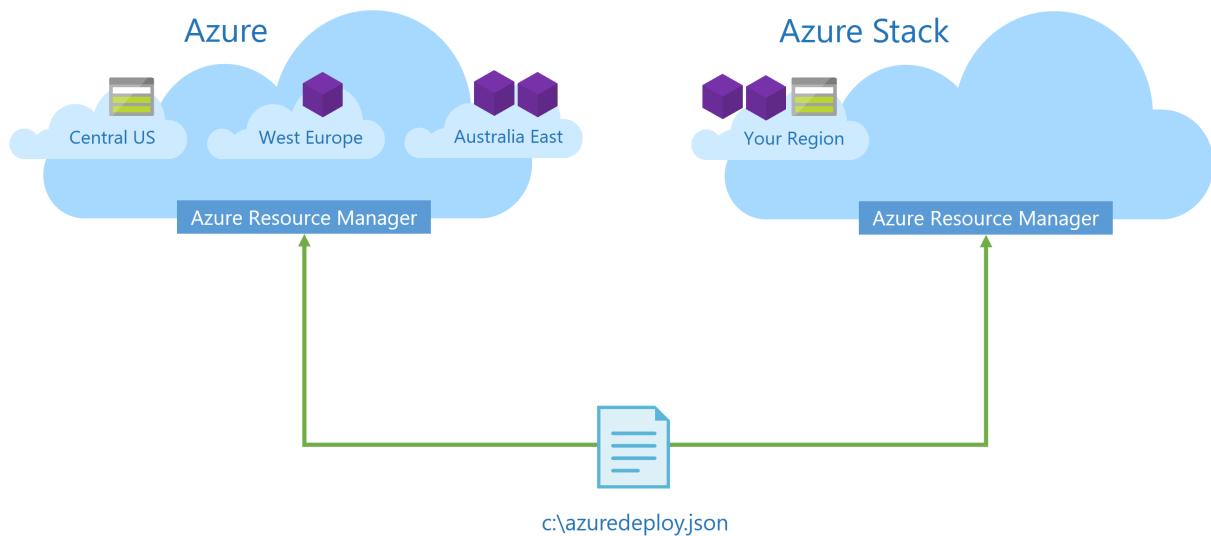
With this approach, all deployment artifacts, including configuration scripts, can be stored in the same location with the template itself. To change the location of all the links, you only need to specify a different base URL for the `_artifactsLocation` parameters.

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, ResourceType, 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.

```
{
    "$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 Redis Cache

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, SQL Azure, 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 URL 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**:

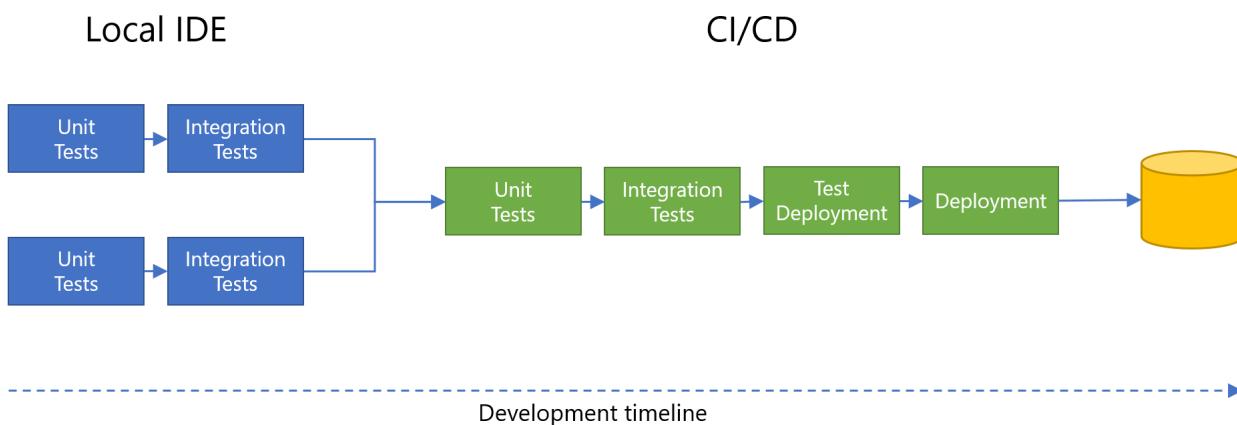
```
Get-AzureRmVMExtensionImage -Location myLocation -PublisherName Microsoft.PowerShell -Type DSC | FT
```

To get a list of publishers, use the [Get-AzureRmVmImagePublisher](#) command. To request type, use the [Get-AzureRmVMExtensionImageType](#) command.

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 address 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](#)

Understand the structure and syntax of Azure Resource Manager Templates

7/17/2018 • 6 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. For a step-by-step tutorial on creating a template, see [Create your first Azure Resource Manager template](#).

Template format

In its simplest structure, a template has the following elements:

```
{  
  "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",  
  "contentVersion": "",  
  "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. Use the URL shown in the preceding example.
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.
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.

ELEMENT NAME	REQUIRED	DESCRIPTION
outputs	No	Values that are returned after deployment.

Each element has properties you can set. The following example shows the full syntax for a template:

```
{
    "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "",
    "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>"
            }
        }
    },
    "variables": {
        "<variable-name>": "<variable-value>",
        "<variable-object-name>": {
            <variable-complex-type-value>
        },
        "<variable-object-name>": {
            "copy": [
                {
                    "name": "<name-of-array-property>",
                    "count": <number-of-iterations>,
                    "input": {
                        <properties-to-repeat>
                    }
                }
            ]
        },
        "copy": [
            {
                "name": "<variable-array-name>",
                "count": <number-of-iterations>,
                "input": {
                    <properties-to-repeat>
                }
            }
        ]
    },
    "functions": [
        {
            "namespace": "<namespace-for-your-function>",
            "members": {
                "<function-name>": {
                    "parameters": [
                        {
                            "name": "<parameter-name>",
                            "type": "<type-of-parameter-value>"
                        }
                    ],
                    "output": {
                        "type": "<type-of-output-value>",
                        "value": "<function-expression>"
                    }
                }
            }
        }
    ]
}
```

```

        }
    ],
    "resources": [
        {
            "condition": "<boolean-value-whether-to-deploy>",
            "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": {}
                    }
                ]
            },
            "resources": [
                "<array-of-child-resources>"
            ]
        }
    ],
    "outputs": {
        "<outputName>": {
            "type": "<type-of-output-value>",
            "value": "<output-value-expression>"
        }
    }
}

```

This article describes the sections of the template in greater detail.

Syntax

The basic syntax of the template is JSON. However, expressions and functions extend the JSON values available within the template. Expressions are written within JSON string literals whose first and last characters are the brackets: [and], respectively. The value of the expression is evaluated when the template is deployed. While written as a string literal, the result of evaluating the expression can be of a different JSON type, such as an array or integer, depending on the actual expression. To have a literal string start with a bracket [but not have it interpreted as an expression, add an extra bracket to start the string with [[.

Typically, you use expressions with functions to perform operations for configuring the deployment. Just like in JavaScript, function calls are formatted as `functionName(arg1,arg2,arg3)`. You reference properties by using the dot and [index] operators.

The following example shows how to use several functions when constructing a value:

```
"variables": {
    "storageName": "[concat(toLower(parameters('storageNamePrefix')), uniqueString(resourceGroup().id))]"
}
```

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.

The following example shows a simple parameter definition:

```
"parameters": {
    "siteNamePrefix": {
        "type": "string",
        "metadata": {
            "description": "The name prefix of the web app that you wish to create."
        }
    },
},
```

For information about defining parameters, see [Parameters section of Azure Resource Manager templates](#).

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.

The following example shows a simple variable definition:

```
"variables": {
    "webSiteName": "[concat(parameters('siteNamePrefix'), uniqueString(resourceGroup().id))]",
},
```

For information about defining variables, see [Variables section of Azure Resource Manager templates](#).

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't access template parameters. That is, the [parameters function](#) is restricted to function parameters.
- 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. This section can get complicated because you must understand the types you're deploying to provide the right values.

```
"resources": [
  {
    "apiVersion": "2016-08-01",
    "name": "[variables('webSiteName')]",
    "type": "Microsoft.Web/sites",
    "location": "[resourceGroup().location]",
    "properties": {
      "serverFarmId": "/subscriptions/<subscription-id>/resourcegroups/<resource-group-name>/providers/Microsoft.Web/serverFarms/<plan-name>"
    }
  }
],
```

For more information, see [Resources section of Azure Resource Manager templates](#).

Outputs

In the Outputs section, you specify values that are returned from deployment. For example, you could return the URI to access a deployed resource.

```
"outputs": {  
    "newHostName": {  
        "type": "string",  
        "value": "[reference(variables('webSiteName')).defaultHostName]"  
    }  
}
```

For more information, see [Outputs section of Azure Resource Manager templates](#).

Template limits

Limit the size of your template to 1 MB, and each parameter file to 64 KB. The 1-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](#).

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 multiple templates during deployment, see [Using linked templates with Azure Resource Manager](#).
- For recommendations on creating Resource Manager templates that you can use across global Azure, Azure sovereign clouds, and Azure Stack, see [Develop Azure Resource Manager templates for cloud consistency](#).

Parameters section of Azure Resource Manager templates

5/21/2018 • 6 minutes to read • [Edit Online](#)

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 do not 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 are limited to 255 parameters in a template. You can reduce the number of parameters by using objects that contain multiple properties, as shown in this article.

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')]"  
        },  
        ...  
    }  
]
```

Available properties

The preceding example showed only some of the properties you can use in the parameter section. The available properties 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, securestring, and array type parameters, this value is inclusive.
maxLength	No	The maximum length for string, securestring, and array type parameters, this value is inclusive.
description	No	Description of the parameter that is displayed to users through the portal.

Template functions with parameters

When providing 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 cannot use the `reference` function in the parameters section. Parameters are evaluated before deployment so the `reference` function cannot obtain 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]"
                    }
                }
            ]
        }
    }
]

```

Recommendations

The following information can be helpful when you work with parameters:

- Minimize your use of parameters. Whenever possible, use a variable or a literal value. Use parameters only for these scenarios:
 - Settings that you want to use variations of according to environment (SKU, size, capacity).
 - Resource names that you want to specify for easy identification.
 - Values that you use frequently to complete other tasks (such as an admin user name).
 - Secrets (such as passwords).
 - The number or array of values to use when you create multiple instances of a resource type.
- Use camel case for parameter names.
- 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 (except for passwords and SSH keys). By providing a default value, the parameter becomes optional during deployment. The default value can be an empty string.

```

"parameters": {
    "storageAccountType": {
        "type": "string",
        "defaultValue": "Standard_GRS",
        "metadata": {
            "description": "The type of the new storage account created to store the VM disks."
        }
    }
}

```

- Use **securestring** for all passwords and secrets. If you pass sensitive data in a JSON object, use the **secureObject** type. Template parameters with securestring or secureObject types cannot be read after resource deployment.

```

"parameters": {
    "secretValue": {
        "type": "securestring",
        "metadata": {
            "description": "The value of the secret to store in the vault."
        }
    }
}

```

- Use a parameter to specify location, and share that parameter value as much as possible with resources that are likely to be in the same location. This approach minimizes the number of times users are asked to provide location information. If a resource type is supported in only a limited number of locations, you might want to specify a valid location directly in the template, or add another location parameter. When an organization limits the allowed regions for its users, the **resourceGroup().location** expression might prevent a user from being able to deploy the template. For example, one user creates a resource group in a region. A second user must deploy to that resource group but does not have access to the region.

```

"resources": [
{
    "name": "[variables('storageAccountName')]",
    "type": "Microsoft.Storage/storageAccounts",
    "apiVersion": "2016-01-01",
    "location": "[parameters('location')]",
    ...
}
]

```

- Avoid using a parameter or variable for the API version for a resource type. Resource properties and values can vary by version number. IntelliSense in a code editor cannot determine the correct schema when the API version is set to a parameter or variable. Instead, hard-code the API version in the template.
- Avoid specifying a parameter name in your template that matches a parameter in the 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-AzureRmResourceGroupDeployment](#) cmdlet. During deployment, you are prompted to provide a value for **ResourceGroupNameFromTemplate**.

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 does not deploy any resources. It constructs parameter values and returns those values.
parameter object	Demonstrates using an object for a parameter. The template does not deploy any resources. It constructs parameter values and returns those values.

Next steps

- To view complete templates for many different types of solutions, see the [Azure Quickstart Templates](#).
- For how to input the parameter values during deployment, see [Deploy an application with Azure Resource Manager template](#).
- For details about the functions you can use from within a template, see [Azure Resource Manager Template Functions](#).
- For information about using a parameter object, see [Use an object as a parameter in an Azure Resource Manager template](#).

Variables section of Azure Resource Manager templates

5/21/2018 • 4 minutes to read • [Edit Online](#)

In the variables section, you construct values that can be used throughout your template. You do not need to define variables, but they often simplify your template by reducing complex expressions.

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 concatenate 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",  
        ...
```

Available definitions

The preceding example showed one way to define a variable. You can use any of the following definitions:

```

"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": {
                    "<properties-to-repeat>
                }
            }
        ]
    },
    "copy": [
        {
            "name": "<variable-array-name>",
            "count": <number-of-iterations>,
            "input": {
                "<properties-to-repeat>
            }
        }
    ]
}

```

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]"
```

Use copy element in variable definition

You can use the **copy** syntax to create a variable with an array of multiple elements. You provide a count for the number of elements. Each element contains the properties within the **input** object. You can use copy either within a variable or to create the variable. When you define a variable and use **copy** within that variable, you create an object that has an array property. When you use **copy** at the top level and define one or more variables within it, you create one or more arrays. Both approaches are shown in the following example:

```
"variables": {
  "disk-array-on-object": {
    "copy": [
      {
        "name": "disks",
        "count": 3,
        "input": {
          "name": "[concat('myDataDisk', copyIndex('disks', 1))]",
          "diskSizeGB": "1",
          "diskIndex": "[copyIndex('disks')]"
        }
      }
    ],
    "copy": [
      {
        "name": "disks-top-level-array",
        "count": 3,
        "input": {
          "name": "[concat('myDataDisk', copyIndex('disks-top-level-array', 1))]",
          "diskSizeGB": "1",
          "diskIndex": "[copyIndex('disks-top-level-array')]"
        }
      }
    ]
  },
}
```

The variable **disk-array-on-object** contains the following object with an array named **disks**:

```
{
  "disks": [
    {
      "name": "myDataDisk1",
      "diskSizeGB": "1",
      "diskIndex": 0
    },
    {
      "name": "myDataDisk2",
      "diskSizeGB": "1",
      "diskIndex": 1
    },
    {
      "name": "myDataDisk3",
      "diskSizeGB": "1",
      "diskIndex": 2
    }
  ]
}
```

The variable **disks-top-level-array** contains the following array:

```
[
  {
    "name": "myDataDisk1",
    "diskSizeGB": "1",
    "diskIndex": 0
  },
  {
    "name": "myDataDisk2",
    "diskSizeGB": "1",
    "diskIndex": 1
  },
  {
    "name": "myDataDisk3",
    "diskSizeGB": "1",
    "diskIndex": 2
  }
]
```

You can also specify more than one object when using copy to create variables. The following example defines two arrays as variables. One is named **disks-top-level-array** and has five elements. The other is named **a-different-array** and has three elements.

```
"variables": {
  "copy": [
    {
      "name": "disks-top-level-array",
      "count": 5,
      "input": {
        "name": "[concat('oneDataDisk', copyIndex('disks-top-level-array', 1))]",
        "diskSizeGB": "1",
        "diskIndex": "[copyIndex('disks-top-level-array')]"
      }
    },
    {
      "name": "a-different-array",
      "count": 3,
      "input": {
        "name": "[concat('twoDataDisk', copyIndex('a-different-array', 1))]",
        "diskSizeGB": "1",
        "diskIndex": "[copyIndex('a-different-array')]"
      }
    }
  ]
},
```

This approach works well when you need to take parameter values and make sure they are in the correct format for a template value. The following example formats parameter values for use in defining security rules:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "securityRules": {
            "type": "array"
        }
    },
    "variables": {
        "copy": [
            {
                "name": "securityRules",
                "count": "[length(parameters('securityRules'))]",
                "input": {
                    "name": "[parameters('securityRules')[copyIndex('securityRules')].name]",
                    "properties": {
                        "description": "[parameters('securityRules')[copyIndex('securityRules')].description]",
                        "priority": "[parameters('securityRules')[copyIndex('securityRules')].priority]",
                        "protocol": "[parameters('securityRules')[copyIndex('securityRules')].protocol]",
                        "sourcePortRange": "[parameters('securityRules')[copyIndex('securityRules')].sourcePortRange]",
                        "destinationPortRange": "[parameters('securityRules')[copyIndex('securityRules')].destinationPortRange]",
                        "sourceAddressPrefix": "[parameters('securityRules')[copyIndex('securityRules')].sourceAddressPrefix]",
                        "destinationAddressPrefix": "[parameters('securityRules')[copyIndex('securityRules')].destinationAddressPrefix]",
                        "access": "[parameters('securityRules')[copyIndex('securityRules')].access]",
                        "direction": "[parameters('securityRules')[copyIndex('securityRules')].direction]"
                    }
                }
            }
        ]
    },
    "resources": [
        {
            "apiVersion": "2015-06-15",
            "type": "Microsoft.Network/networkSecurityGroups",
            "name": "NSG1",
            "location": "[resourceGroup().location]",
            "properties": {
                "securityRules": "[variables('securityRules')]"
            }
        }
    ],
    "outputs": {}
}
```

Recommendations

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.
- You cannot 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.

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 does not 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 does not 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.

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 multiple templates during deployment, see [Using linked templates with Azure Resource Manager](#).
- You may need to use resources that exist within a different resource group. This scenario is common when working with storage accounts or virtual networks that are shared across multiple resource groups. For more information, see the [resourceId function](#).

Understand the structure and syntax of Azure Resource Manager Templates

7/17/2018 • 6 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. For a step-by-step tutorial on creating a template, see [Create your first Azure Resource Manager template](#).

Template format

In its simplest structure, a template has the following elements:

```
{  
  "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",  
  "contentVersion": "",  
  "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. Use the URL shown in the preceding example.
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.
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.

ELEMENT NAME	REQUIRED	DESCRIPTION
outputs	No	Values that are returned after deployment.

Each element has properties you can set. The following example shows the full syntax for a template:

```
{
    "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "",
    "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>"
            }
        }
    },
    "variables": {
        "<variable-name>": "<variable-value>",
        "<variable-object-name>": {
            <variable-complex-type-value>
        },
        "<variable-object-name>": {
            "copy": [
                {
                    "name": "<name-of-array-property>",
                    "count": <number-of-iterations>,
                    "input": {
                        <properties-to-repeat>
                    }
                }
            ]
        },
        "copy": [
            {
                "name": "<variable-array-name>",
                "count": <number-of-iterations>,
                "input": {
                    <properties-to-repeat>
                }
            }
        ]
    },
    "functions": [
        {
            "namespace": "<namespace-for-your-function>",
            "members": {
                "<function-name>": {
                    "parameters": [
                        {
                            "name": "<parameter-name>",
                            "type": "<type-of-parameter-value>"
                        }
                    ],
                    "output": {
                        "type": "<type-of-output-value>",
                        "value": "<function-expression>"
                    }
                }
            }
        }
    ]
}
```

```

        }
    ],
    "resources": [
        {
            "condition": "<boolean-value-whether-to-deploy>",
            "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": {}
                    }
                ]
            },
            "resources": [
                "<array-of-child-resources>"
            ]
        }
    ],
    "outputs": {
        "<outputName>": {
            "type": "<type-of-output-value>",
            "value": "<output-value-expression>"
        }
    }
}

```

This article describes the sections of the template in greater detail.

Syntax

The basic syntax of the template is JSON. However, expressions and functions extend the JSON values available within the template. Expressions are written within JSON string literals whose first and last characters are the brackets: `[` and `]`, respectively. The value of the expression is evaluated when the template is deployed. While written as a string literal, the result of evaluating the expression can be of a different JSON type, such as an array or integer, depending on the actual expression. To have a literal string start with a bracket `[`, but not have it interpreted as an expression, add an extra bracket to start the string with `[[`.

Typically, you use expressions with functions to perform operations for configuring the deployment. Just like in JavaScript, function calls are formatted as `functionName(arg1,arg2,arg3)`. You reference properties by using the dot and `[index]` operators.

The following example shows how to use several functions when constructing a value:

```
"variables": {
    "storageName": "[concat(toLower(parameters('storageNamePrefix')), uniqueString(resourceGroup().id))]"
}
```

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.

The following example shows a simple parameter definition:

```
"parameters": {
    "siteNamePrefix": {
        "type": "string",
        "metadata": {
            "description": "The name prefix of the web app that you wish to create."
        }
    },
},
```

For information about defining parameters, see [Parameters section of Azure Resource Manager templates](#).

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.

The following example shows a simple variable definition:

```
"variables": {
    "webSiteName": "[concat(parameters('siteNamePrefix'), uniqueString(resourceGroup().id))]",
},
```

For information about defining variables, see [Variables section of Azure Resource Manager templates](#).

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't access template parameters. That is, the [parameters function](#) is restricted to function parameters.
- 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. This section can get complicated because you must understand the types you're deploying to provide the right values.

```
"resources": [
  {
    "apiVersion": "2016-08-01",
    "name": "[variables('webSiteName')]",
    "type": "Microsoft.Web/sites",
    "location": "[resourceGroup().location]",
    "properties": {
      "serverFarmId": "/subscriptions/<subscription-id>/resourcegroups/<resource-group-name>/providers/Microsoft.Web/serverFarms/<plan-name>"
    }
  }
],
```

For more information, see [Resources section of Azure Resource Manager templates](#).

Outputs

In the Outputs section, you specify values that are returned from deployment. For example, you could return the URI to access a deployed resource.

```
"outputs": {  
    "newHostName": {  
        "type": "string",  
        "value": "[reference(variables('webSiteName')).defaultHostName]"  
    }  
}
```

For more information, see [Outputs section of Azure Resource Manager templates](#).

Template limits

Limit the size of your template to 1 MB, and each parameter file to 64 KB. The 1-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](#).

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 multiple templates during deployment, see [Using linked templates with Azure Resource Manager](#).
- For recommendations on creating Resource Manager templates that you can use across global Azure, Azure sovereign clouds, and Azure Stack, see [Develop Azure Resource Manager templates for cloud consistency](#).

Resources section of Azure Resource Manager templates

7/12/2018 • 14 minutes to read • [Edit Online](#)

In the resources section, you define the resources that are deployed or updated. This section can get complicated because you must understand the types you're deploying to provide the right values.

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 <code>true</code> , the resource is created during the deployment. When <code>false</code> , the resource is skipped for this deployment.
apiVersion	Yes	Version of the REST API to use for creating the resource.

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).
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.
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
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 multiple 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.
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. The fully qualified type of the child resource includes the parent resource type, such as Microsoft.Web/sites/extensions . Dependency on the parent resource isn't implied. You must explicitly define that dependency.

Condition

When you must decide during deployment whether or not 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 will be created. When the value is false, the resource will not be created. 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](#).

Resource-specific values

The **apiVersion**, **type**, and **properties** elements are different for each resource type. The **sku**, **kind**, and **plan** elements are available for some resource types, but not all. To determine values for these properties, see [template reference](#).

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.

Unique resource names

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 Storage¹
- Web Apps feature of Azure App Service
- SQL Server
- Azure Key Vault
- Azure Redis Cache
- 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')]"
}
```

Resource names for identification

Some resource types you might want to name, but their names do not have to be unique. For these resource types, you can provide a name that identifies both the resource context and the resource type.

```
"parameters": {
    "vmName": {
        "type": "string",
        "defaultValue": "demoLinuxVM",
        "metadata": {
            "description": "The name of the VM to create."
        }
    }
}
```

Generic resource names

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",
    ...
}
```

Location

When deploying a template, you must provide a location for each resource. Different resource types are supported in different locations. To see a list of locations that are available to your subscription for a particular resource type, use Azure PowerShell or Azure CLI.

The following example uses PowerShell to get the locations for the `Microsoft.Web\sites` resource type:

```
((Get-AzureRmResourceProvider -ProviderNamespace Microsoft.Web).ResourceTypes | Where-Object ResourceType -eq sites).Locations
```

The following example uses Azure CLI to get the locations for the `Microsoft.Web\sites` resource type:

```
az provider show -n Microsoft.Web --query "resourceTypes[?resourceType=='sites'].locations"
```

After determining the supported locations for your resources, set that location in your template. The easiest way to set this value is to create a resource group in a location that supports the resource types, and set each location to `[resourceGroup().location]`. You can redeploy the template to resource groups in different locations, and not change any values in the template or parameters.

The following example shows a storage account that is deployed to the same location as the resource group:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "variables": {
    "storageName": "[concat('storage', uniqueString(resourceGroup().id))]"
  },
  "resources": [
    {
      "apiVersion": "2016-01-01",
      "type": "Microsoft.Storage/storageAccounts",
      "name": "[variables('storageName')]",
      "location": "[resourceGroup().location]",
      "tags": {
        "Dept": "Finance",
        "Environment": "Production"
      },
      "sku": {
        "name": "Standard_LRS"
      },
      "kind": "Storage",
      "properties": { }
    }
  ]
}
```

If you need to hardcode the location in your template, provide the name of one of the supported regions. The following example shows a storage account that is always deployed to North Central US:

```
{
  "$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('storageloc', uniqueString(resourceGroup().id))]",
      "location": "North Central US",
      "tags": {
        "Dept": "Finance",
        "Environment": "Production"
      },
      "sku": {
        "name": "Standard_LRS"
      },
      "kind": "Storage",
      "properties": { }
    }
  ]
}
```

Tags

You apply tags to your Azure resources to logically organize them by categories. Each tag consists of a name and a value. 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.

The following limitations apply to tags:

- Each resource or resource group can have a maximum of 15 tag name/value pairs. This limitation applies only to tags directly applied to the resource group or resource. A resource group can contain many resources that each have 15 tag name/value pairs. If you have more than 15 values that you need to associate with a resource, use a JSON string for the tag value. The JSON string can contain many values that are applied to a single tag name. This article shows an example of assigning a JSON string to the tag.
- 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.
- 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: <, >, %, &, \, ?, /

Add tags to your template

To tag a resource during deployment, add the `tags` element to the resource you are 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",
  "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": {}
    }
  ]
}
```

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": {
    "tagValues": {
      "type": "object",
      "defaultValue": {
        "Dept": "Finance",
        "Environment": "Production"
      }
    }
  },
  "resources": [
    {
      "apiVersion": "2016-01-01",
      "type": "Microsoft.Storage/storageAccounts",
      "name": "[concat('storage', uniqueString(resourceGroup().id))]",
      "location": "[resourceGroup().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 cannot 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",
  "resources": [
    {
      "apiVersion": "2016-01-01",
      "type": "Microsoft.Storage/storageAccounts",
      "name": "[concat('storage', uniqueString(resourceGroup().id))]",
      "location": "[resourceGroup().location]",
      "tags": {
        "CostCenter": "{\"Dept\":\"Finance\", \"Environment\":\"Production\"}"
      },
      "sku": {
        "name": "Standard_LRS"
      },
      "kind": "Storage",
      "properties": { }
    }
  ]
}
```

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.

```
{
  "name": "exampleserver",
  "type": "Microsoft.Sql/servers",
  "apiVersion": "2014-04-01",
  ...
  "resources": [
    {
      "name": "exampledatabase",
      "type": "databases",
      "apiVersion": "2014-04-01",
      ...
    }
  ]
}
```

When nested, the type is set to `databases` but its full resource type is `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.

The format of the child resource type is:

```
{resource-provider-namespace}/{parent-resource-type}/{child-resource-type}
```

The format of the child resource name is: `{parent-resource-name}/{child-resource-name}`

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 multiple child resources. With this approach, you must provide the full resource type, and include the parent resource name in the child resource name.

```
{
  "name": "exampleserver",
  "type": "Microsoft.Sql/servers",
  "apiVersion": "2014-04-01",
  "resources": [
  ],
  ...
},
{
  "name": "exampleserver/exampledatabase",
  "type": "Microsoft.Sql/servers/databases",
  "apiVersion": "2014-04-01",
  ...
}
```

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

Recommendations

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), *do not 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'), '.vhf')]"
  }
}
```

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'), '.vhf')]"
  }
}
```

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.

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 use more than one template during deployment, see [Using linked templates with Azure Resource Manager](#).
- You may need to use resources that exist within a different resource group. This scenario is common when working with storage accounts or virtual networks that are shared across several resource groups. For more information, see the [resourceId function](#).
- For information about resource name restrictions, see [Recommended naming conventions for Azure resources](#).

Outputs section in Azure Resource Manager templates

5/21/2018 • 2 minutes to read • [Edit Online](#)

In the Outputs section, you specify values that are returned from deployment. For example, you could return the URI to access a deployed resource.

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'))]"  
    }  
}
```

After the deployment, you can retrieve the value with script. For PowerShell, use:

```
(Get-AzureRmResourceGroupDeployment -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('<name-of-deployment>').outputs.<property-name>.value]" .
```

For example, you can set the IP address on a load balancer by retrieving a value from a linked template.

```
"publicIPAddress": {  
    "id": "[reference('linkedTemplate').outputs.resourceID.value]"  
}
```

You cannot 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.

Available properties

The following example shows the structure of an output definition:

```

"outputs": {
    "<outputName>" : {
        "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.
type	Yes	Type of the output value. Output values support the same types as template input parameters.
value	Yes	Template language expression that is evaluated and returned as output value.

Recommendations

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"
    }
}

```

Example templates

TEMPLATE	DESCRIPTION
Copy variables	Creates complex variables and outputs those values. Does not 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.

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 multiple templates during deployment, see [Using linked templates with Azure Resource Manager](#).

- You may need to use resources that exist within a different resource group. This scenario is common when working with storage accounts or virtual networks that are shared across multiple resource groups. For more information, see the [resourceId function](#).

Using linked and nested templates when deploying Azure resources

5/30/2018 • 9 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 template, 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.

Link or nest a template

To link to another template, add a **deployments** resource to your main template.

```
"resources": [
  {
    "apiVersion": "2017-05-10",
    "name": "linkedTemplate",
    "type": "Microsoft.Resources/deployments",
    "properties": {
      "mode": "Incremental",
      <nested-template-or-external-template>
    }
  }
]
```

The properties you provide for the deployment resource vary based on whether you are 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": [
    {
        "apiVersion": "2017-05-10",
        "name": "nestedTemplate",
        "type": "Microsoft.Resources/deployments",
        "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",
                        "name": "[variables('storageName')]",
                        "apiVersion": "2015-06-15",
                        "location": "West US",
                        "properties": {
                            "accountType": "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.

You cannot 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.

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 cannot 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": [
  {
    "apiVersion": "2017-05-10",
    "name": "linkedTemplate",
    "type": "Microsoft.Resources/deployments",
    "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. To pass a value from the main template to the linked template, use **parameters**.

```

"resources": [
  {
    "apiVersion": "2017-05-10",
    "name": "linkedTemplate",
    "type": "Microsoft.Resources/deployments",
    "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 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('<name-of-deployment>').outputs.<property-name>.value]" .
```

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": [
        {
            "apiVersion": "2017-05-10",
            "name": "linkedTemplate",
            "type": "Microsoft.Resources/deployments",
            "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",
            "name": "[parameters('publicIPAddresses_name')]",
            "apiVersion": "2017-06-01",
            "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",
            "name": "[parameters('loadBalancers_name')]",
            "apiVersion": "2017-06-01",
            "location": "eastus",
            "properties": {
                "frontendIPConfigurations": [
                    {
                        "name": "LoadBalancerFrontEnd",
                        "properties": {
                            "privateIPAllocationMethod": "Dynamic",
                            "publicIPAddress": {
                                "id": "[reference('linkedTemplate').outputs.resourceID.value]"
                            }
                        }
                    }
                ],
                "backendAddressPools": [],
                "loadBalancingRules": [],
                "probes": [],
                "inboundNatRules": [],
                "outboundNatRules": [],
                "inboundNatPools": []
            },
            "dependsOn": [
                "linkedTemplate"
            ]
        },
        {
            "apiVersion": "2017-05-10",
            "name": "linkedTemplate",
            "type": "Microsoft.Resources/deployments",
            "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",
            "name": "[parameters('publicIPAddresses_name')]",
            "apiVersion": "2017-06-01",
            "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": [
        {
            "apiVersion": "2017-05-10",
            "name": "[concat('linkedTemplate', copyIndex())]",
            "type": "Microsoft.Resources/deployments",
            "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())]"}
                }
            },
            "copy": {
                "count": 3,
                "name": "ip-loop"
            }
        }
    ]
}
```

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-AzureRmResourceGroupDeployment -ResourceGroupName examplegroup -Name $name
    Write-Output "deployment $($deployment.DeploymentName) returned
$($deployment.Outputs.returnedIPAddress.value)"
}
```

Or, Azure CLI script:

```
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": [  
        {  
            "apiVersion": "2017-05-10",  
            "name": "linkedTemplate",  
            "type": "Microsoft.Resources/deployments",  
            "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-AzureRmResourceGroupDeployment** command.

```
Set-AzureRmCurrentStorageAccount -ResourceGroupName ManageGroup -Name storagecontosotemplates  
$token = New-AzureStorageContainerSASToken -Name templates -Permission r -ExpiryTime (Get-  
Date).AddMinutes(30.0)  
$url = (Get-AzureStorageBlob -Container templates -Blob parent.json).ICloudBlob.uri.AbsoluteUri  
New-AzureRmResourceGroupDeployment -ResourceGroupName ExampleGroup -TemplateUri ($url + $token) -  
containerSasToken $token
```

In Azure CLI, you get a token for the container and deploy the templates with the following code:

```
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 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](#).

Define the order for deploying resources in Azure Resource Manager Templates

7/6/2018 • 5 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.

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 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](#).

Recommendations for setting dependencies

When deciding what dependencies to set, use the following guidelines:

- Set as few dependencies as possible.
- Set a child resource as dependent on its parent resource.
- Use the **reference** function and pass in the resource name to set implicit dependencies between resources that need to share a property. Don't add an explicit dependency (**dependsOn**) when you've already defined an implicit dependency. This approach reduces the risk of having unnecessary dependencies.
- Set a dependency when a resource can't be **created** without functionality from another resource. Don't set a dependency if the resources only interact after deployment.
- 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.

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 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](#).

Deploy multiple instances of a resource or property in Azure Resource Manager Templates

7/10/2018 • 7 minutes to read • [Edit Online](#)

This article shows you how to iterate in your Azure Resource Manager template to create multiple instances of a resource. If you need to specify whether a resource is deployed at all, see [condition element](#).

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, you specify the number of iterations and a name for this loop. The count value must be a positive integer and can't exceed 800.

The resource to create multiple 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 to perform 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. Therefore, 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 multiple instances 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.

Property iteration

To create multiple values 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 multiple 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` element in the variables section. You can create multiple instances of objects with related values, and then assign those values to instances of the resource. You can use `copy` to create either an object with an array property or an array. Both approaches are shown in the following example:

```
{
    "$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')]"
                    }
                }
            ]
        },
        "copy": [
            {
                "name": "disks-top-level-array",
                "count": 5,
                "input": {
                    "name": "[concat('myDataDisk', copyIndex('disks-top-level-array', 1))]",
                    "diskSizeGB": "1",
                    "diskIndex": "[copyIndex('disks-top-level-array')]"
                }
            }
        ]
    },
    "resources": [],
    "outputs": {
        "exampleObject": {
            "value": "[variables('disk-array-on-object')]",
            "type": "object"
        },
        "exampleArrayOnObject": {
            "value": "[variables('disk-array-on-object').disks]",
            "type": "array"
        },
        "exampleArray": {
            "value": "[variables('disks-top-level-array')]",
            "type": "array"
        }
    }
}
```

With either approach, the copy element is an array so you can specify more than one variable. Add an object for each variable to create.

```

"copy": [
  {
    "name": "first-variable",
    "count": 5,
    "input": {
      "demoProperty": "[concat('myProperty', copyIndex('first-variable'))]"
    }
  },
  {
    "name": "second-variable",
    "count": 3,
    "input": {
      "demoProperty": "[concat('myProperty', copyIndex('second-variable'))]"
    }
  }
]

```

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 multiple instances 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 multiple instances of data sets, 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 multiple resources or properties.

TEMPLATE	DESCRIPTION
Copy storage	Deploys multiple storage accounts with an index number in the name.

TEMPLATE	DESCRIPTION
Serial copy storage	Deploys multiple storage accounts one at time. The name includes the index number.
Copy storage with array	Deploys multiple storage accounts. The name includes a value from an array.
VM deployment with a variable number of data disks	Deploys multiple data disks with a virtual machine.
Copy variables	Demonstrates the different ways of iterating on variables.
Multiple security rules	Deploys multiple 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

- 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](#).

Deploy resources with Resource Manager templates and Azure PowerShell

5/21/2018 • 8 minutes to read • [Edit Online](#)

This article explains how to use Azure PowerShell with Resource Manager templates to deploy your resources to Azure. If you are not familiar with the concepts of deploying and managing your Azure solutions, see [Azure Resource Manager overview](#).

The Resource Manager template you deploy can either be a local file on your machine, or an external file that is located in a repository like GitHub. The template you deploy in this article is available in the [Sample template](#) section, or as [storage account template in GitHub](#).

If needed, install the Azure PowerShell module using the instructions found in the [Azure PowerShell guide](#), and then run `Connect-AzureRmAccount` to create a connection with Azure.

Deploy a template from your local machine

When deploying resources to Azure, you:

1. Log 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 cannot 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:

```
Connect-AzureRmAccount  
  
Select-AzureRmSubscription -SubscriptionName <yourSubscriptionName>  
  
New-AzureRmResourceGroup -Name ExampleResourceGroup -Location "South Central US"  
New-AzureRmResourceGroupDeployment -Name ExampleDeployment -ResourceGroupName ExampleResourceGroup `  
-TemplateFile c:\MyTemplates\storage.json -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 a template from an external source

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.

```
New-AzureRmResourceGroupDeployment -Name ExampleDeployment -ResourceGroupName ExampleResourceGroup ` -TemplateUri https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-storage-account-create/azuredeploy.json ` -storageAccountType Standard_GRS
```

The preceding example requires a publicly accessible URI for the template, which works for most scenarios because your template should not include sensitive data. If you need to specify sensitive data (like an admin password), pass that value as a secure parameter. However, if you do not 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. However, you must first load your template into the storage account for your Cloud Shell. If you have not 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>`.

A screenshot of the Microsoft Azure Resource groups blade. The title bar says "Microsoft Azure" and "Resource groups". Below the title, there's a "Subscriptions" section showing "6 of 7 selected" and a "Filter by name..." input field. The main area shows "11 items" with a "NAME" filter dropdown. Two resource groups are listed: "dashboards" and "cloud-shell-storage-westus", with "cloud-shell-storage-westus" highlighted with a red rectangle.

3. Select the storage account for your Cloud Shell.

A screenshot of the Microsoft Azure Resource group blade for "cloud-shell-storage-westus". The title bar says "cloud-shell-storage-westus" and "Resource group". The left sidebar has tabs for "Overview", "Activity log", "Access control (IAM)", "Tags", "Quickstart", and "Resource costs", with "Overview" selected. The main content area shows "Essentials" details like "Subscription name (change)" and "Third Internal Consumption". Below that is a "Filter by name..." input field and a list of "1 items" with a "NAME" filter dropdown. One item, "cs40773a725d727x4eb8x8cb", is highlighted with a red rectangle.

4. Select **Blobs**.

The screenshot shows the Azure Storage account overview page for 'cs40773a725d727x4eb8x8cb'. On the left, there's a navigation menu with options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, and Settings. Under Settings, there are links for Access keys, Configuration, Encryption, and Shared access signature. On the right, there's a summary of the storage account, including its resource group ('cloud-shell-storage-westus'), status ('Primary: Available'), location ('West US'), subscription ('Third Internal Consumption'), and subscription ID. Below this is a 'Services' section with a 'Blobs' card. The 'Blobs' card is highlighted with a red box and contains the text: 'REST-based object storage for unstructured data', along with links to 'View metrics', 'Configure CORS rules', and 'Setup custom domain'.

5. Select + Container.

The screenshot shows the 'Blob service' container creation dialog. At the top, it says 'Blob service' and 'cs40773a725d727x4eb8x8cb'. Below that are buttons for '+ Container', 'Refresh', and 'Delete'. A red box highlights the '+ Container' button. The next section shows the 'Storage account' as 'cs40773a725d727x4eb8x8cb' and 'Status' as '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**.

The screenshot shows the 'New container' dialog. It has a header 'Blob service' and 'cs40773a725d727x4eb8x8cb'. Below the header are buttons for '+ Container', 'Refresh', and 'Delete'. The main area is titled 'New container'. It has a field labeled '* Name' containing 'templates' with a green checkmark. Below that is a 'Public access level' dropdown set to 'Container (anonymous read access for containers and blobs)'. At the bottom are 'OK' and 'Cancel' buttons.

7. Select the container you created.

The screenshot shows the Azure Blob service interface. At the top, it displays the storage account name: cs40773a725d727x4eb8x8cb. Below this, there are tabs for 'Container' (selected), 'Refresh', and 'Delete'. The 'Status' section indicates 'Primary: Available' and 'Location: West US'. Under 'Subscription', it shows 'Third Internal Consumption' and 'Subscription ID'. A search bar at the bottom allows searching by container prefix. The main table lists a single item: 'NAME' with the value 'templates'.

8. Select **Upload**.

The screenshot shows the 'templates' container details page again. The 'Upload' button is highlighted with a red box. The 'Location' is set to 'templates'. A search bar at the bottom allows searching by blob prefix.

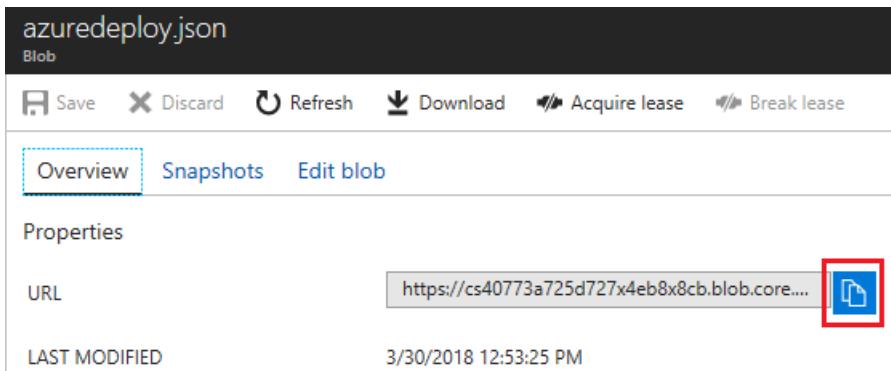
9. Find and upload your template.

The screenshot shows the 'Upload blob' dialog for the 'templates' container. The file 'azuredeploy.json' is selected in the 'Files' input field. An 'Advanced' section is expanded, and the 'Upload' button is visible at the bottom.

10. After it has uploaded, select the template.

The screenshot shows the 'templates' container details page again. The 'NAME' column lists the file 'azuredeploy.json', which is highlighted with a red box. The 'Search blobs by prefix (case-sensitive)' bar is also visible.

11. Copy the URL.



The screenshot shows the Azure Storage Blob Overview page for a file named 'azuredeploy.json'. The URL is highlighted with a red box. The URL is https://cs40773a725d727x4eb8x8cb.blob.core....

12. Open the prompt.



In the Cloud Shell, use the following commands:

```
New-AzureRmResourceGroup -Name ExampleResourceGroup -Location "South Central US"
New-AzureRmResourceGroupDeployment -ResourceGroupName ExampleResourceGroup `
    -TemplateUri <copied URL> `
    -storageAccountType Standard_GRS
```

Deploy to more than one resource group or subscription

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. You can deploy to only five resource groups in a single deployment. For more information, see [Deploy Azure resources to more than one subscription or resource group](#).

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 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 multiple parameter files for different deployment scenarios, and then pass in the appropriate parameter file.

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-AzureRmResourceGroupDeployment -Name ExampleDeployment -ResourceGroupName ExampleResourceGroup `  
-TemplateFile c:\MyTemplates\storage.json `  
-TemplateParameterFile c:\MyTemplates\storage.parameters.json
```

To pass an external parameter file, use the **TemplateParameterUri** parameter:

```
New-AzureRmResourceGroupDeployment -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
```

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 cannot 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 cannot include in the parameter file, either add that value to a key vault, or dynamically provide all parameter values inline.

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-AzureRmResourceGroupDeployment](#) cmdlet. You are 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 a template deployment

To test your template and parameter values without actually deploying any resources, use [Test-AzureRmResourceGroupDeployment](#).

```
Test-AzureRmResourceGroupDeployment -ResourceGroupName ExampleResourceGroup `  
-TemplateFile c:\MyTemplates\storage.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, attempting to pass an incorrect value for the storage account SKU, returns the following error:

```
Test-AzureRmResourceGroupDeployment -ResourceGroupName testgroup `  
-TemplateFile c:\MyTemplates\storage.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 could not parse the template. The message indicates the line number and position of the parsing error.

```
Test-AzureRmResourceGroupDeployment : After parsing a value an unexpected character was encountered:  
". Path 'variables', line 31, position 3.
```

Incremental and complete deployments

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 are not in the template:

- In complete mode, Resource Manager **deletes** resources that exist in the resource group but are not specified in the template.
- In incremental mode, Resource Manager **leaves unchanged** resources that exist in the resource group but are not specified in the template.

For both modes, Resource Manager attempts to provision all resources specified in the template. If the resource already exists in the resource group and its settings are unchanged, the operation results in no change. If you change the settings for a resource, the resource is provisioned with those new settings. If you attempt 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.

By default, Resource Manager uses the incremental mode.

To illustrate the difference between incremental and complete modes, consider the following scenario.

Existing Resource Group contains:

- Resource A
- Resource B
- Resource C

Template defines:

- Resource A
- Resource B
- Resource D

When deployed in **incremental** mode, the resource group contains:

- Resource A
- Resource B
- Resource C
- Resource D

When deployed in **complete** mode, Resource C is deleted. The resource group contains:

- Resource A
- Resource B
- Resource D

To use complete mode, use the `Mode` parameter:

```
New-AzureRmResourceGroupDeployment -Mode Complete -Name ExampleDeployment `  
-ResourceGroupName ExampleResourceGroup -TemplateFile c:\MyTemplates\storage.json
```

Sample template

The following template is used for the examples in this article. Copy and save it as a file named storage.json. To understand how to create this template, see [Create your first Azure Resource Manager 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"
      }
    }
  },
  "variables": {
    "storageAccountName": "[concat(uniquestring(resourceGroup().id), 'standardsa')]"
  },
  "resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "name": "[variables('storageAccountName')]",
      "apiVersion": "2016-01-01",
      "location": "[resourceGroup().location]",
      "sku": {
        "name": "[parameters('storageAccountType')]"
      },
      "kind": "Storage",
      "properties": {}
    }
  ],
  "outputs": {
    "storageAccountName": {
      "type": "string",
      "value": "[variables('storageAccountName')]"
    }
  }
}
```

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](#).
- For a complete sample script that deploys a template, see [Resource Manager template deployment script](#).
- 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](#).
- For guidance on how enterprises can use Resource Manager to effectively manage subscriptions, see [Azure enterprise scaffold - prescriptive subscription governance](#).

Deploy private Resource Manager template with SAS token and Azure PowerShell

5/21/2018 • 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:

```
# create a storage account for templates
New-AzureRmResourceGroup -Name ManageGroup -Location "South Central US"
New-AzureRmStorageAccount -ResourceGroupName ManageGroup -Name {your-unique-name} -Type Standard_LRS -Location "West US"
Set-AzureRmCurrentStorageAccount -ResourceGroupName ManageGroup -Name {your-unique-name}

# create a container and upload template
New-AzureStorageContainer -Name templates -Permission Off
Set-AzureStorageBlobContent -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-AzureRmCurrentStorageAccount -ResourceGroupName ManageGroup -Name {your-unique-name}

# get the URI with the SAS token
$templateuri = New-AzureStorageBlobSASToken -Container templates -Blob storage.json -Permission r `
-ExpiryTime (Get-Date).AddHours(2.0) -FullUri

# provide URI with SAS token during deployment
New-AzureRmResourceGroup -Name ExampleGroup -Location "South Central US"
New-AzureRmResourceGroupDeployment -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](#).

Export Azure Resource Manager templates with PowerShell

5/21/2018 • 4 minutes to read • [Edit Online](#)

Resource Manager enables you to export a Resource Manager template from existing resources in your subscription. You can use that generated template to learn about the template syntax or to automate the redeployment of your solution as needed.

It's important to note that there are two different ways to export a template:

- You can export the **actual template used for a deployment**. The exported template includes all the parameters and variables exactly as they appeared in the original template. This approach is helpful when you need to retrieve a template.
- You can export a **generated template that represents the current state of the resource group**. The exported template is not based on any template that you used for deployment. Instead, it creates a template that is a "snapshot" or "backup" of the resource group. The exported template has many hard-coded values and probably not as many parameters as you would typically define. Use this option to redeploy resources to the same resource group. To use this template for another resource group, you may have to significantly modify it.

This article shows both approaches.

Deploy a solution

To illustrate both approaches for exporting a template, let's start by deploying a solution to your subscription. If you already have a resource group in your subscription that you want to export, you don't have to deploy this solution. However, the rest of this article refers to the template for this solution. The example script deploys a storage account.

```
New-AzureRmResourceGroup -Name ExampleGroup -Location "South Central US"
New-AzureRmResourceGroupDeployment -ResourceGroupName ExampleGroup `
    -DeploymentName NewStorage
    -TemplateUri https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-storage-account-
create/azuredeploy.json
```

Save template from deployment history

You can retrieve a template from your deployment history by using the [Save-AzureRmResourceGroupDeploymentTemplate](#) command. The following example saves the template that you previously deploy:

```
Save-AzureRmResourceGroupDeploymentTemplate -ResourceGroupName ExampleGroup -DeploymentName NewStorage
```

It returns the location of the template.

```
Path
-----
C:\Users\exampleuser\NewStorage.json
```

Open the file, and notice that it's the exact template you used for deployment. The parameters and variables match the template from GitHub. You can redeploy this template.

Export resource group as template

Instead of retrieving a template from the deployment history, you can retrieve a template that represents the current state of a resource group by using the [Export-AzureRmResourceGroup](#) command. You use this command when you have made many changes to your resource group and no existing template represents all the changes. It is intended as a snapshot of the resource group, which you can use to redeploy to the same resource group. To use the exported template for other solutions, you must significantly modify it.

```
Export-AzureRmResourceGroup -ResourceGroupName ExampleGroup
```

It returns the location of the template.

```
Path
-----
C:\Users\exampleuser\ExampleGroup.json
```

Open the file, and notice that it's different than the template in GitHub. It has different parameters and no variables. The storage SKU and location are hard-coded to values. The following example shows the exported template, but your template has a slightly different parameter name:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "storageAccounts_nf3mvst4nqb36standardsa_name": {
      "defaultValue": null,
      "type": "String"
    }
  },
  "variables": {},
  "resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "sku": {
        "name": "Standard_LRS",
        "tier": "Standard"
      },
      "kind": "Storage",
      "name": "[parameters('storageAccounts_nf3mvst4nqb36standardsa_name')]",
      "apiVersion": "2016-01-01",
      "location": "southcentralus",
      "tags": {},
      "properties": {},
      "dependsOn": []
    }
  ]
}
```

You can redeploy this template, but it requires guessing a unique name for the storage account. The name of your parameter is slightly different.

```
New-AzureRmResourceGroupDeployment -ResourceGroupName ExampleGroup ` 
-TemplateFile C:\Users\exampleuser\ExampleGroup.json ` 
-storageAccounts_nf3mvst4nqb36standardsa_name tfnewstorage0501
```

Customize exported template

You can modify this template to make it easier to use and more flexible. To allow for more locations, change the location property to use the same location as the resource group:

```
"location": "[resourceGroup().location]",
```

To avoid having to guess a unique name for storage account, remove the parameter for the storage account name. Add a parameter for a storage name suffix, and a storage SKU:

```
"parameters": {
    "storageSuffix": {
        "type": "string",
        "defaultValue": "standardsa"
    },
    "storageAccountType": {
        "defaultValue": "Standard_LRS",
        "allowedValues": [
            "Standard_LRS",
            "Standard_GRS",
            "Standard_ZRS"
        ],
        "type": "string",
        "metadata": {
            "description": "Storage Account type"
        }
    }
},
```

Add a variable that constructs the storage account name with the uniqueString function:

```
"variables": {
    "storageAccountName": "[concat(uniquestring(resourceGroup().id), 'standardsa')]"
},
```

Set the name of the storage account to the variable:

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

Set the SKU to the parameter:

```
"sku": {
    "name": "[parameters('storageAccountType')]",
    "tier": "Standard"
},
```

Your template now looks like:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "storageSuffix": {
            "type": "string",
            "defaultValue": "standardsa"
        },
        "storageAccountType": {
            "defaultValue": "Standard_LRS",
            "allowedValues": [
                "Standard_LRS",
                "Standard_GRS",
                "Standard_ZRS"
            ],
            "type": "string",
            "metadata": {
                "description": "Storage Account type"
            }
        }
    },
    "variables": {
        "storageAccountName": "[concat(uniquestring(resourceGroup().id), parameters('storageSuffix'))]"
    },
    "resources": [
        {
            "type": "Microsoft.Storage/storageAccounts",
            "sku": {
                "name": "[parameters('storageAccountType')]",
                "tier": "Standard"
            },
            "kind": "Storage",
            "name": "[variables('storageAccountName')]",
            "apiVersion": "2016-01-01",
            "location": "[resourceGroup().location]",
            "tags": {},
            "properties": {},
            "dependsOn": []
        }
    ]
}
```

Redeploy the modified template.

Next steps

- For information about using the portal to export a template, see [Export an Azure Resource Manager template from existing resources](#).
- To define parameters in template, see [Authoring templates](#).
- For tips on resolving common deployment errors, see [Troubleshoot common Azure deployment errors with Azure Resource Manager](#).

Deploy resources with Resource Manager templates and Azure CLI

8/2/2018 • 7 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 are not familiar with the concepts of deploying and managing your Azure solutions, see [Azure Resource Manager overview](#).

The Resource Manager template you deploy can either be a local file on your machine, or an external file that is located in a repository like GitHub. The template you deploy in this article is available in the [Sample template](#) section, or as a [storage account template in GitHub](#).

To run this sample, make sure you have installed the latest [Azure CLI 2.0](#). To start, run `az login` to create a connection with Azure.

This sample works in a Bash shell. For options on running Azure CLI scripts on Windows client, see [Install the Azure CLI on Windows](#).

If you do not have Azure CLI installed, you can use the [Cloud Shell](#).

Deploy local template

When deploying resources to Azure, you:

1. Log 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 cannot 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 login

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 external 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 login

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 should not include sensitive data. If you need to specify sensitive data (like an admin password), pass that value as a secure parameter. However, if you do not 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. However, you must first load your template into the storage account for your Cloud Shell. If you have not 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>`.

The screenshot shows the Microsoft Azure Resource groups interface. On the left is a navigation bar with icons for Home, All resources, Storage, Functions, and Container Registry. The main area has a dark header with the text 'Microsoft Azure Resource groups'. Below the header is a toolbar with 'Add', 'Assign Tags', 'Columns', and a refresh icon. A message 'Subscriptions: 6 of 7 selected – Don't see a subscription?' is displayed. There is a 'Filter by name...' input field. A table lists 11 items, with the 'NAME' column showing icons for dashboards and storage accounts. The row for 'cloud-shell-storage-westus' is highlighted with a red rectangle.

3. Select the storage account for your Cloud Shell.

cloud-shell-storage-westus

Resource group

Search (Ctrl+ /)

Overview

Activity log

Access control (IAM)

Tags

SETTINGS

Quickstart

Resource costs

Add Assign Tags Columns

Subscription name (change)
Third Internal Consumption
Subscription ID

Filter by name...

1 items

NAME cs40773a725d727x4eb8x8cb

4. Select **Blobs**.

cs40773a725d727x4eb8x8cb

Storage account

Search (Ctrl+ /)

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

SETTINGS

Access keys

Configuration

Encryption

Shared access signature

Open in Explorer Move Delete storage account

Resource group (change)
cloud-shell-storage-westus

Status
Primary: Available

Location
West US

Subscription (change)
Third Internal Consumption

Subscription ID

Services

Blobs

REST-based object storage for unstructured data

View metrics
Configure CORS rules
Setup custom domain

5. Select **+ Container**.

Blob service

cs40773a725d727x4eb8x8cb

+ Container Refresh Delete

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**.

Blob service
cs40773a725d727x4eb8x8cb

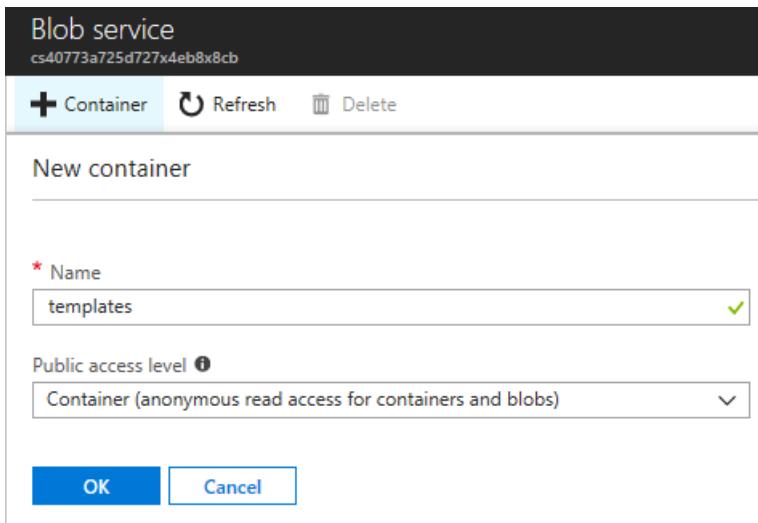
+ Container Refresh Delete

New container

* Name: templates ✓

Public access level: Container (anonymous read access for containers and blobs)

OK Cancel



7. Select the container you created.

Blob service
cs40773a725d727x4eb8x8cb

+ Container Refresh Delete

Storage account: cs40773a725d727x4eb8x8cb

Status: Primary: Available

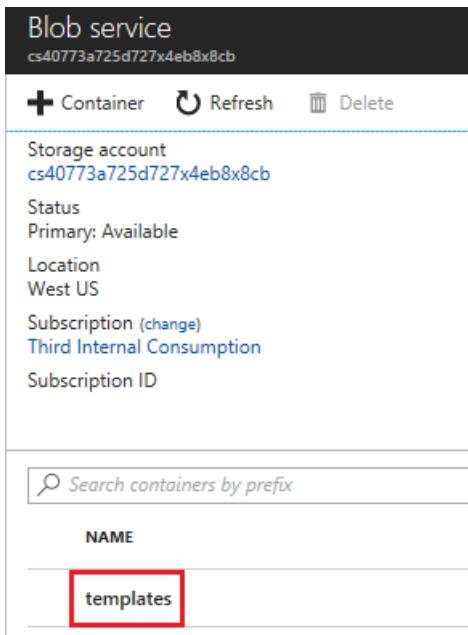
Location: West US

Subscription: Third Internal Consumption

Subscription ID

Search containers by prefix

NAME
templates



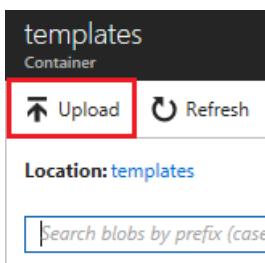
8. Select **Upload**.

templates
Container

Upload Refresh

Location: templates

Search blobs by prefix (case)



9. Find and upload your template.

Upload blob

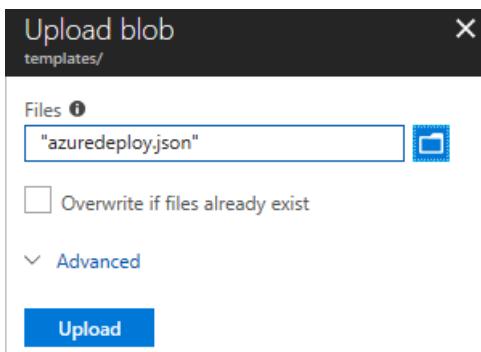
templates/

Files: "azuredeploy.json"

Overwrite if files already exist

Advanced

Upload



10. After it has uploaded, select the template.

A screenshot of the Azure Storage Explorer interface. The left sidebar shows a 'Container' named 'templates'. Below it are buttons for 'Upload', 'Refresh', and 'Delete'. A search bar says 'Search blobs by prefix (case-sensitive)'. A table lists a single item under 'NAME': 'azuredeploy.json', which is highlighted with a red box.

11. Copy the URL.

A screenshot of the Azure Storage Explorer showing the properties of the 'azuredeploy.json' blob. The top navigation bar includes 'Save', 'Discard', 'Refresh', 'Download', 'Acquire lease', and 'Break lease'. Below it are tabs for 'Overview' (which is selected and highlighted with a blue border), 'Synchronizations', and 'Edit blob'. Under 'Properties', there is a 'URL' field containing a long URL, and a 'LAST MODIFIED' field showing '3/30/2018 12:53:25 PM'. A blue copy icon is located next to the URL field, and both the URL field and the copy icon are highlighted with red boxes.

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
```

Deploy to more than one resource group or subscription

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. You can deploy to only five resource groups in a single deployment. For more information, see [Deploy Azure resources to more than one subscription or resource group](#).

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 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 multiple parameter files for different deployment scenarios, and then pass in the appropriate parameter file.

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
```

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, attempting to pass 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 could not 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  
}
```

Incremental and complete deployments

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 are not in the template:

- In complete mode, Resource Manager **deletes** resources that exist in the resource group but are not specified in the template.
- In incremental mode, Resource Manager **leaves unchanged** resources that exist in the resource group but are not specified in the template.

For both modes, Resource Manager attempts to provision all resources specified in the template. If the resource already exists in the resource group and its settings are unchanged, the operation results in no change. If you change the settings for a resource, the resource is provisioned with those new settings. If you attempt 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.

By default, Resource Manager uses the incremental mode.

To illustrate the difference between incremental and complete modes, consider the following scenario.

Existing Resource Group contains:

- Resource A
- Resource B
- Resource C

Template defines:

- Resource A

- Resource B
- Resource D

When deployed in **incremental** mode, the resource group contains:

- Resource A
- Resource B
- Resource C
- Resource D

When deployed in **complete** mode, Resource C is deleted. The resource group contains:

- Resource A
- Resource B
- Resource D

To use complete mode, use the `mode` parameter:

```
az group deployment create \
  --name ExampleDeployment \
  --mode Complete \
  --resource-group ExampleGroup \
  --template-file storage.json \
  --parameters storageAccountType=Standard_GRS
```

Sample template

The following template is used for the examples in this article. Copy and save it as a file named storage.json. To understand how to create this template, see [Create your first Azure Resource Manager 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"
            }
        }
    },
    "variables": {
        "storageAccountName": "[concat(uniquestring(resourceGroup().id), 'standardsa')]"
    },
    "resources": [
        {
            "type": "Microsoft.Storage/storageAccounts",
            "name": "[variables('storageAccountName')]",
            "apiVersion": "2016-01-01",
            "location": "[resourceGroup().location]",
            "sku": {
                "name": "[parameters('storageAccountType')]"
            },
            "kind": "Storage",
            "properties": {}
        }
    ],
    "outputs": {
        "storageAccountName": {
            "type": "string",
            "value": "[variables('storageAccountName')]"
        }
    }
}
```

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](#).
- For a complete sample script that deploys a template, see [Resource Manager template deployment script](#).
- 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](#).
- For guidance on how enterprises can use Resource Manager to effectively manage subscriptions, see [Azure enterprise scaffold - prescriptive subscription governance](#).

Deploy private Resource Manager template with SAS token and Azure CLI

5/21/2018 • 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 vmlinu.json \
    --expiry $expiretime \
    --permissions r \
    --output tsv \
    --connection-string $connection)
url=$(az storage blob url \
    --container-name templates \
    --name vmlinu.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](#).

Export Azure Resource Manager templates with Azure CLI

8/2/2018 • 4 minutes to read • [Edit Online](#)

Resource Manager enables you to export a Resource Manager template from existing resources in your subscription. You can use that generated template to learn about the template syntax or to automate the redeployment of your solution as needed.

It's important to note that there are two different ways to export a template:

- You can export the **actual template used for a deployment**. The exported template includes all the parameters and variables exactly as they appeared in the original template. This approach is helpful when you need to retrieve a template.
- You can export a **generated template that represents the current state of the resource group**. The exported template is not based on any template that you used for deployment. Instead, it creates a template that is a "snapshot" or "backup" of the resource group. The exported template has many hard-coded values and probably not as many parameters as you would typically define. Use this option to redeploy resources to the same resource group. To use this template for another resource group, you may have to significantly modify it.

This article shows both approaches.

Deploy a solution

To illustrate both approaches for exporting a template, let's start by deploying a solution to your subscription. If you already have a resource group in your subscription that you want to export, you do not have to deploy this solution. However, the rest of this article refers to the template for this solution. The example script deploys a storage account.

```
az group create --name ExampleGroup --location "Central US"
az group deployment create \
    --name NewStorage \
    --resource-group ExampleGroup \
    --template-uri "https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-storage-
account-create/azuredeploy.json" \
```

Save template from deployment history

You can retrieve a template from your deployment history by using the [az group deployment export](#) command. The following example saves the template that you previously deploy:

```
az group deployment export --name NewStorage --resource-group ExampleGroup
```

It returns the template. Copy the JSON, and save as a file. Notice that it's the exact template you used for deployment. The parameters and variables match the template from GitHub. You can redeploy this template.

Export resource group as template

Instead of retrieving a template from the deployment history, you can retrieve a template that represents the current state of a resource group by using the [az group export](#) command. You use this command when you have

made many changes to your resource group and no existing template represents all the changes. It is intended as a snapshot of the resource group, which you can use to redeploy to the same resource group. To use the exported template for other solutions, you must significantly modify it.

```
az group export --name ExampleGroup
```

It returns the template. Copy the JSON, and save as a file. Notice that it's different than the template in GitHub. The template has different parameters and no variables. The storage SKU and location are hard-coded to values. The following example shows the exported template, but your template has a slightly different parameter name:

```
{
  "parameters": {
    "storageAccounts_mcyzaljiv7qncstandardsa_name": {
      "type": "String",
      "defaultValue": null
    }
  },
  "variables": {},
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "resources": [
    {
      "tags": {},
      "dependsOn": [],
      "apiVersion": "2016-01-01",
      "sku": {
        "name": "Standard_LRS",
        "tier": "Standard"
      },
      "kind": "Storage",
      "type": "Microsoft.Storage/storageAccounts",
      "location": "centralus",
      "name": "[parameters('storageAccounts_mcyzaljiv7qncstandardsa_name')]",
      "properties": {}
    }
  ],
  "contentVersion": "1.0.0.0"
}
```

You can redeploy this template, but it requires guessing a unique name for the storage account. The name of your parameter is slightly different.

```
az group deployment create --name NewStorage --resource-group ExampleGroup \
--template-file examplegroup.json \
--parameters "{\"storageAccounts_mcyzaljiv7qncstandardsa_name\":{\"value\":\"tfstore0501\"}}"
```

Customize exported template

You can modify this template to make it easier to use and more flexible. To allow for more locations, change the location property to use the same location as the resource group:

```
"location": "[resourceGroup().location]",
```

To avoid having to guess a unique name for storage account, remove the parameter for the storage account name. Add a parameter for a storage name suffix, and a storage SKU:

```
"parameters": {  
    "storageSuffix": {  
        "type": "string",  
        "defaultValue": "standardsa"  
    },  
    "storageAccountType": {  
        "defaultValue": "Standard_LRS",  
        "allowedValues": [  
            "Standard_LRS",  
            "Standard_GRS",  
            "Standard_ZRS"  
        ],  
        "type": "string",  
        "metadata": {  
            "description": "Storage Account type"  
        }  
    }  
},
```

Add a variable that constructs the storage account name with the uniqueString function:

```
"variables": {  
    "storageAccountName": "[concat(uniquestring(resourceGroup().id), 'standardsa')]"  
},
```

Set the name of the storage account to the variable:

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

Set the SKU to the parameter:

```
"sku": {  
    "name": "[parameters('storageAccountType')]",  
    "tier": "Standard"  
},
```

Your template now looks like:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "storageSuffix": {
            "type": "string",
            "defaultValue": "standardsa"
        },
        "storageAccountType": {
            "defaultValue": "Standard_LRS",
            "allowedValues": [
                "Standard_LRS",
                "Standard_GRS",
                "Standard_ZRS"
            ],
            "type": "string",
            "metadata": {
                "description": "Storage Account type"
            }
        }
    },
    "variables": {
        "storageAccountName": "[concat(uniquestring(resourceGroup().id), parameters('storageSuffix'))]"
    },
    "resources": [
        {
            "type": "Microsoft.Storage/storageAccounts",
            "sku": {
                "name": "[parameters('storageAccountType')]",
                "tier": "Standard"
            },
            "kind": "Storage",
            "name": "[variables('storageAccountName')]",
            "apiVersion": "2016-01-01",
            "location": "[resourceGroup().location]",
            "tags": {},
            "properties": {},
            "dependsOn": []
        }
    ]
}
```

Redeploy the modified template.

Next steps

- For information about using the portal to export a template, see [Export an Azure Resource Manager template from existing resources](#).
- To define parameters in template, see [Authoring templates](#).
- For tips on resolving common deployment errors, see [Troubleshoot common Azure deployment errors with Azure Resource Manager](#).

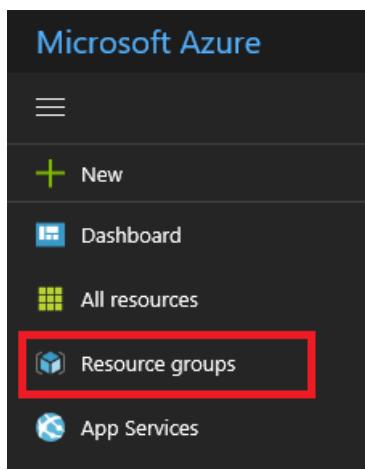
Deploy resources with Resource Manager templates and Azure portal

8/3/2018 • 3 minutes to read • [Edit Online](#)

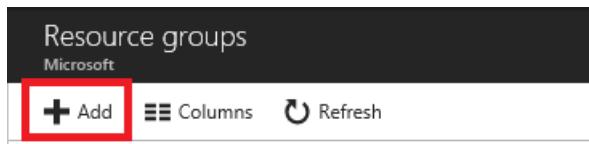
This article shows how to use the [Azure portal](#) with [Azure Resource Manager](#) to deploy your Azure resources. To learn about managing your resources, see [Manage Azure resources through portal](#).

Create resource group

1. To create an empty resource group, select **Resource groups**.



2. Under Resource groups, select **Add**.



3. Give it a name and location, and, if necessary, select a subscription. You need to provide a location for the resource group because 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.

Resource group X

Create an empty resource group

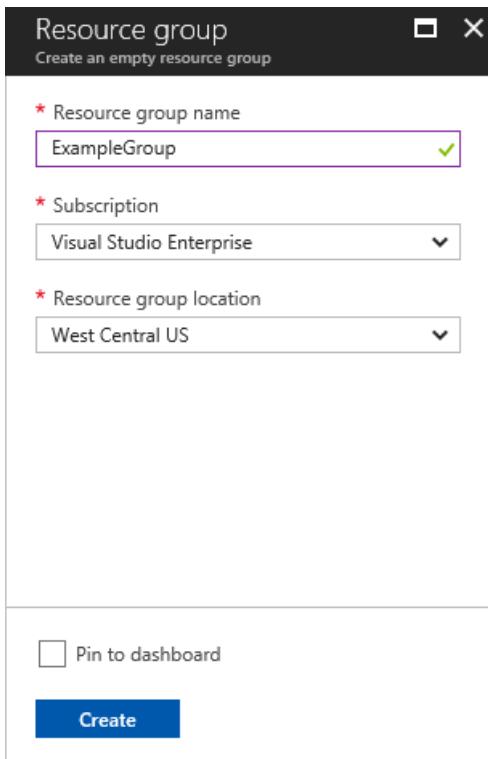
* Resource group name
ExampleGroup ✓

* Subscription
Visual Studio Enterprise ▼

* Resource group location
West Central US ▼

Pin to dashboard

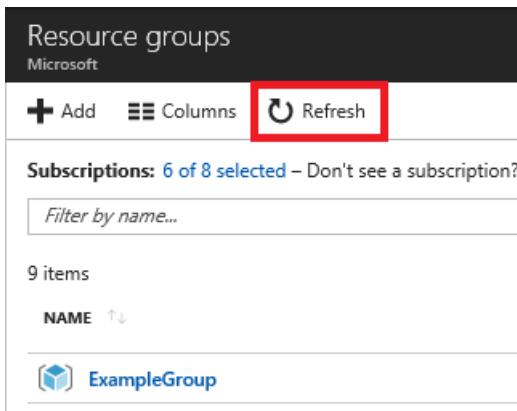
Create



When you have finished setting the properties, select **Create**.

4. To see your new resource group, select **Refresh**.

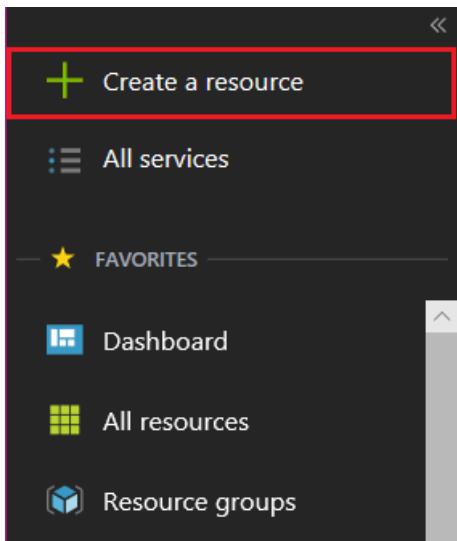
Resource groups	
Microsoft	
+ Add	Columns ↻ Refresh
Subscriptions: 6 of 8 selected – Don't see a subscription?	
<input type="text"/> Filter by name...	
9 items	
NAME ↑↓	
 ExampleGroup	



Deploy resources from Marketplace

After you create a resource group, you can deploy resources to it from the Marketplace. The Marketplace provides pre-defined solutions for common scenarios.

1. To start a deployment, select **Create a resource**.



2. Find the type of resource you would like to deploy.

A screenshot of the Azure Marketplace search interface. The title bar says 'New'. A search bar contains the placeholder 'Search the Marketplace'. Below it, there are tabs for 'Azure Marketplace' (selected), 'See all', 'Featured' (selected), and another 'See all'. Under 'Storage', which is highlighted with a blue dashed box, there are two items: 'Storage account - blob, file, table, queue' (with a green icon) and 'Data Lake Store' (with a green folder icon). Both have 'Details' buttons. Other categories like 'Networking' and 'Web + Mobile' are also visible.

3. If you don't see the particular solution you would like to deploy, you can search the Marketplace for it. For example, to find a Wordpress solution, start typing **Wordpress** and select the option you want.

A screenshot of the Azure Marketplace search interface. The title bar says 'New'. A search bar has 'Wordpre' typed into it. Below the search bar, a list of results includes 'Wordpre', 'WordPress', 'WordPress on Linux', 'WordPress Multi-Tier', and 'Wordpress on Windows 2012 R2'. The first result, 'Wordpre', is highlighted with a blue dashed box.

4. 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 web app and deploy it to the resource group you created.

Create storage account

The cost of your storage account depends on the usage and the options you choose below.

[Learn more](#)

* Name [?](#)
storage0908example

.core.windows.net

Deployment model [?](#)
 Resource manager Classic

Account kind [?](#)

Performance [?](#)
 Standard Premium

Replication [?](#)

* Secure transfer required [?](#)
 Disabled Enabled

* Subscription

* Resource group
 Create new Use existing

* Location

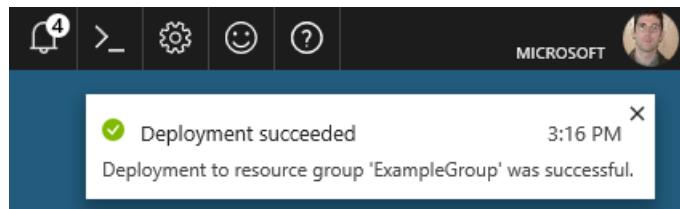
Pin to dashboard

Create [Automation options](#)

Alternatively, you can decide to create a resource group when deploying your resources. Select **Create new** and give the resource group a name.

* Resource Group [?](#)
 Create new Use existing

5. Your deployment begins. The deployment could take a few minutes. When the deployment has finished, you see a notification.



6. After deploying your resources, you can add more resources to the resource group by selecting **Add**.

The screenshot shows the 'ExampleGroup' Resource group overview. At the top right, there is a red box around the '+ Add' button. Below it, there are options for 'Columns' and 'Delete resource'. The 'Essentials' section displays the 'Subscription name (change)' as 'Visual Studio Enterprise' and the 'Subscription ID'. On the left, there's a sidebar with 'Overview', 'Activity log', and 'Access control (IAM)'.

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**, and search for **Template Deployment** until you can select it from the options.

The screenshot shows the Microsoft Azure portal with the search bar containing 'template'. The results list includes 'template' and 'Template deployment', with 'Template deployment' highlighted by a red box. Other results include 'Puppet Enterprise 2017.2 Template', 'PrestaShop Kickstart Template', and 'PrestaShop Advanced Template'. The left sidebar shows navigation options like 'All services', 'Dashboard', 'All resources', 'Resource groups', 'App Services', and 'SQL databases'. A 'Web App' icon with a 'Quickstart tutorial' link is visible at the bottom right.

2. Select **Create**.

The screenshot shows the Microsoft Azure 'Template deployment' page. At the top, there's a header with the Microsoft logo and the title 'Template deployment'. Below the header, a paragraph explains that Azure Resource Manager templates enable deploying and managing resources as a group. A note says to edit the template with IntelliSense and deploy it to a new or existing resource group. Below this, there are social sharing icons for Twitter, Facebook, LinkedIn, YouTube, Google+, and Email. A horizontal line separates this from the main content area. In the content area, there are three rows of information: 'PUBLISHER' (Microsoft), 'LOGICAPPSUPPORTED' (none), and 'USEFUL LINKS' (Documentation). At the bottom of the content area is a blue 'Create' button, which is highlighted with a red rectangular border.

3. You see several options for creating a template. Select **Build your own template in the editor**.

The screenshot shows the Microsoft Azure 'Custom deployment' page. The top navigation bar has the title 'Custom deployment' and a sub-link 'Deploy from a custom template'. Below the navigation, there's a section titled 'Learn about template deployment' with two options: 'Read the docs' and 'Build your own template in the editor'. The 'Build your own template in the editor' option is highlighted with a red rectangular border. Below this, there's a section titled 'Common templates' with four items: 'Create a Linux virtual machine', 'Create a Windows virtual machine', 'Create a web app', and 'Create a SQL database', each with its own icon. At the bottom, there's a section titled 'Load a GitHub quickstart template' with a dropdown menu labeled 'Select a template (disclaimer)' and a search bar with the placeholder 'Type to start filtering...'. The 'Build your own template in the editor' button is also highlighted with a red rectangular border.

4. You have a blank template that is available for customizing.

Edit template
Edit your Azure Resource Manager template

Add resource Quickstart template Load file Download

Parameters (0) Variables (0) Resources (0)

```
1 {
2   "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deployment.json#",
3   "contentVersion": "1.0.0.0",
4   "parameters": {},
5   "resources": []
6 }
```

5. You can edit the JSON syntax manually, or select a pre-built template from the [Quickstart template gallery](#). However, for this article, you use the **Add resource** option.

Edit template
Edit your Azure Resource Manager template

Add resource Quickstart template Load file Download

Parameters (0) Variables (0) Resources (0)

```
1 {
2   "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deployment.json#",
3   "contentVersion": "1.0.0.0",
4   "parameters": {},
5   "resources": []
6 }
```

6. Select **Storage account** and provide a name. When finished providing values, select **OK**.

Edit template
Edit your Azure Resource Manager template

Add resource Quickstart template Load file Download

Add a resource to the template

* Select a resource

Storage account

 Creates a storage account.

* Name:

OK Cancel

7. The editor automatically adds JSON for the resource type. Notice that it includes a parameter for defining the type of storage account. Select **Save**.

Edit template

Edit your Azure Resource Manager template

[+ Add resource](#) [↑ Quickstart template](#) [↑ Load file](#) [Download](#)

► Parameters (1)

► Variables (1)

▼ Resources (1)

mystorage (Microsoft.Storage/stor...

```
1 [
2   "$schema": "https://schema.management.azure.com/schemas/2015-01-01-deploymentTemplate.json#",
3   "contentVersion": "1.0.0.0",
4   "parameters": {
5     "mystorageType": {
6       "type": "string",
7       "defaultValue": "Standard_LRS",
8       "allowedValues": [
9         "Standard_LRS",
10        "Standard_ZRS",
11        "Standard_GRS",
12        "Standard_RAGRS",
13        "Premium_LRS"
14      ]
15    }
16  },
17  "resources": [
18    {
19      "name": "[variables('mystorageName')]",
20      "type": "Microsoft.Storage/storageAccounts",
21      "location": "[resourceGroup().location]",
22      "apiVersion": "2015-06-15",
23      "dependsOn": [],
24      "tags": {
25        "displayName": "mystorage"
26      },
27      "properties": {
28        "accountType": "[parameters('mystorageType')]"
29      }
30    }
31  ],
32  "variables": {
33    "mystorageName": "[concat('mystorage', uniqueString(resourceGroup().id))]"
34  }
35 ]
```

[Save](#)

[Discard](#)

8. Now, you have the option to deploy the resources defined in the template. To deploy, agree to the terms and conditions, and select **Purchase**.

Custom deployment

Deploy from a custom template

TEMPLATE



Customized template

1 resource



Edit template



Edit parameters



Learn more

BASICS

* Subscription

Visual Studio Enterprise

* Resource group

Create new Use existing

ExampleGroup

* Location

West Central US

SETTINGS

Mystorage Type

Standard_LRS

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.

Microsoft assumes no responsibility for any actions performed by third-party templates and does not provide rights for third-

I agree to the terms and conditions stated above

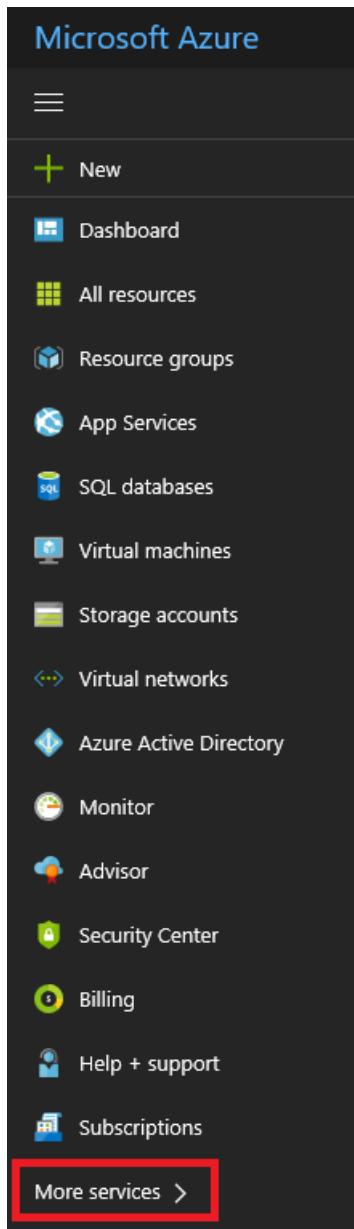
Pin to dashboard

Purchase

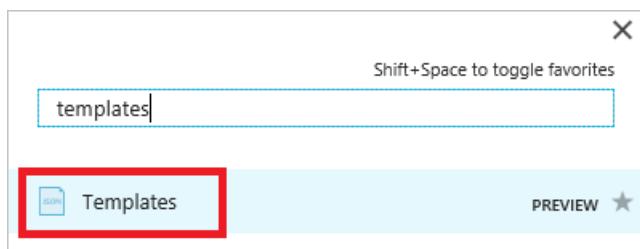
Deploy resources from a template saved to your account

The portal enables you to save a template to your Azure account, and redeploy it later. For more information on templates, see [Create and deploy your first Azure Resource Manager template](#).

1. To find your saved templates, select **More services**.



2. Search for **templates** and select that option.



3. From the list of templates saved to your account, select the one you wish to work on.

Templates

Microsoft - PREVIEW

+ Add **Columns** **Refresh**

Directory: Microsoft – [Switch directories](#)

5 items

NAME	↑↓
batchtest	
crossrg	
myvnettest	
storagedemo	
webandsql	

4. Select **Deploy** to redeploy this saved template.

[storagedemo](#)
Template - PREVIEW

Deploy **Edit** **More**

DESCRIPTION
a test template

PUBLISHER

MODIFIED
5/3/2017

[View Template](#)

Next steps

- To view audit logs, see [Audit operations with Resource Manager](#).
- To troubleshoot deployment errors, see [View deployment operations](#).
- To retrieve a template from a deployment or resource group, see [Export Azure Resource Manager template from existing resources](#).
- For guidance on how enterprises can use Resource Manager to effectively manage subscriptions, see [Azure enterprise scaffold - prescriptive subscription governance](#).

Export an Azure Resource Manager template from existing resources

6/26/2018 • 7 minutes to read • [Edit Online](#)

In this article, you learn how to export a Resource Manager template from existing resources in your subscription. You can use that generated template to gain a better understanding of template syntax.

There are two ways to export a template:

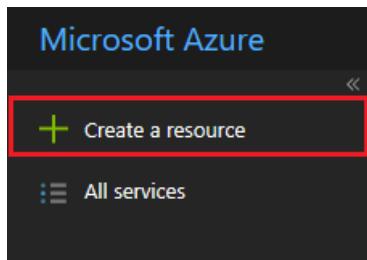
- You can export the **actual template used for deployment**. The exported template includes all the parameters and variables exactly as they appeared in the original template. This approach is helpful when you deployed resources through the portal, and want to see the template to create those resources. This template is readily usable.
- You can export a **generated template that represents the current state of the resource group**. The exported template isn't based on any template that you used for deployment. Instead, it creates a template that is a "snapshot" or "backup" of the resource group. The exported template has many hard-coded values and probably not as many parameters as you would typically define. Use this option to redeploy resources to the same resource group. To use this template for another resource group, you may have to significantly modify it.

This article shows both approaches through the portal.

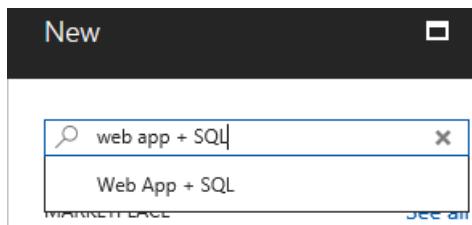
Deploy resources

Let's start by deploying resources to Azure that you can use for exporting as a template. If you already have a resource group in your subscription that you want to export to a template, you can skip this section. The rest of this article assumes you've deployed the web app and SQL database solution shown in this section. If you use a different solution, your experience might be a little different, but the steps to export a template are the same.

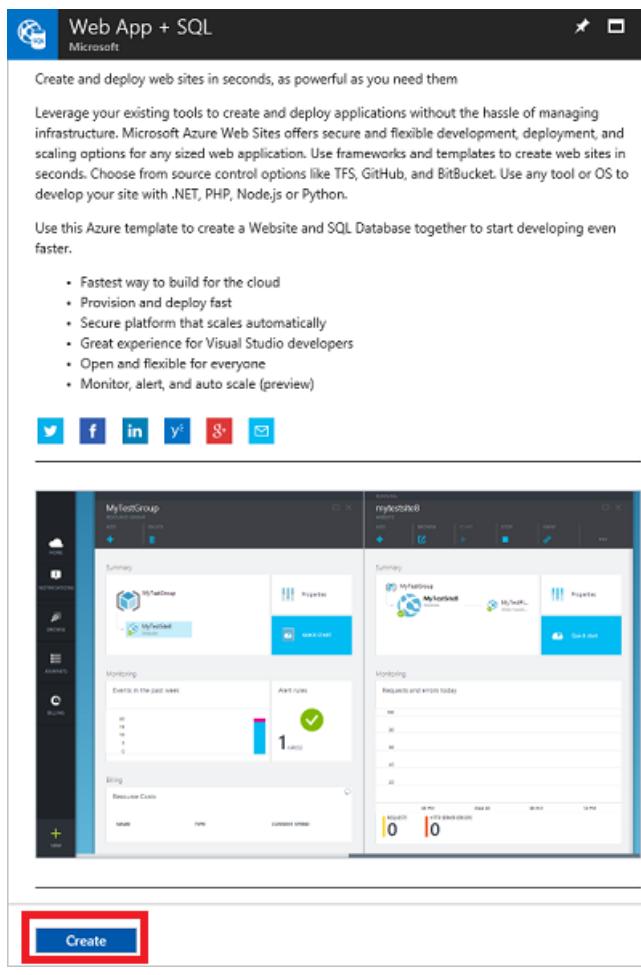
1. In the [Azure portal](#), select **Create a resource**.



2. Search for **web app + SQL** and select it from the available options.



3. Select **Create**.



4. Provide the required values for the web app and SQL database. Select **Create**.

Web App + SQL

Create

* App name
exportsite .azurewebsites.net

* Subscription
Microsoft Azure Consumption

* Resource Group
 Create new Use existing
exportsite

* App Service plan/Location
ServicePlan2a307b08-bb2c(Sout...)

* SQL Database
exportdatabase

Application Insights
 On Off

Pin to dashboard

Create Automation options

The deployment may take a minute. After the deployment finishes, your subscription contains the solution.

View template from deployment history

1. Go to the resource group for your new resource group. Notice that the portal shows the result of the last deployment. Select this link.

A screenshot of the Azure Resource Group Overview page for the 'exportsite' resource group. The top navigation bar shows the resource group name. Below it is a search bar and a toolbar with 'Add', 'Columns', 'Delete', 'Refresh', and 'Move' buttons. A sidebar on the left lists 'Overview', 'Activity log', and 'Access control (IAM)'. The main content area displays 'Essentials' information: 'Subscription name (change)', 'Microsoft Azure Consumption', and 'Subscription ID'. To the right, a 'Deployments' section is shown with a red box highlighting '1 Succeeded'.

2. You see a history of deployments for the group. In your case, the portal probably lists only one deployment. Select this deployment.

A screenshot of the Azure Deployment History page. It shows a table with columns 'DEPLOYMENT NAME' and 'STATUS'. A single row is listed: 'Microsoft.WebSiteSQLDatabased1...' with a status of 'Succeeded'. The 'View template' button is visible at the top of the table.

3. The portal displays a summary of the deployment. The summary includes the status of the deployment and its operations and the values that you provided for parameters. To see the template that you used for the deployment, select **View template**.

A screenshot of the Azure Deployment Summary page for the deployment 'Microsoft.WebSiteSQLDatabased13386b0-9908'. The top navigation bar shows the deployment name. Below it is a toolbar with 'Delete', 'Cancel', 'Refresh', 'Redeploy', and a redboxed 'View template' button. On the left is a sidebar with icons for different services. The main content area shows deployment details: DEPLOYMENT DATE (7/5/2017 4:01:15 PM), STATUS (Succeeded), DURATION (1 minute 30 seconds), RESOURCE GROUP (exportsite), and RELATED (Events). The 'View template' button is highlighted with a red box.

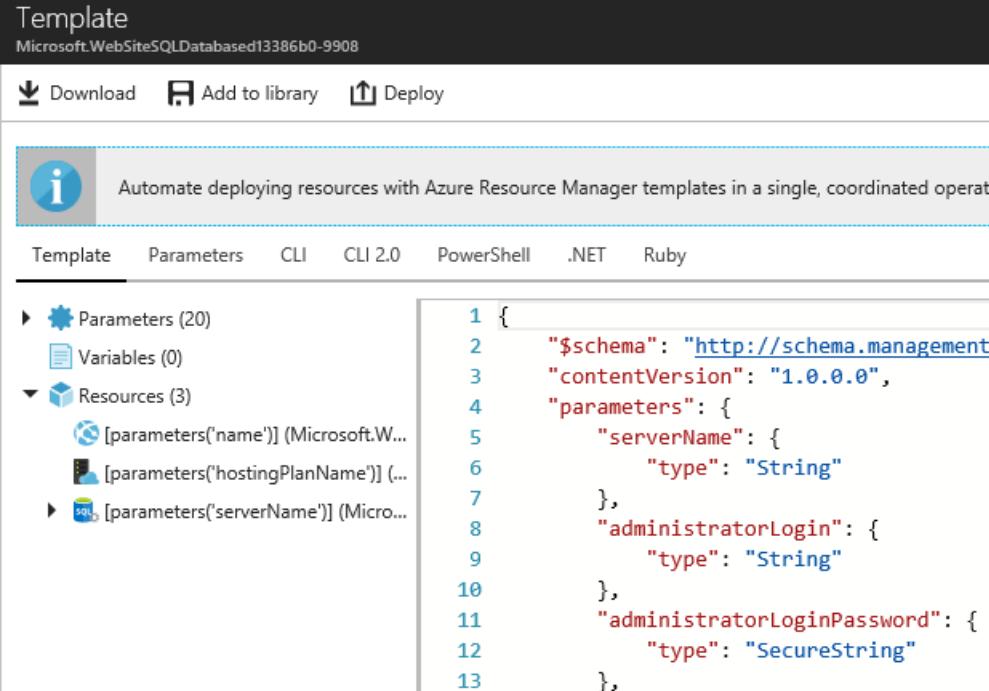
4. Resource Manager retrieves the following seven files for you:

- a. **Template** - The template that defines the infrastructure for your solution. When you created the storage account through the portal, Resource Manager used a template to deploy it and saved that template for future reference.
- b. **Parameters** - A parameter file that you can use to pass in values during deployment. It contains the values that you provided during the first deployment. You can change any of these values when you

redeploy the template.

- c. **CLI** - An Azure CLI script file that you can use to deploy the template.
- d. **PowerShell** - An Azure PowerShell script file that you can use to deploy the template.
- e. **.NET** - A .NET class that you can use to deploy the template.
- f. **Ruby** - A Ruby class that you can use to deploy the template.

By default, the portal displays the template.



```
1 {
2     "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
3     "contentVersion": "1.0.0.0",
4     "parameters": {
5         "serverName": {
6             "type": "String"
7         },
8         "administratorLogin": {
9             "type": "String"
10        },
11        "administratorLoginPassword": {
12            "type": "SecureString"
13        }
14    }
15}
```

This template is the actual template used to create your web app and SQL database. Notice it contains parameters that enable you to provide different values during deployment. To learn more about the structure of a template, see [Authoring Azure Resource Manager templates](#).

Export the template from resource group

If you've manually changed your resources or added resources in multiple deployments, retrieving a template from the deployment history doesn't reflect the current state of the resource group. This section shows you how to export a template that reflects the current state of the resource group. It is intended as a snapshot of the resource group, which you can use to redeploy to the same resource group. To use the exported template for other solutions, you must significantly modify it.

NOTE

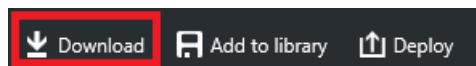
You can't export a template for a resource group that has more than 200 resources.

1. To view the template for a resource group, select **Automation script**.

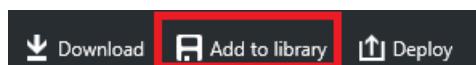
Resource Manager evaluates the resources in the resource group, and generates a template for those resources. Not all resource types support the export template function. You may see an error stating that there is a problem with the export. You learn how to handle those issues in the [Fix export issues](#) section.

2. You again see the six files that you can use to redeploy the solution. However, this time the template is a little different. Notice that the generated template contains fewer parameters than the template in previous section. Also, many of the values (like location and SKU values) are hard-coded in this template rather than accepting a parameter value. Before reusing this template, you might want to edit the template to make better use of parameters.
3. You have a couple of options for continuing to work with this template. You can either download the template and work on it locally with a JSON editor. Or, you can save the template to your library and work on it through the portal.

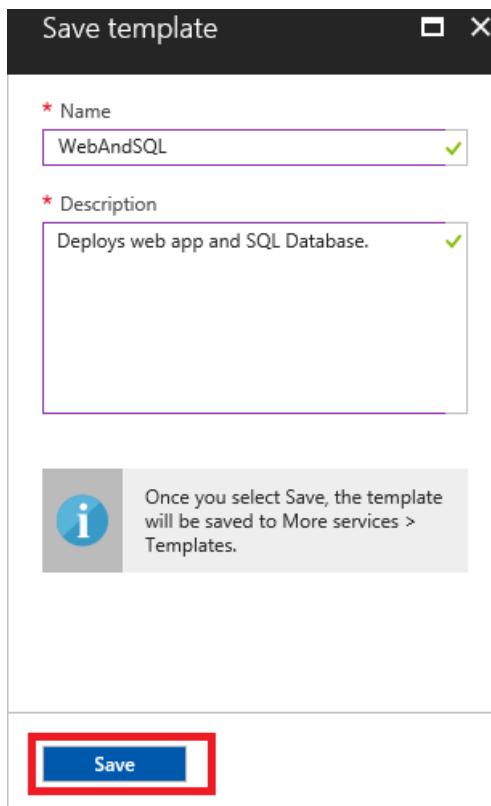
If you're comfortable using a JSON editor like [VS Code](#) or [Visual Studio](#), you might prefer downloading the template locally and using that editor. To work locally, select **Download**.



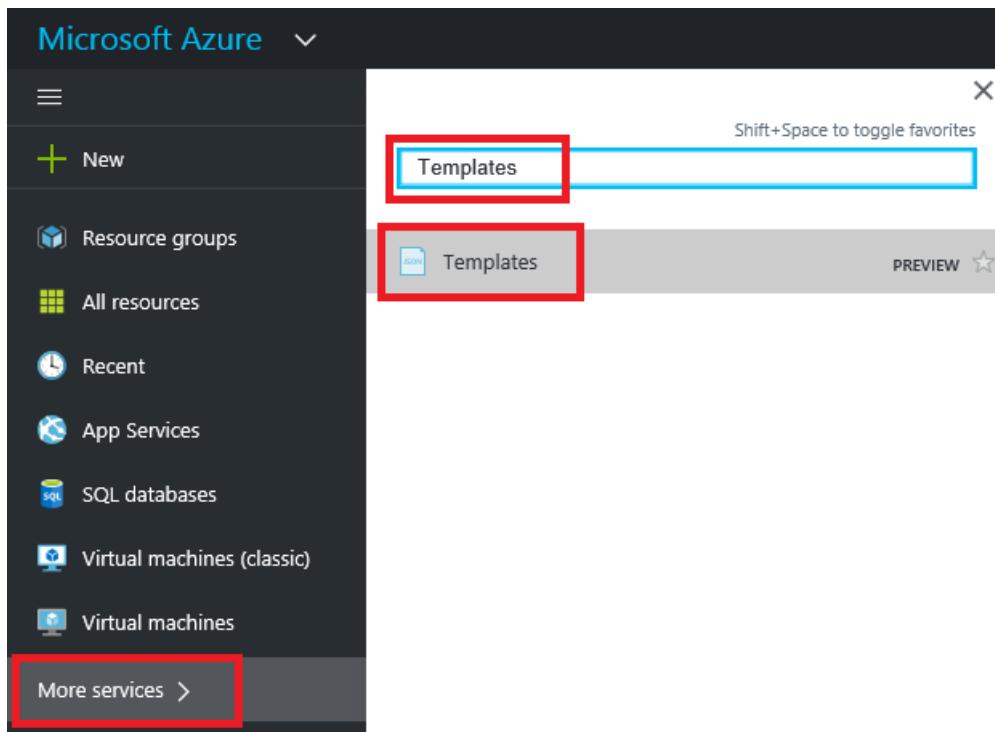
If you aren't set up with a JSON editor, you might prefer editing the template through the portal. The rest of this article assumes you've saved the template to your library in the portal. However, you make the same syntax changes to the template whether working locally with a JSON editor or through the portal. To work through the portal, select **Add to library**.



When adding a template to the library, you give the template a name and description. Then, select **Save**.



4. To view a template saved in your library, select **More services**, type **Templates** to filter results, select **Templates**.



5. Select the template with the name you saved.

The screenshot shows the 'Templates' blade in the Azure portal. At the top, there are buttons for 'Add', 'Columns', and 'Refresh'. Below that, a 'Directory' section shows 'Microsoft - Switch directories' and a 'Filter by name...' input field. A message indicates '4 items'. A table lists four templates: 'crossrg', 'myvnettest', 'storagedemo', and 'webandsql'. The 'webandsql' row is highlighted with a red box.

Customize the template

The exported template works fine if you want to create the same web app and SQL database for every deployment. However, Resource Manager provides options so that you can deploy templates with a lot more flexibility. This article shows you how to add parameters for the database administrator name and password. You can use this same approach to add more flexibility for other values in the template.

1. To customize the template, select **Edit**.

The screenshot shows the 'Edit Template' preview window for the 'webandsql' template. The window has a title bar 'webandsql Template - PREVIEW' with 'Deploy' and 'Edit' buttons. The 'Edit' button is highlighted with a red box. Below the buttons are sections for 'DESCRIPTION' (Deployes web app and SQL Database), 'PUBLISHER' (a placeholder box with a file icon), 'MODIFIED' (7/6/2017), and a 'View Template' link at the bottom.

2. Select the template.

The screenshot shows the 'Edit Template' preview window with the title 'Edit Template PREVIEW'. It includes 'Save' and 'Discard' buttons. Below is a navigation menu with 'General' and 'webandsql' items, followed by a large red box highlighting the 'ARM Template' section which contains the text 'Template added'.

3. To pass values that you might want to specify during deployment, add the following two parameters to the

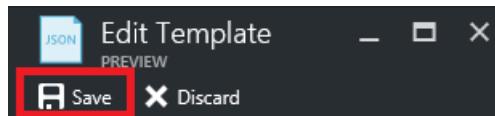
parameters section in the template:

```
"administratorLogin": {  
    "type": "String"  
},  
"administratorLoginPassword": {  
    "type": "SecureString"  
},
```

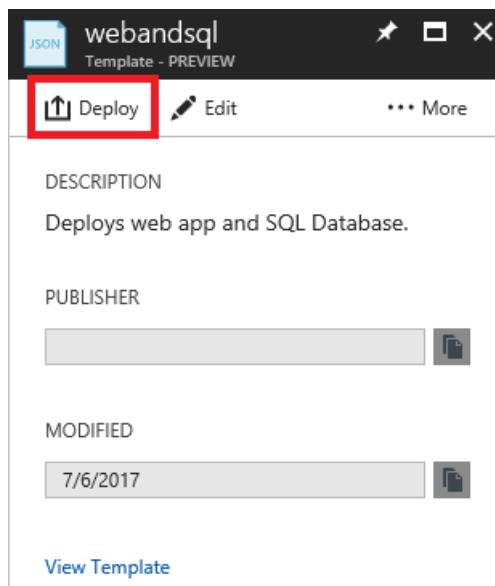
4. To use the new parameters, replace the SQL server definition in the **resources** section. Notice that **administratorLogin** and **administratorLoginPassword** now use parameter values.

```
{  
    "comments": "Generalized from resource: '/subscriptions/{subscription-  
id}/resourceGroups/exportsite/providers/Microsoft.Sql/servers/tfserverexport'.",  
    "type": "Microsoft.Sql/servers",  
    "kind": "V12.0",  
    "name": "[parameters('servers_tfserverexport_name')]",  
    "apiVersion": "2014-04-01-preview",  
    "location": "South Central US",  
    "scale": null,  
    "properties": {  
        "administratorLogin": "[parameters('administratorLogin')]",  
        "administratorLoginPassword": "[parameters('administratorLoginPassword')]",  
        "version": "12.0"  
    },  
    "dependsOn": []  
},
```

5. Select **OK** when you're done editing the template.
6. Select **Save** to save the changes to the template.



7. To redeploy the updated template, select **Deploy**.



8. Provide parameter values, and select a resource group to deploy the resources to.

Fix export issues

Not all resource types support the export template function. You only see export issues when exporting from a resource group rather than from your deployment history. If your last deployment accurately represents the current state of the resource group, you should export the template from the deployment history rather than from the resource group. Only export from a resource group when you have made changes to the resource group that aren't defined in a single template.

To resolve export issues, manually add the missing resources back into your template. The error message includes the resource types that can't be exported. Find that resource type in [Template reference](#). For example, to manually add a virtual network gateway, see [Microsoft.Network/virtualNetworkGateways template reference](#). The template reference gives you the JSON to add the resource to your template.

After getting the JSON format for the resource, you need to get the resource values. You can see the values for the resource by using the GET operation in the REST API for the resource type. For example, to get the values for your virtual network gateway, see [Virtual Network Gateways - Get](#).

Next steps

- You can deploy a template through [PowerShell](#), [Azure CLI](#), or [REST API](#).
- To see how to export a template through PowerShell, see [Export Azure Resource Manager templates with PowerShell](#).
- To see how to export a template through Azure CLI, see [Export Azure Resource Manager templates with Azure CLI](#).

Deploy resources with Resource Manager templates and Resource Manager REST API

5/21/2018 • 4 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.

TIP

For help with debugging an error during deployment, see:

- [View deployment operations](#) to learn about getting information that helps you troubleshoot your error
- [Troubleshoot common errors when deploying resources to Azure with Azure Resource Manager](#) to learn how to resolve common deployment errors

Your template can be either a local file or an external file that is available through a URI. 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.

Incremental and complete deployments

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 are not in the template:

- In complete mode, Resource Manager **deletes** resources that exist in the resource group but are not specified in the template.
- In incremental mode, Resource Manager **leaves unchanged** resources that exist in the resource group but are not specified in the template.

For both modes, Resource Manager attempts to provision all resources specified in the template. If the resource already exists in the resource group and its settings are unchanged, the operation results in no change. If you change the settings for a resource, the resource is provisioned with those new settings. If you attempt 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.

By default, Resource Manager uses the incremental mode.

To illustrate the difference between incremental and complete modes, consider the following scenario.

Existing Resource Group contains:

- Resource A
- Resource B
- Resource C

Template defines:

- Resource A
- Resource B
- Resource D

When deployed in **incremental** mode, the resource group contains:

- Resource A
- Resource B
- Resource C
- Resource D

When deployed in **complete** mode, Resource C is deleted. The resource group contains:

- Resource A
- Resource B
- Resource D

Deploy with the REST API

1. Set [common parameters and headers](#), including authentication tokens.
2. If you do not 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=2015-01-01  
{  
  "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. Create a deployment. Provide your subscription ID, the name of the resource group, the name of the deployment, and a link to your template. For information about the template file, see [Parameter file](#). For more information about the REST API to create a resource group, see [Create a template deployment](#). 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 are not in your template.

```
PUT  
https://management.azure.com/subscriptions/<YourSubscriptionId>/resourcegroups/<YourResourceGroupName>/p  
roviders/Microsoft.Resources/deployments/<YourDeploymentName>?api-version=2015-01-01  
{  
  "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"  
    }  
  }  
}
```

If you want to log response content, request content, or both, include **debugSetting** in the request.

```
"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. Get the status of the template deployment. For more information, see [Get information about a template deployment](#).

```
GET  
https://management.azure.com/subscriptions/<YourSubscriptionId>/resourcegroups/<YourResourceGroupName>/providers/Microsoft.Resources/deployments/<YourDeploymentName>?api-version=2015-01-01
```

Redeploy when deployment fails

For deployments that fail, you can specify that an earlier deployment from your deployment history is automatically redeployed. 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:

```
"onErrorDeployment": {  
    "type": "LastSuccessful",  
},
```

To redeploy a specific deployment if the current deployment fails, use:

```
"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 learn about handling asynchronous REST operations, see [Track asynchronous Azure operations](#).
- For an example of deploying resources through the .NET client library, see [Deploy resources using .NET libraries and a template](#).
- To define parameters in template, see [Authoring templates](#).
- For guidance on deploying your solution to different environments, see [Development and test environments in Microsoft Azure](#).
- For guidance on how enterprises can use Resource Manager to effectively manage subscriptions, see [Azure enterprise scaffold - prescriptive subscription governance](#).

Deploy Azure resources to more than one subscription or resource group

6/13/2018 • 6 minutes to read • [Edit Online](#)

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"
    },
    "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.
Cross resource group properties template	Demonstrates how the <code>resourceGroup()</code> 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-AzureRmResourceGroup -Name $firstRG -Location southcentralus
New-AzureRmResourceGroup -Name $secondRG -Location eastus

New-AzureRmResourceGroupDeployment ` 
    -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-AzureRmSubscription -Subscription $secondSub
New-AzureRmResourceGroup -Name $secondRG -Location eastus

Select-AzureRmSubscription -Subscription $firstSub
New-AzureRmResourceGroup -Name $firstRG -Location southcentralus

New-AzureRmResourceGroupDeployment ` 
    -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-AzureRmResourceGroup -Name parentGroup -Location southcentralus
New-AzureRmResourceGroup -Name inlineGroup -Location southcentralus
New-AzureRmResourceGroup -Name linkedGroup -Location southcentralus

New-AzureRmResourceGroupDeployment ` 
    -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	<pre>{ "id": "/subscriptions/<subscription-id>/resourceGroups/parentGroup", "name": "parentGroup", "location": "southcentralus", "properties": { "provisioningState": "Succeeded" } }</pre>
inlineRG	Object	<pre>{ "id": "/subscriptions/<subscription-id>/resourceGroups/parentGroup", "name": "parentGroup", "location": "southcentralus", "properties": { "provisioningState": "Succeeded" } }</pre>
linkedRG	Object	<pre>{ "id": "/subscriptions/<subscription-id>/resourceGroups/linkedGroup", "name": "linkedGroup", "location": "southcentralus", "properties": { "provisioningState": "Succeeded" } }</pre>

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](#).

Continuous integration in Visual Studio Team Services using Azure Resource Group deployment projects

4/25/2018 • 8 minutes to read • [Edit Online](#)

To deploy an Azure template, you perform tasks in various stages: Build, Test, Copy to Azure (also called "Staging"), and Deploy Template. There are two different ways to deploy templates to Visual Studio Team Services (VS Team Services). Both methods provide the same results, so choose the one that best fits your workflow.

1. Add a single step to your build definition that runs the PowerShell script that's included in the Azure Resource Group deployment project (`Deploy-AzureResourceGroup.ps1`). The script copies artifacts and then deploys the template.
2. Add multiple VS Team Services build steps, each one performing a stage task.

This article demonstrates both options. The first option has the advantage of using the same script used by developers in Visual Studio and providing consistency throughout the lifecycle. The second option offers a convenient alternative to the built-in script. Both procedures assume you already have a Visual Studio deployment project checked into VS Team Services.

Copy artifacts to Azure

Regardless of the scenario, if you have any artifacts that are needed for template deployment, you must give Azure Resource Manager access to them. These artifacts can include files such as:

- Nested templates
- Configuration scripts and DSC scripts
- Application binaries

Nested Templates and Configuration Scripts

When you use the templates provided by Visual Studio (or built with Visual Studio snippets), the PowerShell script not only stages the artifacts, it also parameterizes the URI for the resources for different deployments. The script then copies the artifacts to a secure container in Azure, creates a SaaS token for that container, and then passes that information on to the template deployment. See [Create a template deployment](#) to learn more about nested templates. When using tasks in VS Team Services, you must select the appropriate tasks for your template deployment and if necessary, pass parameter values from the staging step to the template deployment.

Set up continuous deployment in VS Team Services

To call the PowerShell script in VS Team Services, you need to update your build definition. In brief, the steps are:

1. Edit the build definition.
2. Set up Azure authorization in VS Team Services.
3. Add an Azure PowerShell build step that references the PowerShell script in the Azure Resource Group deployment project.
4. Set the value of the `-ArtifactsStagingDirectory` parameter to work with a project built in VS Team Services.

Detailed walkthrough for Option 1

The following procedures walk you through the steps necessary to configure continuous deployment in VS Team Services using a single task that runs the PowerShell script in your project.

1. Edit your VS Team Services build definition and add an Azure PowerShell build step. Choose the build

definition under the **Build definitions** category and then choose the **Edit** link.

The screenshot shows the Visual Studio Online interface with the 'testProject' selected. The 'BUILD' tab is active. In the 'Explorer' sidebar, the 'Build definitions' section is expanded, showing 'All build definitions' and a list item 'DSC-CI'. The 'Edit' link next to 'DSC-CI' is highlighted with a red box. The main pane displays the 'DSC-CI' build definition with tabs for 'Queued' and 'Completed' (which is selected). Below the tabs are buttons for 'Queue build...', 'Lock', 'Unlock', and 'Name'.

2. Add a new **Azure PowerShell** build step to the build definition and then choose the **Add build step...** button.

The screenshot shows the 'Builds' tab for the 'DSC-CI' build definition. At the top, there are tabs for 'Build', 'Options', 'Repository', 'Variables', and 'Triggers'. Below the tabs are buttons for 'Save', 'Queue build...', and 'Undo'. A prominent red box highlights the 'Add build step...' button. The main area lists a single build step: 'Visual Studio Build' with the command 'Build solution \$/testProject/AzureResourceGro...'. There is a small red arrow pointing from the 'Add build step...' button towards this list item.

3. Choose the **Deploy task** category, select the **Azure PowerShell** task, and then choose its **Add** button.

The screenshot shows the 'ADD TASKS' dialog. On the left, a sidebar has categories: All, Build, Utility, Test, Package, and Deploy, with 'Deploy' selected and highlighted with a red box. The main area lists tasks under the 'Deploy' category. One task, 'Azure PowerShell', is highlighted with a red box. It has a description: 'Run a PowerShell script within an Azure environment'. To the right of each task is an 'Add' button.

4. Choose the **Azure PowerShell** build step and then fill in its values.

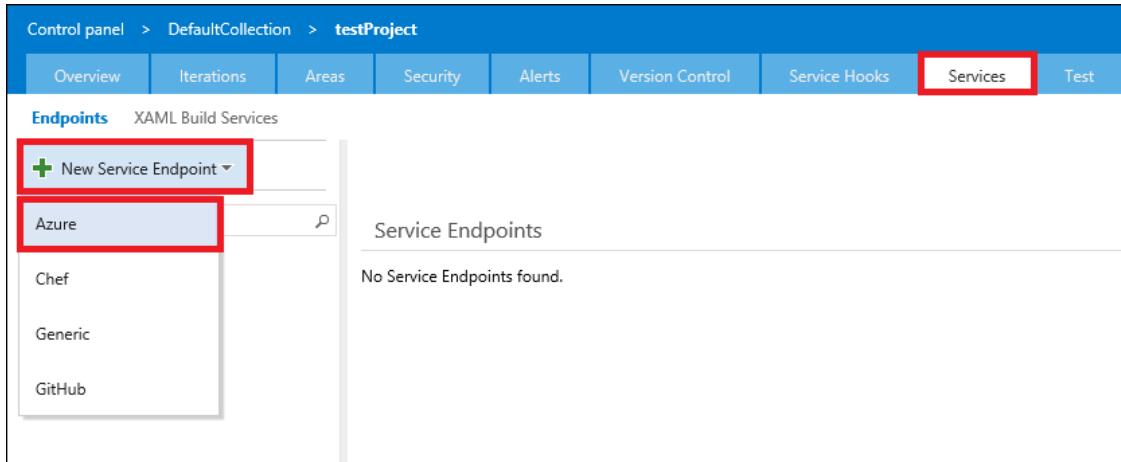
- a. If you already have an Azure service endpoint added to VS Team Services, choose the subscription in the **Azure Subscription** drop-down list box and then skip to the next section.

If you don't have an Azure service endpoint in VS Team Services, you need to add one. This subsection takes you through the process. If your Azure account uses a Microsoft account (such as Hotmail), you must take the following steps to get a Service Principal authentication.

- b. Choose the **Manage** link next to the **Azure Subscription** drop-down list box.



- c. Choose **Azure** in the **New Service Endpoint** drop-down list box.



- d. In the **Add Azure Subscription** dialog box, select the **Service Principal** option.



- e. Add your Azure subscription information to the **Add Azure Subscription** dialog box. You need to provide the following items:

- Subscription Id
- Subscription Name
- Service Principal Id
- Service Principal Key
- Tenant Id

- f. Add a name of your choice to the **Subscription** name box. This value appears later in the **Azure Subscription** drop-down list in VS Team Services.

- g. If you don't know your Azure subscription ID, you can use one of the following commands to retrieve it.

For PowerShell scripts, use:

```
Get-AzureRmSubscription
```

For Azure CLI, use:

```
az account show
```

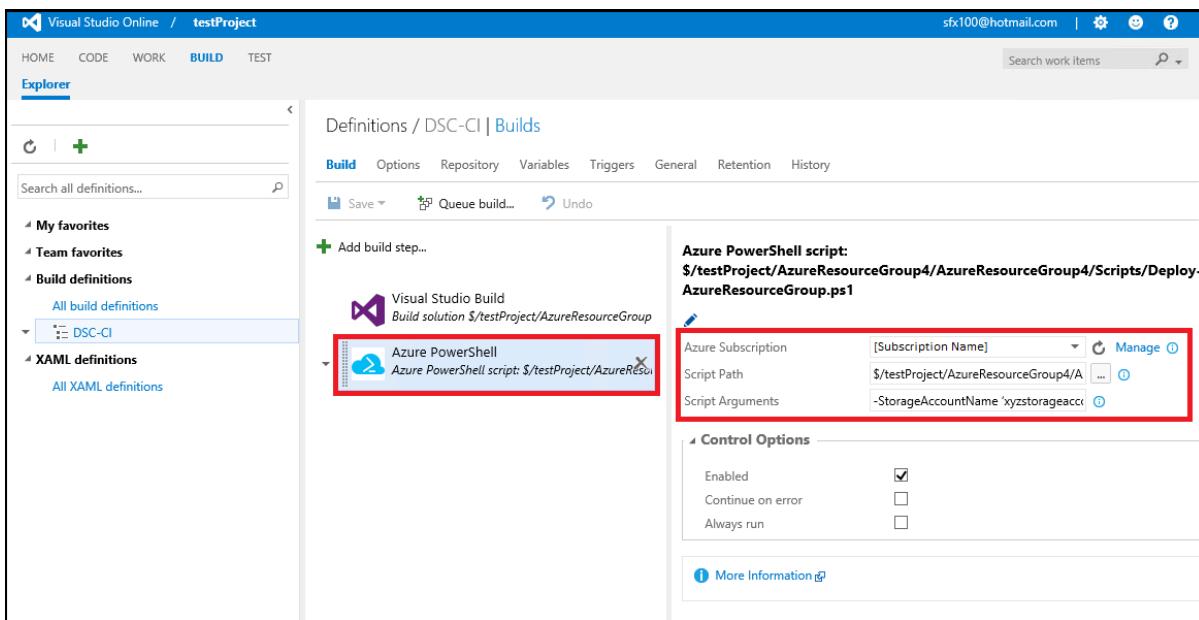
- h. To get a Service Principal ID, Service Principal Key, and Tenant ID, follow the procedure in [Create Active Directory application and service principal using portal](#) or [Authenticating a service principal with Azure Resource Manager](#).

- i. Add the Service Principal ID, Service Principal Key, and Tenant ID values to the **Add Azure Subscription** dialog box and then choose the **OK** button.

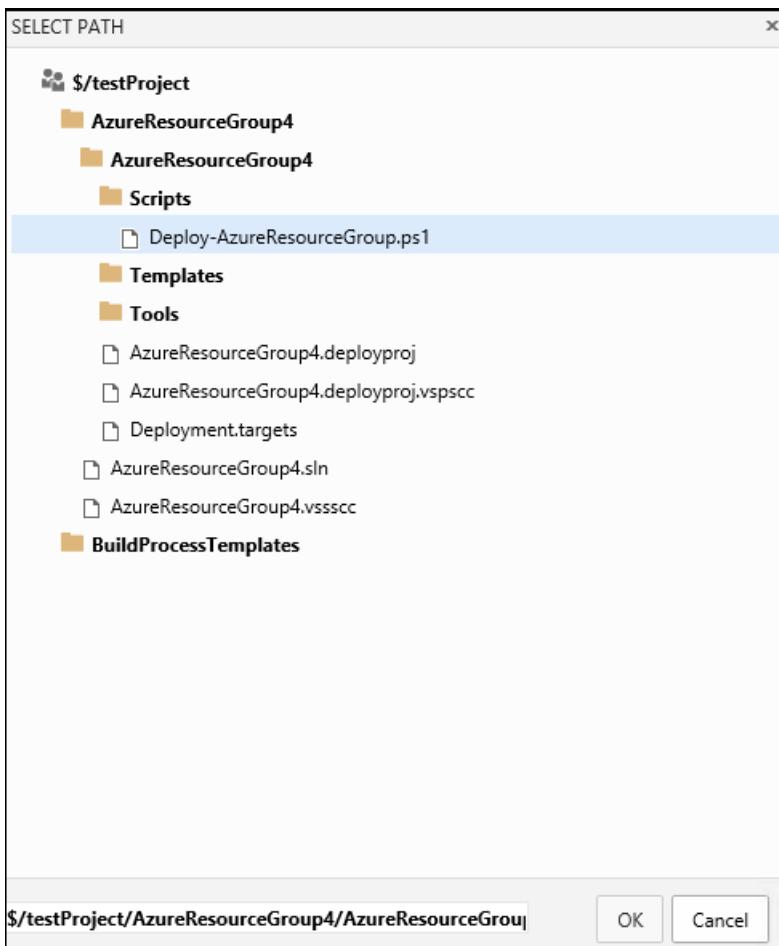
You now have a valid Service Principal to use to run the Azure PowerShell script.

5. Edit the build definition and choose the **Azure PowerShell** build step. Select the subscription in the **Azure**

Subscription drop-down list box. (If the subscription doesn't appear, choose the **Refresh** button next the **Manage** link.)



- Provide a path to the Deploy-AzureResourceGroup.ps1 PowerShell script. To do this, choose the ellipsis (...) button next to the **Script Path** box, navigate to the Deploy-AzureResourceGroup.ps1 PowerShell script in the **Scripts** folder of your project, select it, and then choose the **OK** button.



- After you select the script, update the path to the script so that it's run from the **Build.StagingDirectory** (the same directory that **ArtifactsLocation** is set to). You can do this by adding "\$(Build.StagingDirectory)/" to the beginning of the script path.



8. In the **Script Arguments** box, enter the following parameters (in a single line). When you run the script in Visual Studio, you can see how VS uses the parameters in the **Output** window. You can use this as a starting point for setting the parameter values in your build step.

PARAMETER	DESCRIPTION
-ResourceGroupLocation	The geo-location value where the resource group is located, such as eastus or ' East US '. (Add single quotes if there's a space in the name.) See Azure Regions for more information.
-ResourceGroupName	The name of the resource group used for this deployment.
-UploadArtifacts	This parameter, when present, specifies that artifacts that need to be uploaded to Azure from the local system. You only need to set this switch if your template deployment requires extra artifacts that you want to stage using the PowerShell script (such as configuration scripts or nested templates).
-StorageAccountName	The name of the storage account used to stage artifacts for this deployment. This parameter is only used if you are staging artifacts for deployment. If this parameter is supplied, a new storage account is created if the script has not created one during a previous deployment. If the parameter is specified, the storage account must already exist.
-StorageAccountResourceGroupName	The name of the resource group associated with the storage account. This parameter is required only if you provide a value for the StorageAccountName parameter.
-TemplateFile	The path to the template file in the Azure Resource Group deployment project. To enhance flexibility, use a path for this parameter that is relative to the location of the PowerShell script instead of an absolute path.
-TemplateParametersFile	The path to the parameters file in the Azure Resource Group deployment project. To enhance flexibility, use a path for this parameter that is relative to the location of the PowerShell script instead of an absolute path.
-ArtifactStagingDirectory	This parameter lets the PowerShell script know the folder from where the project's binary files should be copied. This value overrides the default value used by the PowerShell script. For VS Team Services use, set the value to: -ArtifactStagingDirectory \$(Build.StagingDirectory)

Here's a script arguments example (line broken for readability):

```

-ResourceGroupName 'MyGroup' -ResourceGroupLocation 'eastus' -TemplateFile
'..\templates\azuredeploy.json'
-TemplateParametersFile '..\templates\azuredeploy.parameters.json' -UploadArtifacts -StorageAccountName
'mystorageacct'
-StorageAccountResourceGroupName 'Default-Storage-EastUS' -ArtifactStagingDirectory
'$(Build.StagingDirectory)'

```

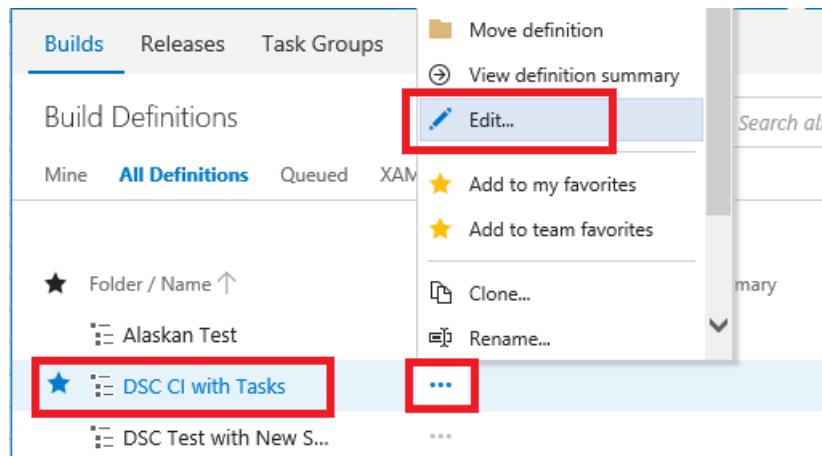
When you're finished, the **Script Arguments** box should resemble the following list:

- After you've added all the required items to the Azure PowerShell build step, choose the **Queue** build button to build the project. The **Build** screen shows the output from the PowerShell script.

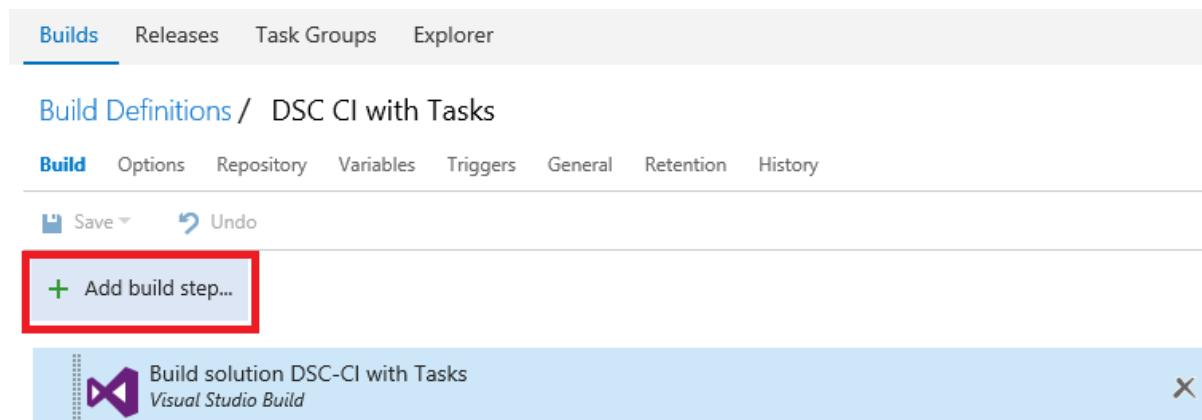
Detailed walkthrough for Option 2

The following procedures walk you through the steps necessary to configure continuous deployment in VS Team Services using the built-in tasks.

- Edit your VS Team Services build definition to add two new build steps. Choose the build definition under the **Build definitions** category and then choose the **Edit** link.



- Add the new build steps to the build definition using the **Add build step...** button.



- Choose the **Deploy** task category, select the **Azure File Copy** task, and then choose its **Add** button.

Task catalog

All Build Utility Test Package Deploy

Azure File Copy

Azure PowerShell

Azure Resource Group Deployment

Azure SQL Database Deployment

Chef

Don't see what you need? Check out our Marketplace.

Add Add Add Add Add Add Add Add

Close

The screenshot shows the 'Task catalog' window with the 'Deploy' category selected (highlighted with a red box). The 'Azure File Copy' task is also highlighted with a red box. Other tasks listed include Azure PowerShell, Azure Resource Group Deployment, Azure SQL Database Deployment, and Chef. Each task has an 'Add' button to its right. A note at the bottom suggests checking the Marketplace if no suitable task is found.

4. Choose the **Azure Resource Group Deployment** task, then choose its **Add** button and then **Close** the **Task Catalog**.

The screenshot shows the 'Task catalog' window in VS Team Services. On the left, there's a sidebar with categories: All, Build, Utility, Test, Package, Deploy (which is highlighted with a red box), and a link to the Marketplace. The main area lists various tasks with their descriptions and 'Add' buttons. One task, 'Azure Resource Group Deployment', is also highlighted with a red box. At the bottom right of the catalog window is a 'Close' button.

Task	Description	Add Button
Azure App Service Swap	Directs destination Azure App Service slot's traffic to the source slot by performing swap	Add
Azure CLI Preview	Run a Shell or Batch script with Azure CLI commands against an azure subscription	Add
Azure Cloud Service Deployment	Deploy an Azure Cloud Service	Add
Azure File Copy	Copy files to Azure blob or VM(s)	Add
Azure PowerShell	Run a PowerShell script within an Azure environment	Add
Azure Resource Group Deployment	Deploy, start, stop, delete Azure Resource Groups	Add
Azure SQL Database Deployment	Deploy Azure SQL DB using DACPAC or run scripts using SQLCMD	Add
Chef	Deploy to Chef environments by editing environment attributes	Add

5. Choose the **Azure File Copy** task and fill in its values.

If you already have an Azure service endpoint added to VS Team Services, choose the subscription in the **Azure Subscription** drop-down list box. If you do not have a subscription, see [Option 1](#) for instructions on setting one up in VS Team Services.

- Source - enter **\$(Build.StagingDirectory)**
- Azure Connection Type - select **Azure Resource Manager**
- Azure RM Subscription - select the subscription for the storage account you want to use in the **Azure Subscription** drop-down list box. If the subscription doesn't appear, choose the **Refresh** button next the **Manage** link.
- Destination Type - select **Azure Blob**
- RM Storage Account - select the storage account you would like to use for staging artifacts
- Container Name - enter the name of the container you would like to use for staging; it can be any valid container name, but use one dedicated to this build definition

For the output values:

- Storage Container URI - enter **artifactsLocation**
- Storage Container SAS Token - enter **artifactsLocationSasToken**

Build Definitions / DSC CI with Tasks

not built Summary Queue new build S

Build Options Repository Variables Triggers General Retention History

Save Undo

Add build step...

Build solution DSC-CI with Visual Studio Build

AzureBlob File Copy

Azure Deployment Azure Resource Group Deployr

AzureBlob File Copy

Source

`$(Build.StagingDirectory)`

Azure Connection Type

Azure Resource Manager

Azure RM Subscription

Azure Build Principal

Destination Type

Azure Blob

RM Storage Account

stageec0f79d316f04892816

Container Name

nestedsamplevsts

Blob Prefix

Additional Arguments

Output

Storage Container URI

artifactsLocation

Storage Container SAS Token

artifactsLocationSasToken

Control Options

6. Choose the **Azure Resource Group Deployment** build step and then fill in its values.

- Azure Connection Type - select **Azure Resource Manager**
- Azure RM Subscription - select the subscription for deployment in the **Azure Subscription** drop-down list box. This will usually be the same subscription used in the previous step
- Action - select **Create or Update Resource Group**
- Resource Group - select a resource group or enter the name of a new resource group for the deployment
- Location - select the location for the resource group
- Template - enter the path and name of the template to be deployed prepending **\$(Build.StagingDirectory)**, for example: **\$(Build.StagingDirectory)/DSC-CI/azuredeploy.json**)
- Template Parameters - enter the path and name of the parameters to be used, prepending **\$(Build.StagingDirectory)**, for example: **\$(Build.StagingDirectory)/DSC-CI/azuredeploy.parameters.json**)
- Override Template Parameters - enter or copy and paste the following code:

```
_artifactsLocation $(artifactsLocation) _artifactsLocationSasToken (ConvertTo-SecureString -  
String "$(artifactsLocationSasToken)" -AsPlainText -Force)
```

Save Undo

Add build step...

Build solution DSC-CI with Visual Studio Build

 AzureBlob File Copy
Azure File Copy Azure Deployment
Azure Resource Group Deploy**Azure Deployment**

Azure Connection Type	Azure Resource Manager
Azure RM Subscription	Azure Build Principal
Action	Create Or Update Resource Group
Resource Group	DSC-CI
Location	Central US
Template	\$(Build.StagingDirectory)/DSC-CI/azuredeploy.json
Template Parameters	\$(Build.StagingDirectory)/DSC-CI/azuredeploy.parameters.json
Override Template Parameters	-_artifactsLocation \$(artifactsLocation) -_artifactsLocationSasToken (ConvertTo-SecureString)
Deployment Mode	Incremental
Enable Deployment Prerequisites	<input type="checkbox"/>

7. After you've added all the required items, save the build definition and choose **Queue new build** at the top.

Next steps

Read [Azure Resource Manager overview](#) to learn more about Azure Resource Manager and Azure resource groups.

Use Azure Key Vault to pass secure parameter value during deployment

7/10/2018 • 5 minutes to read • [Edit Online](#)

When you need to pass a secure value (like a password) as a parameter during deployment, you can retrieve the value from an [Azure Key Vault](#). 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 are deploying to.

Enable access to the secret

There are two important conditions that must exist for accessing a key vault during template deployment:

1. The key vault property `enabledForTemplateDeployment` must be `true`.
2. The user deploying the template must have access to the secret. The user must have the `Microsoft.KeyVault/vaults/deploy/action` permission for the key vault. The [Owner](#) and [Contributor](#) roles both grant this access.

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](#).

Deploy a key vault and secret

To create a key vault and secret, use either Azure CLI or PowerShell. Notice that the key vault is enabled for template deployment.

For Azure CLI, use:

```
vaultname={your-unique-vault-name}
password={password-value}

az group create --name examplegroup --location 'South Central US'
az keyvault create \
--name $vaultname \
--resource-group examplegroup \
--location 'South Central US' \
--enabled-for-template-deployment true
az keyvault secret set --vault-name $vaultname --name examplesecret --value $password
```

For PowerShell, use:

```

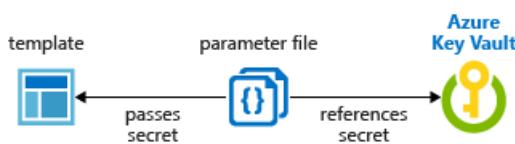
$vaultname = "{your-unique-vault-name}"
$password = "{password-value}"

New-AzureRmResourceGroup -Name examplegroup -Location "South Central US"
New-AzureRmKeyVault ` 
    -VaultName $vaultname ` 
    -ResourceGroupName examplegroup ` 
    -Location "South Central US" ` 
    -EnabledForTemplateDeployment
$secretvalue = ConvertTo-SecureString $password -AsPlainText -Force
Set-AzureKeyVaultSecret -VaultName $vaultname -Name "examplesecret" -SecretValue $secretvalue

```

Reference a secret with static ID

The template that receives a key vault secret is like any other template. That's because **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.



For example, the [following template](#) deploys a SQL database that includes an administrator password. The `password` parameter is set to a secure string. But, the template does not 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. Copy this file locally, and set the subscription ID, vault name, and SQL server name.

```
{  
    "$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/examplegroup/providers/Microsoft.KeyVault/vaults/<vault-name>"  
                },  
                "secretName": "examplesecret"  
            }  
        },  
        "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": "examplesecret",  
"secretVersion": "cd91b2b7e10e492ebb870a6ee0591b68"
```

Now, deploy the template and pass in the parameter file. You can use the example template from GitHub, but you must use a local parameter file with the values set to your environment.

For Azure CLI, use:

```
az group create --name datagroup --location "South Central US"  
az group deployment create \  
    --name exampledeployment \  
    --resource-group datagroup \  
    --template-uri https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-  
    manager/keyvaultparameter/sqlserver.json \  
    --parameters @sqlserver.parameters.json
```

For PowerShell, use:

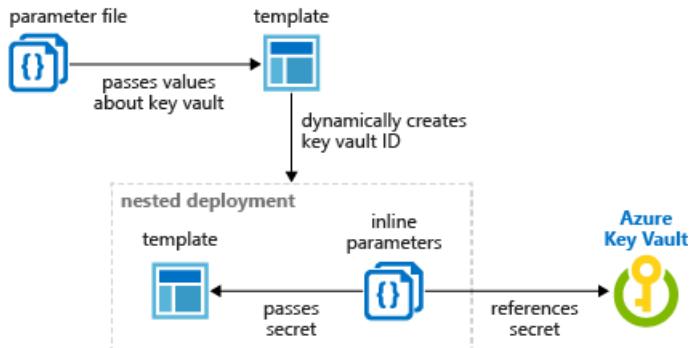
```
New-AzureRmResourceGroup -Name datagroup -Location "South Central US"  
New-AzureRmResourceGroupDeployment `  
    -Name exampledeployment `  
    -ResourceGroupName datagroup `  
    -TemplateUri https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-  
    manager/keyvaultparameter/sqlserver.json `  
    -TemplateParameterFile sqlserver.parameters.json
```

Reference a secret with dynamic ID

The previous section showed how to pass a static resource ID for the key vault secret. However, in some scenarios, you need to reference a key vault secret that varies based on the current deployment. In that case, you can't hard-code the resource ID in the parameters file. Unfortunately, you can't dynamically generate the resource ID in the

parameters file because template expressions aren't allowed in the parameters file.

To dynamically generate the resource ID for a key vault secret, you must move the resource that needs the secret into a linked template. 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.



Your linked template must be available through an external URI. Typically, you add your template to a storage account, and access it through the URI like

```
https://<storage-name>.blob.core.windows.net/templatecontainer/sqlserver.json .
```

The [following template](#) dynamically creates the key vault ID and passes it as a parameter. It links to an [example template](#) in GitHub.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "vaultName": {
            "type": "string"
        },
        "vaultResourceGroup": {
            "type": "string"
        },
        "secretName": {
            "type": "string"
        },
        "adminLogin": {
            "type": "string"
        },
        "sqlServerName": {
            "type": "string"
        }
    },
    "resources": [
    {
        "apiVersion": "2015-01-01",
        "name": "nestedTemplate",
        "type": "Microsoft.Resources/deployments",
        "properties": {
            "mode": "incremental",
            "templateLink": {
                "uri": "https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-manager/keyvaultparameter/sqlserver.json",
                "contentVersion": "1.0.0.0"
            },
            "parameters": {
                "adminPassword": {
                    "reference": {
                        "keyVault": {
                            "id": "[resourceId(subscription().subscriptionId, parameters('vaultResourceGroup'), 'Microsoft.KeyVault/vaults', parameters('vaultName'))]"
                        },
                        "secretName": "[parameters('secretName')]"
                    }
                },
                "adminLogin": { "value": "[parameters('adminLogin')]" },
                "sqlServerName": { "value": "[parameters('sqlServerName')]" }
            }
        }
    }],
    "outputs": {}
}
```

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 datagroup --location "South Central US"
az group deployment create \
    --name exampledeployment \
    --resource-group datagroup \
    --template-uri https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-manager/keyvaultparameter/sqlserver-dynamic-id.json \
    --parameters vaultName=<your-vault> vaultResourceGroup=examplegroup secretName=examplesecret
    adminLogin=exampleadmin sqlServerName=<server-name>
```

For PowerShell, use:

```
New-AzureRmResourceGroup -Name datagroup -Location "South Central US"
New-AzureRmResourceGroupDeployment ` 
    -Name exampledeployment ` 
    -ResourceGroupName datagroup ` 
    -TemplateUri https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/keyvaultparameter/sqlserver-dynamic-id.json ` 
    -vaultName <your-vault> -vaultResourceGroup examplegroup -secretName examplesecret -adminLogin exampleadmin
    -sqlServerName <server-name>
```

Next steps

- For general information about key vaults, see [Get started with Azure Key Vault](#).
- For complete examples of referencing key secrets, see [Key Vault examples](#).

Manage resources with Azure PowerShell

7/23/2018 • 13 minutes to read • [Edit Online](#)

When deploying resources to Azure, you have tremendous flexibility when deciding what types of resources to deploy, where they are located, and how to set them up. However, that flexibility may open more options than you would like to allow in your organization. As you consider deploying resources to Azure, you might be wondering:

- How do I meet legal requirements for data sovereignty in certain countries?
- How do I control costs?
- How do I ensure that someone does not inadvertently change a critical system?
- How do I track resource costs and bill it accurately?

This article addresses those questions. Specifically, you:

- Assign users to roles and assign the roles to a scope so users have permission to perform expected actions but not more actions.
- Apply policies that prescribe conventions for resources in your subscription.
- Lock resources that are critical to your system.
- Tag resources so you can track them by values that make sense to your organization.

This article focuses on the tasks you take to implement governance. For a broader discussion of the concepts, see [Governance in Azure](#).

Launch Azure Cloud Shell

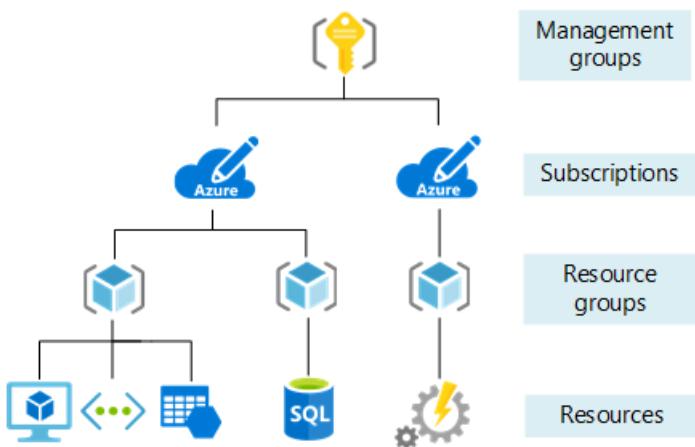
The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account. Just click the **Copy** to copy the code, paste it into the Cloud Shell, and then press enter to run it. There are a few ways to launch the Cloud Shell:

Click Try It in the upper right corner of a code block.	
Open Cloud Shell in your browser.	
Click the Cloud Shell button on the menu in the upper right of the Azure portal.	

If you choose to install and use the PowerShell locally, see [Install Azure PowerShell module](#). If you are running PowerShell locally, you also need to run `Connect-AzureRmAccount` to create a connection with Azure.

Understand scope

Before creating any items, let's review the concept of scope. Azure provides four levels of management: management groups, subscription, resource group, and resource. [Management groups](#) are in a preview release. 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. When you apply a setting to the subscription, that setting is applied to all resource groups and resources in your subscription. When you apply a setting on the resource group, that setting is applied to the resource group and all its resources. However, another resource group does not have that setting.

Usually, it makes sense to apply critical settings at higher levels and project-specific requirements at lower levels. For example, you might want to make sure all resources for your organization are deployed to certain regions. To accomplish this requirement, apply a policy to the subscription that specifies the allowed locations. As other users in your organization add new resource groups and resources, the allowed locations are automatically enforced.

In this article, you apply all management settings to a resource group so you can easily remove those settings when done.

Let's create the resource group.

```
Set-AzureRmContext -Subscription <subscription-name>
New-AzureRmResourceGroup -Name myResourceGroup -Location EastUS
```

Currently, the resource group is empty.

Role-based access control

You want to make sure users in your organization have the right level of access to these resources. You don't want to grant unlimited access to users, but you also need to make sure they can do their work. Role-based access control (RBAC) enables you to manage which users have permission to complete specific actions at a scope. A role defines a set of permitted actions. You assign the role to a scope, and specify which users belong to that role for the scope.

When planning your access control strategy, grant users the least privilege to get their work done. The following image shows a suggested pattern for assigning RBAC.

Best Practice: Manage to Least Privilege

	Reader	Resource-specific or Custom role	Contributor	Owner
Subscription				
Resource Group	Observers		Users managing resources	Admin
Resource			Automated processes	

There are three roles that apply to all resources - Owner, Contributor, and Reader. Any accounts assigned to the Owner role should be tightly controlled and rarely used. Users that only need to observe the state of solutions should be granted the Reader role.

Most users are granted [resource-specific roles](#) or [custom roles](#) at either the subscription or resource group level. These roles tightly define the permitted actions. By assigning users to these roles, you grant the required access for users without permitting too much control. You can assign an account to more than one role, and that user gets the combined permissions of the roles. Granting access at the resource level is often too restrictive for users, but may work for an automated process designed for specific task.

Who can assign roles

To create and remove role assignments, users must have `Microsoft.Authorization/roleAssignments/*` access. This access is granted through the Owner or User Access Administrator roles.

Assign a role

In this article, you deploy a virtual machine and its related virtual network. For managing virtual machine solutions, there are three resource-specific roles that provide commonly needed access:

- [Virtual Machine Contributor](#)
- [Network Contributor](#)
- [Storage Account Contributor](#)

Instead of assigning roles to individual users, it's often easier to [create an Azure Active Directory group](#) for users who need to take similar actions. Then, assign that group to the appropriate role. To simplify this article, you create an Azure Active Directory group without members. You can still assign this group to a role for a scope.

The following example creates a group and assigns it to the Virtual Machine Contributor role for the resource group. To run the `New-AzureAdGroup` command, you must either use the [Azure Cloud Shell](#) or [download the Azure AD PowerShell module](#).

```
$adgroup = New-AzureADGroup -DisplayName VMdemоАContributors `  
-MailNickname vmdemogroup `  
-MailEnabled $false `  
-SecurityEnabled $true  
New-AzureRmRoleAssignment -ObjectId $adgroup.ObjectId `  
-ResourceGroupName myResourceGroup `  
-RoleDefinitionName "Virtual Machine Contributor"
```

Typically, you repeat the process for **Network Contributor** and **Storage Account Contributor** to make sure users are assigned to manage the deployed resources. In this article, you can skip those steps.

Azure Policy

[Azure Policy](#) helps you make sure all resources in subscription meet corporate standards. Your subscription already has several policy definitions. To see the available policy definitions, use:

```
(Get-AzureRmPolicyDefinition).Properties | Format-Table displayName, policyType
```

You see the existing policy definitions. The policy type is either **BuiltIn** or **Custom**. Look through the definitions for ones that describe a condition you want assign. In this article, you assign policies that:

- limit the locations for all resources
- limit the SKUs for virtual machines
- audit virtual machines that do not use managed disks

```
$locations = "eastus", "eastus2"
$skus = "Standard_DS1_v2", "Standard_E2s_v2"

$rg = Get-AzureRmResourceGroup -Name myResourceGroup

$locationDefinition = Get-AzureRmPolicyDefinition | where-object {$_.properties.displayname -eq "Allowed locations"}
$skuDefinition = Get-AzureRmPolicyDefinition | where-object {$_.properties.displayname -eq "Allowed virtual machine SKUs"}
$auditDefinition = Get-AzureRmPolicyDefinition | where-object {$_.properties.displayname -eq "Audit VMs that do not use managed disks"}

New-AzureRMPolicyAssignment -Name "Set permitted locations" ` 
    -Scope $rg.ResourceId ` 
    -PolicyDefinition $locationDefinition ` 
    -listOfAllowedLocations $locations
New-AzureRMPolicyAssignment -Name "Set permitted VM SKUs" ` 
    -Scope $rg.ResourceId ` 
    -PolicyDefinition $skuDefinition ` 
    -listOfAllowedSKUs $skus
New-AzureRMPolicyAssignment -Name "Audit unmanaged disks" ` 
    -Scope $rg.ResourceId ` 
    -PolicyDefinition $auditDefinition
```

Deploy the virtual machine

You have assigned roles and policies, so you're ready to deploy your solution. The default size is Standard_DS1_v2, which is one of your allowed SKUs. When running this step, you are prompted for credentials. The values that you enter are configured as the user name and password for the virtual machine.

```
New-AzureRmVm -ResourceGroupName "myResourceGroup" ` 
    -Name "myVM" ` 
    -Location "East US" ` 
    -VirtualNetworkName "myVnet" ` 
    -SubnetName "mySubnet" ` 
    -SecurityGroupName "myNetworkSecurityGroup" ` 
    -PublicIpAddressName "myPublicIpAddress" ` 
    -OpenPorts 80,3389
```

After your deployment finishes, you can apply more management settings to the solution.

Lock resources

Resource locks prevent users in your organization from accidentally deleting or modifying critical resources. Unlike role-based access control, resource locks apply a restriction across all users and roles.

You can set the lock level to **CanNotDelete** or **ReadOnly**. In the portal, the locks levels are displayed as **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.

TIP

Be careful when applying a **ReadOnly** lock. Some operations that seem like read operations actually require additional actions. For example, 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.

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.

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 process 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 allowed because those operations are not sent to <https://management.azure.com>.

Who can create or delete locks in your organization

To create or delete management locks, you must have access to [Microsoft.Authorization/locks/*](#) actions. Of the built-in roles, only **Owner** and **User Access Administrator** are granted those actions.

Lock a resource

To lock the virtual machine and network security group, use:

```
New-AzureRmResourceLock -LockLevel CanNotDelete `  
-LockName LockVM `  
-ResourceName myVM `  
-ResourceType Microsoft.Compute/virtualMachines `  
-ResourceGroupName myResourceGroup  
New-AzureRmResourceLock -LockLevel CanNotDelete `  
-LockName LockNSG `  
-ResourceName myNetworkSecurityGroup `  
-ResourceType Microsoft.Network/networkSecurityGroups `  
-ResourceGroupName myResourceGroup
```

The virtual machine can only be deleted if you specifically remove the lock. That step is shown in [Clean up resources](#).

Tag resources

You apply tags to your Azure resources to logically organize them by categories. Each tag consists of a name and a value. 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.

The following limitations apply to tags:

- Each resource or resource group can have a maximum of 15 tag name/value pairs. This limitation applies only to tags directly applied to the resource group or resource. A resource group can contain many resources that each have 15 tag name/value pairs. If you have more than 15 values that you need to associate with a resource, use a JSON string for the tag value. The JSON string can contain many values that are applied to a single tag name. This article shows an example of assigning a JSON string to the tag.
- 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.
- 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: <, >, %, &, \, ?, /

Tag resources

To add two tags to a resource group, use the [Set-AzureRmResourceGroup](#) command:

```
Set-AzureRmResourceGroup -Name myResourceGroup -Tag @{ Dept="IT"; Environment="Test" }
```

Let's suppose you want to add a third tag. Every time you apply tags to a resource or a resource group, you overwrite the existing tags on that resource or resource group. To add a new tag without losing the existing tags, you must retrieve the existing tags, add a new tag, and reapply the collection of tags:

```
# Get existing tags and add a new tag
$tags = (Get-AzureRmResourceGroup -Name myResourceGroup).Tags
$tags.Add("Project", "Documentation")

# Reapply the updated set of tags
Set-AzureRmResourceGroup -Tag $tags -Name myResourceGroup
```

Resources don't inherit tags from the resource group. Currently, your resource group has three tags but the resources do not have any tags. To apply all tags from a resource group to its resources, and retain existing tags on resources that are not duplicates, use the following script:

```

# Get the resource group
$group = Get-AzureRmResourceGroup myResourceGroup

if ($group.Tags -ne $null) {
    # Get the resources in the resource group
    $resources = Get-AzureRmResource -ResourceGroupName $group.ResourceGroupName

    # Loop through each resource
    foreach ($r in $resources)
    {
        # Get the tags for this resource
        $resourcetags = (Get-AzureRmResource -ResourceId $r.ResourceId).Tags

        # If the resource has existing tags, add new ones
        if ($resourcetags)
        {
            foreach ($key in $group.Tags.Keys)
            {
                if (-not($resourcetags.ContainsKey($key)))
                {
                    $resourcetags.Add($key, $group.Tags[$key])
                }
            }

            # Reapply the updated tags to the resource
            Set-AzureRmResource -Tag $resourcetags -ResourceId $r.ResourceId -Force
        }
        else
        {
            Set-AzureRmResource -Tag $group.Tags -ResourceId $r.ResourceId -Force
        }
    }
}

```

Alternatively, you can apply tags from the resource group to the resources without keeping the existing tags:

```

# Get the resource group
$g = Get-AzureRmResourceGroup -Name myResourceGroup

# Find all the resources in the resource group, and for each resource apply the tags from the resource group
Get-AzureRmResource -ResourceGroupName $g.ResourceGroupName | ForEach-Object {Set-AzureRmResource -ResourceId $_.ResourceId -Tag $g.Tags -Force }

```

To combine several values in a single tag, use a JSON string.

```
Set-AzureRmResourceGroup -Name myResourceGroup -Tag @{ CostCenter="{"Dept`":`"IT`",`"Environment`":`"Test`"}"
}
```

To remove all tags, you pass an empty hash table.

```
Set-AzureRmResourceGroup -Name myResourceGroup -Tag @{} 
```

To apply tags to a virtual machine, use:

```
$r = Get-AzureRmResource -ResourceName myVM ` 
-ResourceGroupName myResourceGroup ` 
-ResourceType Microsoft.Compute/virtualMachines
Set-AzureRmResource -Tag @{ Dept="IT"; Environment="Test"; Project="Documentation" } -ResourceId $r.ResourceId -Force
```

Find resources by tag

To find resources with a tag name and value, use:

```
(Find-AzureRmResource -TagName Environment -TagValue Test).Name
```

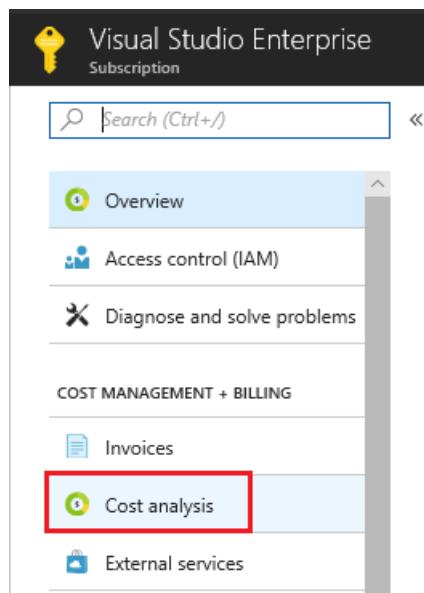
You can use the returned values for management tasks like stopping all virtual machines with a tag value.

```
Find-AzureRmResource -TagName Environment -TagValue Test | Where-Object {$_ . ResourceType -eq "Microsoft.Compute/virtualMachines"} | Stop-AzureRmVM
```

View costs by tag values

After applying tags to resources, you can view costs for resources with those tags. It takes a while for cost analysis to show the latest usage, so you may not see the costs yet. When the costs are available, you can view costs for resources across resource groups in your subscription. Users must have [subscription level access to billing information](#) to see the costs.

To view costs by tag in the portal, select your subscription and select **Cost Analysis**.



Then, filter by the tag value, and select **Apply**.

A screenshot of the 'Costs by service' blade in the Azure portal. It shows a summary of costs: 'Total cost 0.24 USD'. On the left, there are filters for 'Subscription' (Visual Studio Enterprise), 'Resource type' (3 selected), and 'Resource group' (2 selected). Under 'Timespan', 'Current period' is selected. In the 'Tag' section, 'Environment: Test' is selected. A tooltip explains: 'There is a delay between the time when resources are tagged and when they reach the billing system. Due to this, costs shown here may be delayed. Amounts shown here do not include taxes.' At the bottom, there are 'Apply' and 'Download' buttons.

You can also use the [Azure Billing APIs](#) to programmatically view costs.

Clean up resources

The locked network security group can't be deleted until the lock is removed. To remove the lock, use:

```
Remove-AzureRmResourceLock -LockName LockVM `  
-ResourceName myVM `  
-ResourceType Microsoft.Compute/virtualMachines `  
-ResourceGroupName myResourceGroup  
Remove-AzureRmResourceLock -LockName LockNSG `  
-ResourceName myNetworkSecurityGroup `  
-ResourceType Microsoft.Network/networkSecurityGroups `  
-ResourceGroupName myResourceGroup
```

When no longer needed, you can use the [Remove-AzureRmResourceGroup](#) command to remove the resource group, VM, and all related resources.

```
Remove-AzureRmResourceGroup -Name myResourceGroup
```

Next steps

- To learn about monitoring your virtual machines, see [Monitor and update a Windows Virtual Machine with Azure PowerShell](#).
- To learn about using Azure Security Center to implement recommended security practices, [Monitor virtual machine security by using Azure Security Center](#).
- You can move existing resources to a new resource group. For examples, see [Move Resources to New Resource Group or Subscription](#).
- For guidance on how enterprises can use Resource Manager to effectively manage subscriptions, see [Azure enterprise scaffold - prescriptive subscription governance](#).

Use the Azure CLI to manage Azure resources and resource groups

7/25/2018 • 6 minutes to read • [Edit Online](#)

In this article, you learn how to manage your solutions with Azure CLI and Azure Resource Manager. If you are not familiar with Resource Manager, see [Resource Manager Overview](#). This article focuses on management tasks. You will:

1. Create a resource group
2. Add a resource to the resource group
3. Add a tag to the resource
4. Query resources based on names or tag values
5. Apply and remove a lock on the resource
6. Delete a resource group

This article does not show how to deploy a Resource Manager template to your subscription. For that information, see [Deploy resources with Resource Manager templates and Azure CLI](#).

Open Azure Cloud Shell

Azure Cloud Shell is a free, interactive shell that you can use to run the steps in this article. Common Azure tools are preinstalled and configured in Cloud Shell for you to use with your account. Just select the **Copy** button to copy the code, paste it in Cloud Shell, and then press Enter to run it. There are a few ways to open Cloud Shell:

Select Try It in the upper-right corner of a code block.	
Open Cloud Shell in your browser.	
Select the Cloud Shell button on the menu in the upper-right corner of the Azure portal .	

To install and use the CLI locally, see [Install Azure CLI](#).

Set subscription

If you have more than one subscription, you can switch to a different subscription. First, let's see all the subscriptions for your account.

```
az account list
```

It returns a list of your enabled and disabled subscriptions.

```
[  
 {  
   "cloudName": "AzureCloud",  
   "id": "<guid>",  
   "isDefault": true,  
   "name": "Example Subscription One",  
   "registeredProviders": [],  
   "state": "Enabled",  
   "tenantId": "<guid>",  
   "user": {  
     "name": "example@contoso.org",  
     "type": "user"  
   }  
 },  
 ...  
 ]
```

Notice that one subscription is marked as the default. This subscription is your current context for operations. To switch to a different subscription, provide the subscription name with the **az account set** command.

```
az account set -s "Example Subscription Two"
```

To show the current subscription context, use **az account show** without a parameter:

```
az account show
```

Create a resource group

Before deploying any resources to your subscription, you must create a resource group that will contain the resources.

To create a resource group, use the **az group create** command. The command uses the **name** parameter to specify a name for the resource group and the **location** parameter to specify its location.

```
az group create --name TestRG1 --location "South Central US"
```

The output is in the following format:

```
{  
   "id": "/subscriptions/<subscription-id>/resourceGroups/TestRG1",  
   "location": "southcentralus",  
   "managedBy": null,  
   "name": "TestRG1",  
   "properties": {  
     "provisioningState": "Succeeded"  
   },  
   "tags": null  
}
```

If you need to retrieve the resource group later, use the following command:

```
az group show --name TestRG1
```

To get all the resource groups in your subscription, use:

```
az group list
```

Add resources to a resource group

To add a resource to the resource group, you can use the **az resource create** command or a command that is specific to the type of resource you are creating (like **az storage account create**). You might find it easier to use a command that is specific to a resource type because it includes parameters for the properties that are needed for the new resource. To use **az resource create**, you must know all the properties to set without being prompted for them.

However, adding a resource through script might cause future confusion because the new resource does not exist in a Resource Manager template. Templates enable you to reliably and repeatedly deploy your solution.

The following command creates a storage account. Instead of using the name shown in the example, provide a unique name for the storage account. The name must be between 3 and 24 characters in length, and use only numbers and lower-case letters. If you use the name shown in the example, you receive an error because that name is already in use.

```
az storage account create -n myuniquestorage -g TestRG1 -l westus --sku Standard_LRS
```

If you need to retrieve this resource later, use the following command:

```
az storage account show --name myuniquestorage --resource-group TestRG1
```

Add a tag

Tags enable you to organize your resources according to different properties. For example, you may have several resources in different resource groups that belong to the same department. You can apply a department tag and value to those resources to mark them as belonging to the same category. Or, you can mark whether a resource is used in a production or test environment. In this article, you apply tags to only one resource, but in your environment it most likely makes sense to apply tags to all your resources.

The following command applies two tags to your storage account:

```
az resource tag --tags Dept=IT Environment=Test -g TestRG1 -n myuniquestorage --resource-type "Microsoft.Storage/storageAccounts"
```

Tags are updated as a single object. To add a tag to a resource that already includes tags, first retrieve the existing tags. Add the new tag to the object that contains the existing tags, and reapply all the tags to the resource.

```
jsonrtag=$(az resource show -g TestRG1 -n myuniquestorage --resource-type "Microsoft.Storage/storageAccounts" -q tags)
rt=$(echo $jsonrtag | tr -d '"'{}',' | sed 's/: /=g')
az resource tag --tags $rt Project=Redesign -g TestRG1 -n myuniquestorage --resource-type "Microsoft.Storage/storageAccounts"
```

Search for resources

Use the **az resource list** command to retrieve resources for different search conditions.

- To get a resource by name, provide the **name** parameter:

```
az resource list -n myuniquestorage
```

- To get all the resources in a resource group, provide the **resource-group** parameter:

```
az resource list --resource-group TestRG1
```

- To get all the resources with a tag name and value, provide the **tag** parameter:

```
az resource list --tag Dept=IT
```

- To get all the resources with a particular resource type, provide the **resource-type** parameter:

```
az resource list --resource-type "Microsoft.Storage/storageAccounts"
```

Get resource ID

Many commands take a resource ID as a parameter. To get the ID for a resource and store in a variable, use:

```
webappID=$(az resource show -g exampleGroup -n exampleSite --resource-type "Microsoft.Web/sites" --query id --output tsv)
```

Lock a resource

When you need to make sure a critical resource is not accidentally deleted or modified, apply a lock to the resource. You can specify either a **CanNotDelete** or **ReadOnly**.

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.

To apply a lock, use the following command:

```
az lock create --lock-type CanNotDelete --resource-name myuniquestorage --resource-group TestRG1 --resource-type Microsoft.Storage/storageAccounts --name storagelock
```

The locked resource in the preceding example cannot be deleted until the lock is removed. To remove a lock, use:

```
az lock delete --name storagelock --resource-group TestRG1 --resource-type Microsoft.Storage/storageAccounts --resource-name myuniquestorage
```

For more information about setting locks, see [Lock resources with Azure Resource Manager](#).

Remove resources or resource group

You can remove a resource or resource group. When you remove a resource group, you also remove all the resources within that resource group.

- To delete a resource from the resource group, use the delete command for the resource type you are deleting. The command deletes the resource, but does not delete the resource group.

```
az storage account delete -n myuniquestorage -g TestRG1
```

- To delete a resource group and all its resources, use the **az group delete** command.

```
az group delete -n TestRG1
```

For both commands, you are asked to confirm that you wish to remove the resource or resource group.

Next steps

- To learn about creating Resource Manager templates, see [Authoring Azure Resource Manager Templates](#).
- To learn about deploying templates, see [Deploy an application with Azure Resource Manager Template](#).
- You can move existing resources to a new resource group. For examples, see [Move Resources to New Resource Group or Subscription](#).
- For guidance on how enterprises can use Resource Manager to effectively manage subscriptions, see [Azure enterprise scaffold - prescriptive subscription governance](#).

Manage Azure resources through portal

5/21/2018 • 5 minutes to read • [Edit Online](#)

This article shows how to use the [Azure portal](#) with [Azure Resource Manager](#) to manage your Azure resources. To learn about deploying resources through the portal, see [Deploy resources with Resource Manager templates and Azure portal](#).

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](#).

Manage resource groups

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.

1. To see all the resource groups in your subscription, select **Resource groups**.

The screenshot shows the Microsoft Azure portal interface. On the left, there's a sidebar with various options like 'Create a resource', 'All services', 'FAVORITES', 'Dashboard', 'All resources', and 'Resource groups'. The 'Resource groups' option is highlighted with a red box. The main content area is titled 'Resource groups' and shows a list of subscriptions. At the top of this list, there's a 'Filter by name...' input field. Below it, it says '9 items' and lists three entries: 'AzureCloudService1', 'AzureCloudService2', and 'AzureCloudService8-8-17', each preceded by a checkbox.

2. To create an empty resource group, select **Add**.

The screenshot shows the 'Resource groups' page again. The 'Add' button at the top left of the main content area is highlighted with a red box.

3. Provide a name and location for the new resource group. Select **Create**.

Resource group

Create an empty resource group

* Resource group name
ContosoGroup

* Subscription
Windows Azure MSDN - Visual Studio Ultir

* Resource group location
West US

4. You may need to select **Refresh** to see the recently created resource group.

Resource groups

+ Add Columns Refresh

Subscriptions: Windows Azure MSDN - Visual Studio Ultima

Filter items...

NAME
ContosoGroup
dashboards
ExampleGroup

The screenshot shows a list of resource groups. At the top, there are buttons for '+ Add', 'Columns' (which is highlighted with a red box), and 'Refresh'. Below that, it says 'Subscriptions: Windows Azure MSDN - Visual Studio Ultima'. There is a 'Filter items...' input field. The main area is a table with a single column labeled 'NAME'. It lists three groups: 'ContosoGroup', 'dashboards', and 'ExampleGroup'. Each item has a small blue icon next to it.

5. To customize the information displayed for your resource groups, select **Columns**.

Resource groups

+ Add Columns Refresh

6. Select the columns to add, and then select **Update**.

Choose columns

Resource groups

COLUMN

SUBSCRIPTION

LOCATION ⓘ

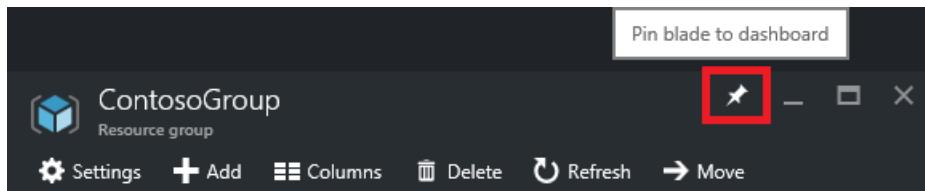
LOCATION ID ⓘ

RESOURCE GROUP ID ⓘ

STATUS ⓘ

SUBSCRIPTION ID

7. To learn about deploying resources to your new resource group, see [Deploy resources with Resource Manager templates and Azure portal](#).
8. For quick access to a resource group, you can pin the resource group to your dashboard.



9. The dashboard displays the resource group and its resources. You can select either the resource groups or any of its resources to navigate to the item.

All resources
ALL SUBSCRIPTIONS

Subscriptions Feedback Marketplace How it works

Resources
CONTOSOGROUP

- ContosoWebAppExample
- contososerverexample
- ContosoData
- ServicePlanf6ec7013-8b9e
- ContosoWebAppExample

Tag resources

You can apply tags to resource groups and resources to logically organize your assets. For information about working with tags, see [Using tags to organize your Azure resources](#).

1. To view the tags for a resource or a resource group, look for existing tags in the overview. If you have not previously applied tags, the list is empty.

demoGroup
Resource group

Search (Ctrl+)

Overview Activity log Access control (IAM)

Subscription (change)
Microsoft Azure Internal Consum...

Tags (change)
Click here to add tags

2. To add a tag, select **Click here to add tags**.
3. Provide a name and value. Select + to add the tag.

Edit tags
Tags for demoGroup

NAME	VALUE
Dept	Finance

4. Continue adding tags as needed. When done, select **Save**.

Edit tags

Tags for demoGroup

NAME	VALUE
Dept	Finance
Environment	Production
<i>name</i>	<i>value</i>

2 to be added

Save **Cancel**

- The tags are now displayed in the overview.

demoGroup

Resource group

Search (Ctrl+ /)

Add Edit columns Delete resource group

Subscription (change) Subscription ID
Microsoft Azure Internal Consum...

Tags (change)

Dept : Finance Environment : Production

Overview Activity log Access control (IAM) Tags

Filter by name... All types

- To add or delete a tag, select **change**.
- To delete a tag, select the trash icon. Then, select **Save**.

Edit tags

Tags for demoGroup

NAME	VALUE
Dept	Finance
Environment	Production
<i>name</i>	<i>value</i>

Save **Cancel**

To bulk assign tags to multiple resources:

- From any list of resources, select the checkbox for the resources you want to assign the tag.

Add **Edit columns** **Delete resource group** **Refresh** **Move** **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

2. Select **Assign tags**

Add **Edit columns** **Delete resource group** **Refresh** **Move** **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
<i>name</i>	<i>value</i>

Selected resources

- tfstoredemo1 (Storage account)
2 to be added ⓘ
- tfstoredemo2 (Storage account)
2 to be added ⓘ
- tfstoredemo3 (Storage account)
2 to be added ⓘ

Buttons: Assign (highlighted with a red box) | Cancel

To view all resources with a tag:

1. Select **All services** and **Tags**.

Microsoft Azure

The screenshot shows the Azure portal interface. On the left, there's a sidebar with a 'Create a resource' button and a 'Favorites' section containing 'Dashboard', 'All resources', 'Resource groups', and 'App Services'. Below these are 'All services' and 'Tags', both highlighted with red boxes. The main area is titled 'All services' with a 'Filter' input and a 'By category' dropdown. Under the 'GENERAL (14)' heading, there are several service tiles: 'Dashboard' (selected), 'Management Groups', 'Cost Management + Billing', 'Help + support', and 'Tags' (also highlighted with a red box).

2. Select the tag for viewing resources.

The screenshot shows the Microsoft Tags interface. At the top, it says "Tags" and "Microsoft". Below that is a "Refresh" button. A message states "Subscriptions: 3 of 10 selected – Don't see a subscription? Switch directories". A dropdown menu shows "3 subscriptions". A tooltip explains what tags are: "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." Below the tooltip, there is a list of tags:

- Dept : Finance
- Dept : IT
- Environment : Production
- Status : Approved

The "Environment : Production" tag is highlighted with a red box.

3. All resources with that tag are displayed.

The screenshot shows the Microsoft Tags interface with the title "Environment : Production". It includes a "Refresh" button and a message about selected subscriptions. A "Filter items..." input field and a "3 subscriptions" dropdown are also present. The main area displays a table of resources:

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.

The screenshot shows the Microsoft Tags interface with the title "Environment : Production". The "Pin" icon (a red-bordered square) is visible in the top right corner. The rest of the interface is identical to the previous screenshot, displaying the filtered list of resources.

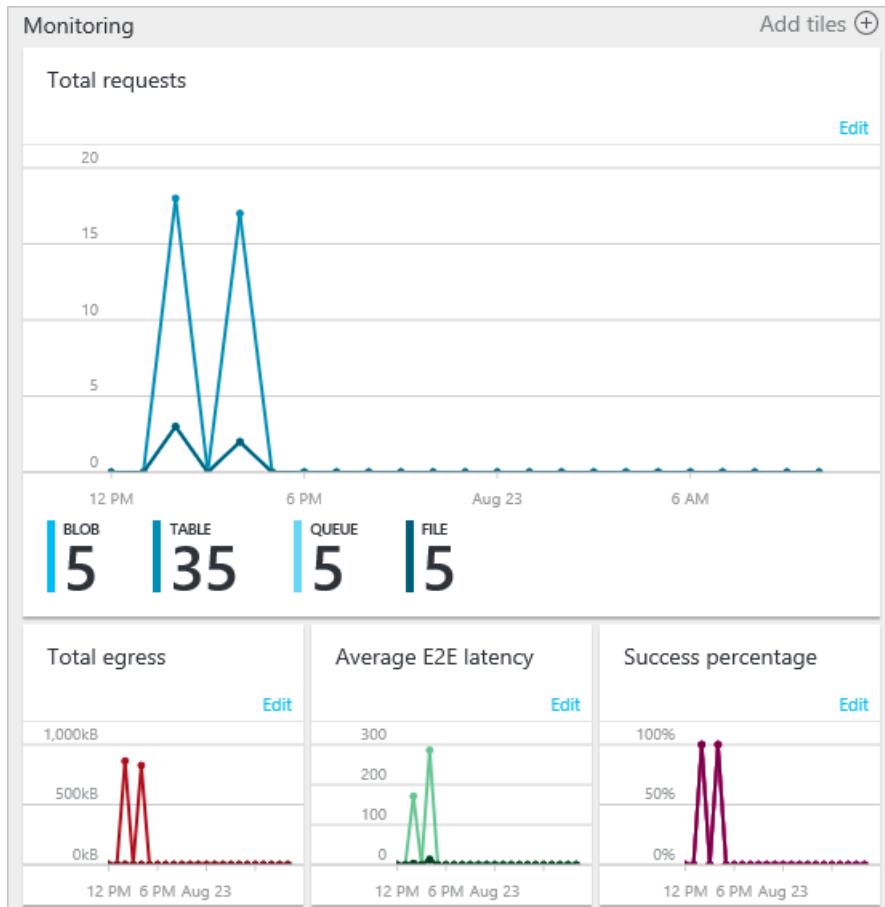
5. The view is available from the dashboard.

The Microsoft Azure dashboard interface. On the left, a sidebar lists 'Create a resource', 'All services', and 'FAVORITES' (Dashboard, All resources, Resource groups, App Services, SQL databases, Virtual machines, Storage accounts). The main area shows a 'Service Health' section with a heart icon and personalized support information, and an 'Environment : Production' section with a purple tag icon.

Monitor resources

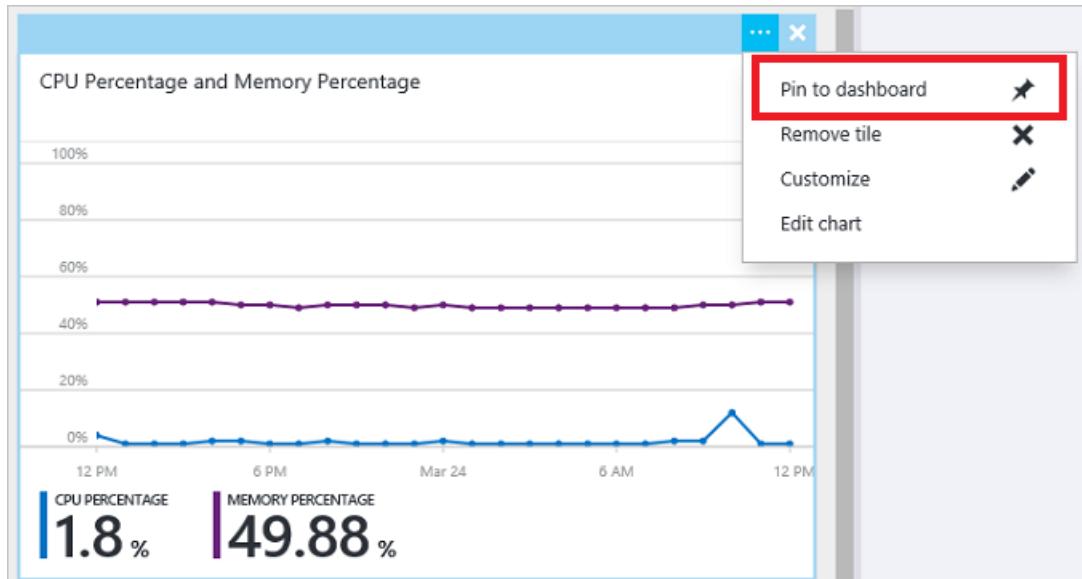
When you select a resource, the portal presents default graphs and tables for monitoring that resource type.

1. Select a resource and notice the **Monitoring** section. It includes graphs that are relevant to the resource type. The following image shows the default monitoring data for a storage account.

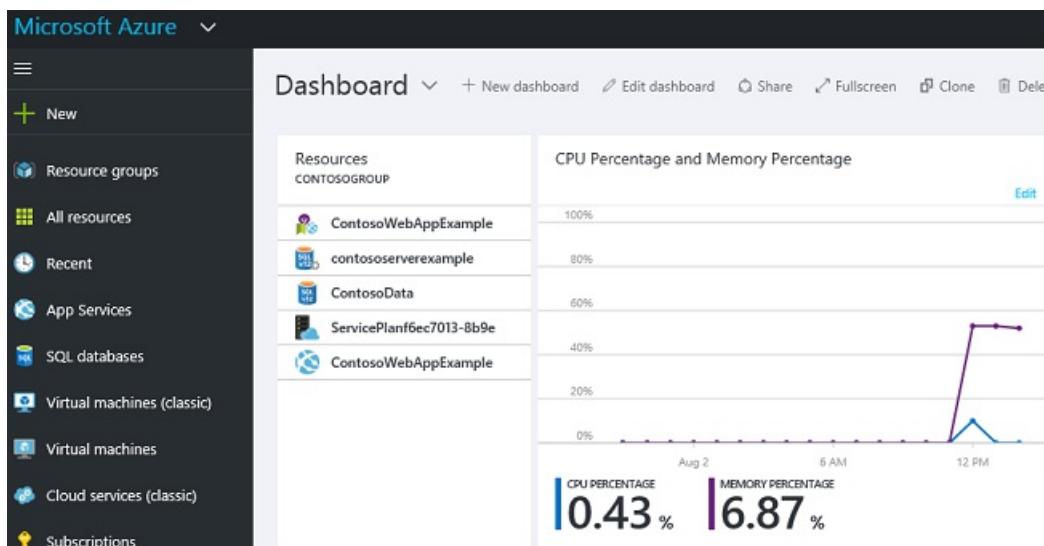


2. You can pin a section to your dashboard by selecting the ellipsis (...) above the section. You can also customize the size the section or remove it completely. The following image shows how to pin, customize, or

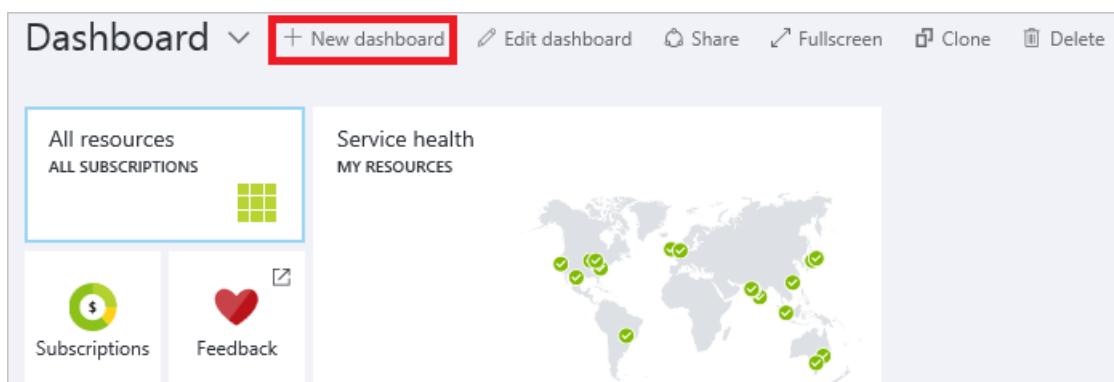
remove the CPU and Memory section.



3. After pinning the section to the dashboard, you will see the summary on the dashboard. And, selecting it immediately takes you to more details about the data.



4. To completely customize the data you monitor through the portal, navigate to your default dashboard, and select **New dashboard**.



5. Give your new dashboard a name and drag tiles onto the dashboard. The tiles are filtered by different options.

The screenshot shows the Tile Gallery for App Services in the Azure portal. The search bar at the top has 'Type' selected. A dropdown menu shows 'App Services' is currently chosen. Below that, a 'Resources' dropdown shows 'tfwebapp' and a link to 'See related resources'. Under 'Categories', 'All' is selected. There are two tiles: 'Process explorer' (monitoring icon) and 'Streaming logs' (log icon).

To learn about working with dashboards, see [Creating and sharing dashboards in the Azure portal](#).

Manage resources

When viewing a resource in the portal, you see the options for managing that particular resource.

The screenshot shows the details for a virtual machine named 'tfvm'. The top navigation bar includes 'tfvm' and 'Virtual machine'. The main area has a 'Connect' button, followed by 'Start', 'Restart', 'Stop', and 'Delete' buttons, all highlighted with a red box. The left sidebar is also highlighted with a red box. The 'Essentials' section displays the following information:

Resource group	testrg1	Computer name	tfvm
Status	Running	Operating system	Windows
Location	West US	Size	Standard DS1 v2 (1 vCPU)
Subscription name	Windows Azure MSDN - Visual Studio Ul...	Public IP address/DNS	40.118.247.228
Subscription ID		Virtual network/subnet	testrg1-vnet/default

The 'Monitoring' section shows a chart for 'CPU percentage' with values from 0% to 100%. The chart has a single data series with a sharp peak reaching nearly 100%.

From these options, you can perform operations such as starting and stopping a virtual machine, or reconfiguring the properties of the virtual machine.

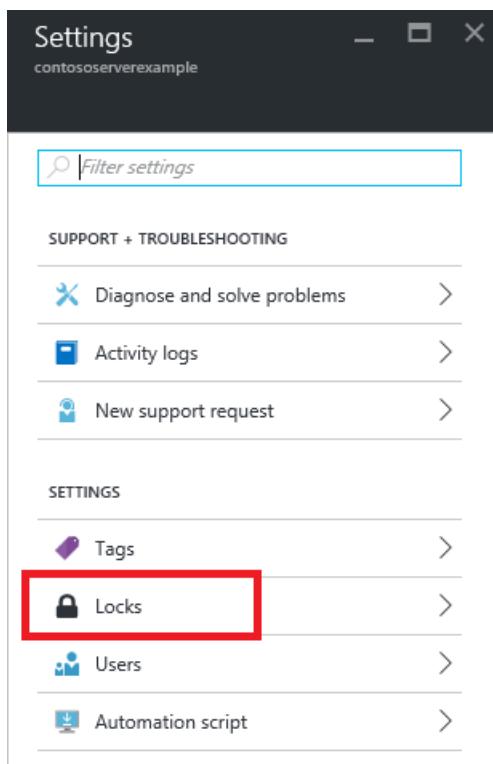
Move resources

If you need to move resources to another resource group or another subscription, see [Move resources to new resource group or subscription](#).

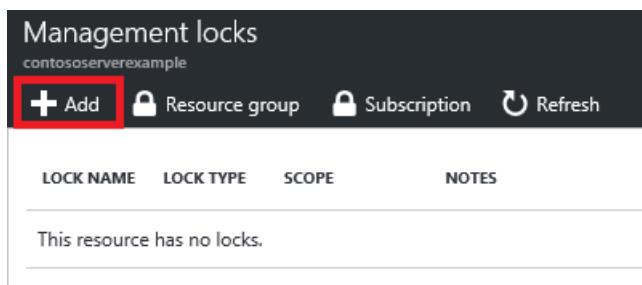
Lock resources

You can lock a subscription, resource group, or resource to prevent other users in your organization from accidentally deleting or modifying critical resources. For more information, see [Lock resources with Azure Resource Manager](#).

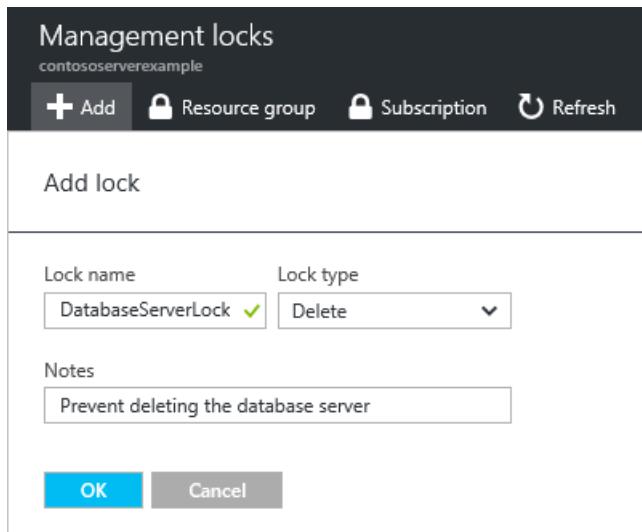
1. In the Settings blade for the resource, resource group, or subscription that you wish to lock, select **Locks**.



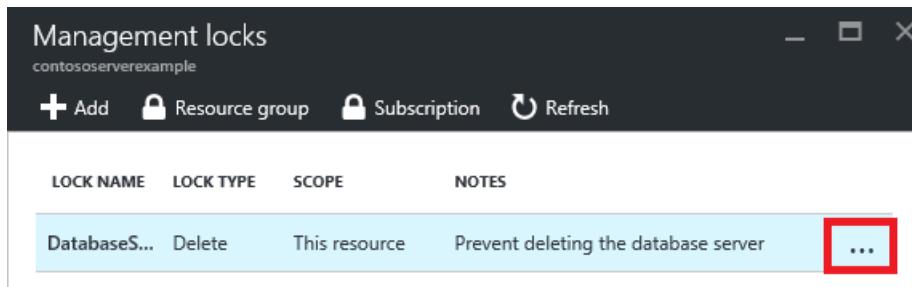
2. To add a lock, select **Add**. If you want to create a lock at a parent level, select the parent. The currently selected resource inherits the lock from the parent. For example, you could lock the resource group to apply a lock to all its resources.



3. Give the lock a name and lock level. Optionally, you can add notes that describe the lock.



4. To delete the lock, select the ellipsis and **Delete** from the available options.



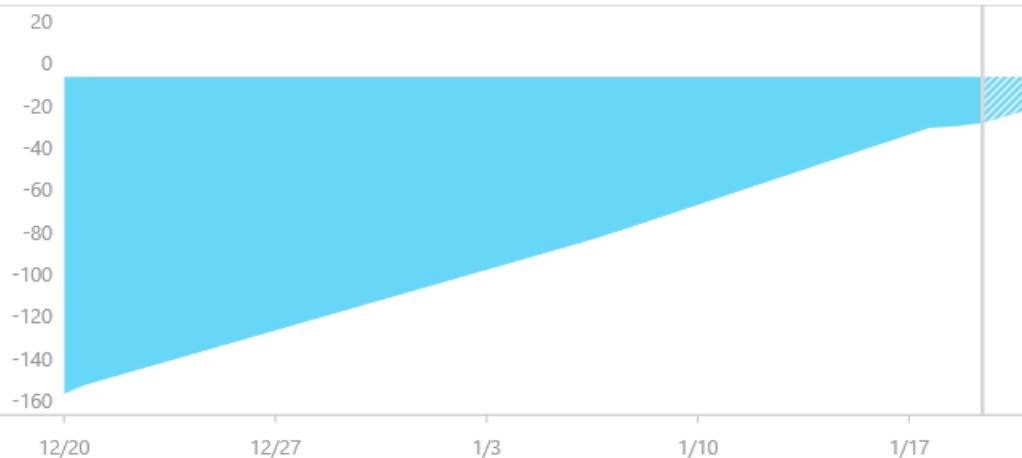
View your subscription and costs

You can view information about your subscription and the rolled-up costs for all your resources. Select **Subscriptions** and the subscription you want to see. You might only have one subscription to select.

The screenshot shows the Microsoft Azure portal with the navigation bar 'Microsoft Azure > Subscriptions > Windows Azure MSDN - Visual Studio Ultimate'. On the left, a sidebar lists various resources like 'Resource groups', 'All resources', and 'Subscriptions'. The 'Subscriptions' link is highlighted with a red box. The main area shows a table of subscriptions with a single row selected: 'Windows Azure MSDN - Visual Studio Ultim...'.

You see the burn rate.

Burn rate



And, a breakdown of costs by resource type.

Cost by resource

WINDOWS AZURE MSDN - VISUAL STUDIO ULTIMATE



STANDARD C1 (HOURS) - AZURE REDIS CACHE	97.84 USD
SHARED APP SERVICE HOURS - AZURE APP SERV	18.38 USD
BASIC DATABASE DAYS - SQL DATABASE	6.43 USD
OTHERS	5.59 USD

Export template

After setting up your resource group, you may want to view the Resource Manager template for the resource group. Exporting the template offers two benefits:

1. You can easily automate future deployments of the solution because the template contains all the complete infrastructure.
2. You can become familiar with template syntax by looking at the JavaScript Object Notation (JSON) that represents your solution.

For step-by-step guidance, see [Export Azure Resource Manager template from existing resources](#).

Delete resource group or resources

Deleting a resource group deletes all the resources contained within it. You can also delete individual resources within a resource group. Use caution when deleting a resource group. That resource group might contain resources that resources in other resource groups depend on.



Next steps

- To view activity logs, see [Audit operations with Resource Manager](#).
- To view details about a deployment, see [View deployment operations](#).
- To deploy resources through the portal, see [Deploy resources with Resource Manager templates and Azure portal](#).
- To manage access to resources, see [Use role assignments to manage access to your Azure subscription resources](#).
- For guidance on how enterprises can use Resource Manager to effectively manage subscriptions, see [Azure enterprise scaffold - prescriptive subscription governance](#).

Use tags to organize your Azure resources

5/23/2018 • 9 minutes to read • [Edit Online](#)

You apply tags to your Azure resources to logically organize them by categories. Each tag consists of a name and a value. 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.

The following limitations apply to tags:

- Each resource or resource group can have a maximum of 15 tag name/value pairs. This limitation applies only to tags directly applied to the resource group or resource. A resource group can contain many resources that each have 15 tag name/value pairs. If you have more than 15 values that you need to associate with a resource, use a JSON string for the tag value. The JSON string can contain many values that are applied to a single tag name. This article shows an example of assigning a JSON string to the tag.
- 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.
- 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: <, >, %, &, \, ?, /

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](#).

PowerShell

The examples in this article require version 6.0 or later of Azure PowerShell. If you do not have version 6.0 or later, [update your version](#).

To see the existing tags for a *resource group*, use:

```
(Get-AzureRmResourceGroup -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-AzureRmResource -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-AzureRmResource -ResourceName examplevnet -ResourceGroupName examplegroup).Tags
```

To get *resource groups that have a specific tag*, use:

```
(Get-AzureRmResourceGroup -Tag @{ Dept="Finance" }).ResourceGroupName
```

To get *resources that have a specific tag*, use:

```
(Get-AzureRmResource -Tag @{ Dept="Finance"}).Name
```

To get *resources that have a specific tag name*, use:

```
(Get-AzureRmResource -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-AzureRmResourceGroup -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-AzureRmResourceGroup -Name examplegroup).Tags  
$tags.Add("Status", "Approved")  
Set-AzureRmResourceGroup -Tag $tags -Name examplegroup
```

To add tags to a *resource without existing tags*, use:

```
$r = Get-AzureRmResource -ResourceName examplevnet -ResourceGroupName examplegroup  
Set-AzureRmResource -Tag @{ Dept="IT"; Environment="Test" } -ResourceId $r.ResourceId -Force
```

To add tags to a *resource that has existing tags*, use:

```
$r = Get-AzureRmResource -ResourceName examplevnet -ResourceGroupName examplegroup  
$r.Tags.Add("Status", "Approved")  
Set-AzureRmResource -Tag $r.Tags -ResourceId $r.ResourceId -Force
```

To apply all tags from a resource group to its resources, and *not retain existing tags on the resources*, use the following script:

```
$groups = Get-AzureRmResourceGroup  
foreach ($g in $groups)  
{  
    Get-AzureRmResource -ResourceGroupName $g.ResourceGroupName | ForEach-Object {Set-AzureRmResource -  
    ResourceId $_.ResourceId -Tag $g.Tags -Force }  
}
```

To apply all tags from a resource group to its resources, and *retain existing tags on resources that are not duplicates*, use the following script:

```
$group = Get-AzureRmResourceGroup "examplegroup"
if ($group.Tags -ne $null) {
    $resources = Get-AzureRmResource -ResourceGroupName $group.ResourceGroupName
    foreach ($r in $resources)
    {
        $resourcetags = (Get-AzureRmResource -ResourceId $r.ResourceId).Tags
        if ($resourcetags)
        {
            foreach ($key in $group.Tags.Keys)
            {
                if (-not($resourcetags.ContainsKey($key)))
                {
                    $resourcetags.Add($key, $group.Tags[$key])
                }
            }
            Set-AzureRmResource -Tag $resourcetags -ResourceId $r.ResourceId -Force
        }
        else
        {
            Set-AzureRmResource -Tag $group.Tags -ResourceId $r.ResourceId -Force
        }
    }
}
```

To remove all tags, pass an empty hash table:

```
Set-AzureRmResourceGroup -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 retain 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 *retain 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
done

```

Templates

To tag a resource during deployment, add the `tags` element to the resource you are 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",
  "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": { }
    }
  ]
}
```

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": {
    "tagValues": {
      "type": "object",
      "defaultValue": {
        "Dept": "Finance",
        "Environment": "Production"
      }
    }
  },
  "resources": [
    {
      "apiVersion": "2016-01-01",
      "type": "Microsoft.Storage/storageAccounts",
      "name": "[concat('storage', uniqueString(resourceGroup().id))]",
      "location": "[resourceGroup().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 cannot 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",
  "resources": [
    {
      "apiVersion": "2016-01-01",
      "type": "Microsoft.Storage/storageAccounts",
      "name": "[concat('storage', uniqueString(resourceGroup().id))]",
      "location": "[resourceGroup().location]",
      "tags": {
        "CostCenter": "{\"Dept\":\"Finance\", \"Environment\":\"Production\"}"
      },
      "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.

The screenshot shows the Azure portal interface for a resource group named 'demoGroup'. On the left, there's a navigation bar with 'Overview', 'Activity log', and 'Access control (IAM)'. On the right, under the 'Tags (change)' section, there's a button labeled 'Click here to add tags' which is highlighted with a red box.

2. To add a tag, select **Click here to add tags**.
3. Provide a name and value. Select + to add the tag.

The screenshot shows the 'Edit tags' dialog for the 'demoGroup' resource group. It displays a table with two columns: 'NAME' and 'VALUE'. A single tag entry is shown: 'Dept' in the NAME column and 'Finance' in the VALUE column. There is a red box around the '+' icon in the top right corner of the table header.

4. Continue adding tags as needed. When done, select **Save**.

The screenshot shows the 'Edit tags' dialog with three tag entries: 'Dept : Finance', 'Environment : Production', and a third entry 'name : value' which is currently empty. Below the table, it says '2 to be added'. At the bottom, there are 'Save' and 'Cancel' buttons, with 'Save' being highlighted with a red box.

5. The tags are now displayed in the overview.

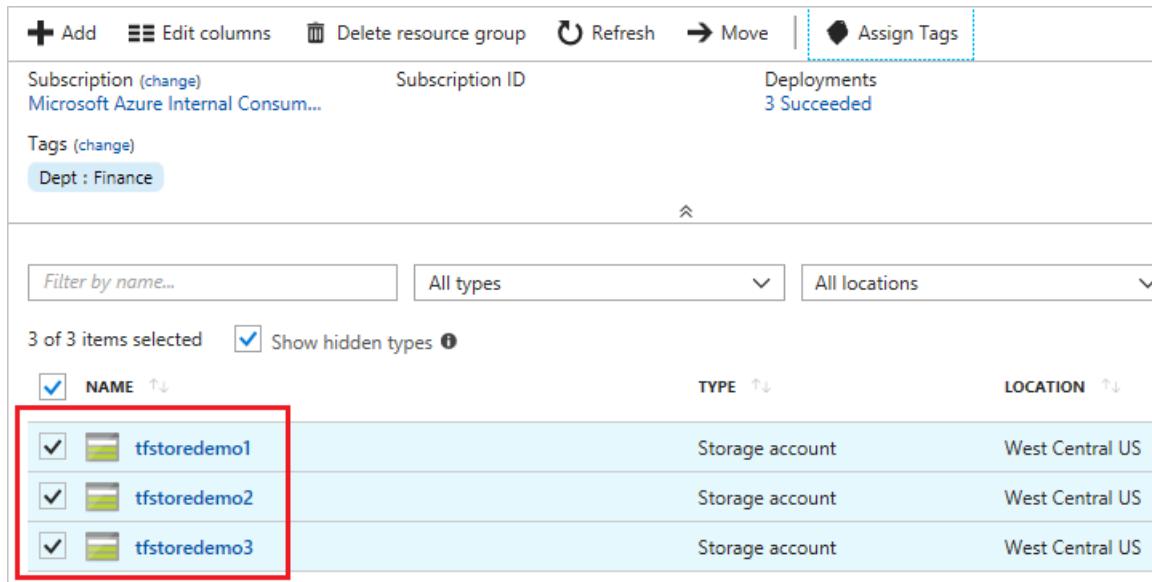
The screenshot shows the 'demoGroup' overview page. In the center, under the 'Tags (change)' section, the applied tags 'Dept : Finance' and 'Environment : Production' are listed. These tags are also highlighted with a red box.

6. To add or delete a tag, select **change**.
7. To delete a tag, select the trash icon. Then, select **Save**.

Edit tags	
Tags for demoGroup	
NAME	VALUE
Dept	Finance
Environment	Production
name	value

To bulk assign tags to multiple resources:

- From any list of resources, select the checkbox for the resources you want to assign the tag.

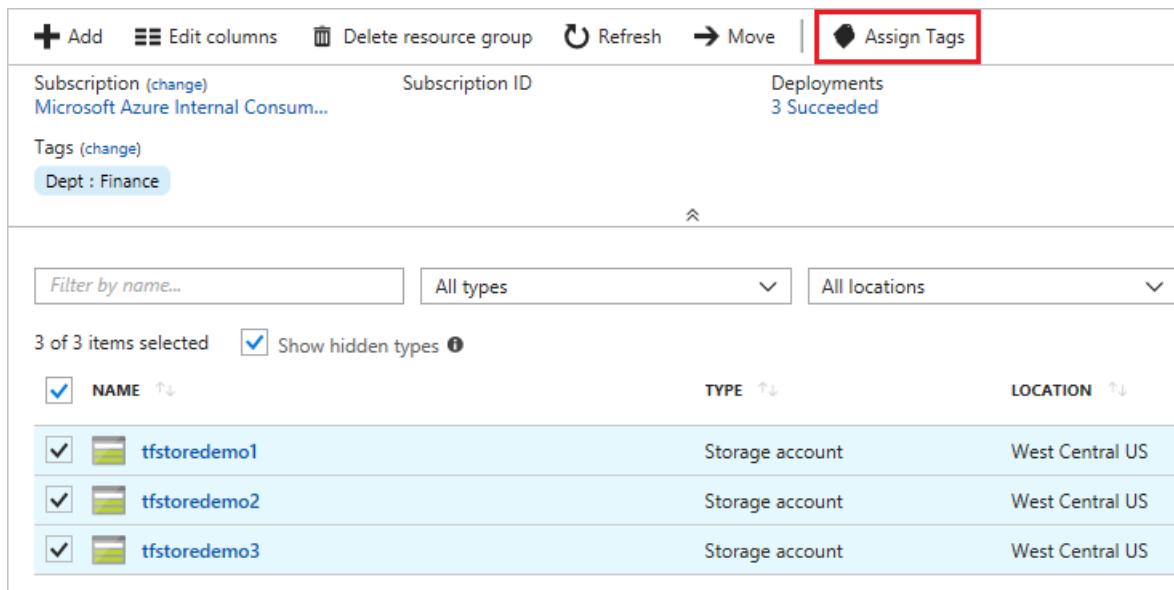


The screenshot shows the Azure portal interface for managing resources. At the top, there's a toolbar with 'Add', 'Edit columns', 'Delete resource group', 'Refresh', 'Move', and a highlighted 'Assign Tags' button. Below the toolbar, it displays 'Subscription (change)' as 'Microsoft Azure Internal Consum...' and 'Subscription ID'. Under 'Tags (change)', there's a tag named 'Dept : Finance'. The main area shows a list of resources with the following details:

NAME	TYPE	LOCATION
tfstoredemo1	Storage account	West Central US
tfstoredemo2	Storage account	West Central US
tfstoredemo3	Storage account	West Central US

The rows for 'tfstoredemo1', 'tfstoredemo2', and 'tfstoredemo3' are highlighted with a red box, indicating they are selected for bulk tagging. There are also filters at the bottom for 'Filter by name...', 'All types', and 'All locations'.

- Select **Assign Tags**



This screenshot is identical to the previous one, showing the same list of three selected storage accounts ('tfstoredemo1', 'tfstoredemo2', 'tfstoredemo3') in the Azure portal. The 'Assign Tags' button is highlighted with a red box, indicating the next step in the process.

- After each name and value, select +. When done, select **Assign**.

Assign tags

Assign tags to 3 resources

NAME	VALUE
Status	Approved
Environment	Production
<i>name</i>	<i>value</i>

Selected resources

- tfstoredemo1 (Storage account)
2 to be added ⓘ
- tfstoredemo2 (Storage account)
2 to be added ⓘ
- tfstoredemo3 (Storage account)
2 to be added ⓘ

Assign **Cancel**

To view all resources with a tag:

1. Select **All services** and **Tags**.

Microsoft Azure

+ Create a resource

All services By category ▾

GENERAL (14)

- Dashboard
- Management Groups
- Cost Management + Billing
- Help + support
- 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

i 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

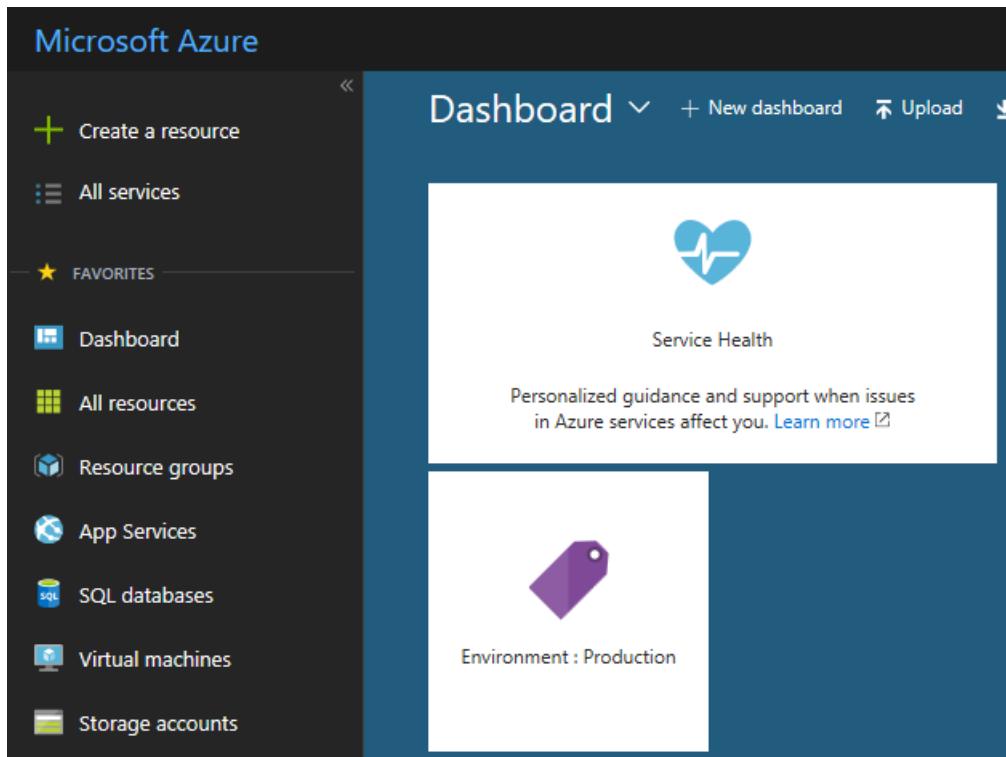
Refresh

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 are 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 portal](#) or EA portal. For more information about programmatic access to billing information, see [Gain insights into your Microsoft Azure resource consumption](#). For REST API operations, see [Azure Billing REST API Reference](#).

When you download the usage CSV for services that support tags with billing, the tags appear in the **Tags** column. For more information, see [Understand your bill for Microsoft Azure](#).

Daily Usage							Tags
Usage Date	Meter Category	Unit	Consume	Resource Gr	Instance Id		
5/14/2015	"Virtual Machines"	"Hours"	3.999984	"computeRG"	"virtualMachines/catalogVM"		{"costCenter":"finance", "env":"prod"}
5/14/2015	"Virtual Machines"	"Hours"	3.999984	"businessRG"	"virtualMachines/dataVM"		{"costCenter":"hr", "env":"test"}

Next steps

- You can apply restrictions and conventions across your subscription by using customized policies. A policy that you define might require that all resources have a value for a particular tag. For more information, see [What is Azure Policy?](#)
- For an introduction to using Azure PowerShell when you're deploying resources, see [Using Azure PowerShell with Azure Resource Manager](#).

- For an introduction to using the Azure CLI when you're deploying resources, see [Using the Azure CLI for Mac, Linux, and Windows with Azure Resource Manager](#).
- For an introduction to using the portal, see [Using the Azure portal to manage your Azure resources](#).
- For guidance on how enterprises can use Resource Manager to effectively manage subscriptions, see [Azure enterprise scaffold - prescriptive subscription governance](#).

Move resources to new resource group or subscription

8/3/2018 • 13 minutes to read • [Edit Online](#)

This article shows you how to move resources to either a new subscription or a new resource group in the same subscription. You can use the portal, PowerShell, Azure CLI, or the REST API to move resource. The move operations in this article are available to you without any assistance from Azure support.

When moving resources, both the source group and the target group are locked during the 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.

You can't change the location of the resource. Moving a resource only moves it to a new resource group. The new resource group may have a different location, but that doesn't change the location of the resource.

NOTE

This article describes how to move resources within an existing Azure account offering. If you actually want to change your Azure account offering (such as upgrading from pay-as-you-go to pre-pay) while continuing to work with your existing resources, see [Switch your Azure subscription to another offer](#).

Checklist before moving resources

There are some important steps to perform before moving a resource. By verifying these conditions, you can avoid errors.

1. 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-AzureRmSubscription -SubscriptionName <your-source-subscription>).TenantId  
(Get-AzureRmSubscription -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](#)
2. The service must enable the ability to move resources. This article lists which services enable moving resources and which services don't enable moving resources.
 3. 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 encounter this problem 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-AzureRmContext -Subscription <destination-subscription-name-or-id>
Get-AzureRmResourceProvider -ListAvailable | Select-Object ProviderNamespace, RegistrationState
```

To register a resource provider, use:

```
Register-AzureRmResourceProvider -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
```

4. 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.

5. 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](#).

6. When possible, break large moves into separate move operations. Resource Manager immediately fails attempts to move more than 800 resources in a single operation. However, moving less than 800 resources may also fail by timing out.

When to call support

You can move most resources through the self-service operations shown in this article. Use the self-service operations to:

- Move Resource Manager resources.
- Move classic resources according to the [classic deployment limitations](#).

Contact [support](#) when you need to:

- Move your resources to a new Azure account (and Azure Active Directory tenant) and you need help with the instructions in the preceding section.
- Move classic resources but are having trouble with the limitations.

Services that can be moved

The services that enable moving to both a new resource group and subscription are:

- API Management
- App Service apps (web apps) - see [App Service limitations](#)
- App Service Certificates
- Application Insights
- Analysis Services
- Automation
- Azure Active Directory B2C
- Azure Cosmos DB
- Azure Maps
- Azure Relay
- Azure Stack - registrations
- Azure Migrate
- Batch
- BizTalk Services
- Bot Service
- CDN
- Cloud Services - see [Classic deployment limitations](#)
- Cognitive Services
- Container Registry
- Content Moderator
- Data Catalog
- Data Factory
- Data Lake Analytics
- Data Lake Store
- DNS
- Event Grid
- Event Hubs
- HDInsight clusters - see [HDInsight limitations](#)
- IoT Hubs
- Key Vault
- Load Balancers - see [Load Balancer limitations](#)
- Log Analytics
- Logic Apps
- Machine Learning - Machine Learning Studio web services can be moved to a resource group in the same subscription, but not a different subscription. Other Machine Learning resources can be moved across subscriptions.
- Media Services
- Mobile Engagement
- Notification Hubs
- Operational Insights
- Operations Management
- Portal dashboards
- Power BI - both Power BI Embedded and Power BI Workspace Collection
- Public IP - see [Public IP limitations](#)
- Redis Cache
- Scheduler
- Search

- Service Bus
- Service Fabric
- SignalR Service
- Storage
- Storage (classic) - see [Classic deployment limitations](#)
- Stream Analytics - Stream Analytics jobs can't be moved when in running state.
- SQL Database server - database and server must reside 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.
- Time Series Insights
- Traffic Manager
- Virtual Machines - VMs with managed disks can't be moved. See [Virtual Machines limitations](#)
- Virtual Machines (classic) - see [Classic deployment limitations](#)
- Virtual Machine Scale Sets - see [Virtual Machines limitations](#)
- Virtual Networks - see [Virtual Networks limitations](#)
- Visual Studio Team Services - VSTS accounts with non-Microsoft extension purchases must [cancel their purchases](#) before they can move the account across subscriptions.
- VPN Gateway

Services that cannot be moved

The services that currently don't enable moving a resource are:

- AD Domain Services
- AD Hybrid Health Service
- Application Gateway
- Azure Database for MySQL
- Azure Database for PostgreSQL
- Azure Database Migration
- Azure Databricks
- Batch AI
- Certificates - App Service Certificates can be moved, but uploaded certificates have [limitations](#).
- Container Service
- Dynamics LCS
- Express Route
- Kubernetes Service
- Lab Services - move to new resource group in same subscription is enabled, but cross subscription move isn't enabled.
- Load Balancers - see [Load Balancer limitations](#)
- Managed Applications
- Managed Disks - see [Virtual Machines limitations](#)
- Microsoft Genomics
- Public IP - see [Public IP limitations](#)
- Recovery Services vault - also don't move the Compute, Network, and Storage resources associated with the Recovery Services vault, see [Recovery Services limitations](#).
- SAP HANA on Azure
- Security
- Site Recovery

- StorSimple Device Manager
- Virtual Networks (classic) - see [Classic deployment limitations](#)

Virtual Machines limitations

Managed disks don't support move. This restriction means that several related resources can't be moved too. You can't move:

- Managed disks
- Virtual machines with the managed disks
- Images created from managed disks
- Snapshots created from managed disks
- Availability sets with virtual machines with managed disks

Although you can't move a managed disk, you can create a copy and then create a new virtual machine from the existing managed disk. For more information, see:

- Copy managed disks in the same subscription or different subscription with [PowerShell](#) or [Azure CLI](#)
- Create a virtual machine using an existing managed OS disk with [PowerShell](#) or [Azure CLI](#).

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 with certificate stored in Key Vault can be moved to a new resource group in the same subscription, but not across subscriptions.

Virtual Networks limitations

When moving a virtual network, you must also move its dependent resources. For example, you must move gateways with the 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.

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 a Redis Cache resource is deployed into a subnet, that subnet has a resource navigation link.

You can't move a virtual network to a different subscription if the virtual network contains a custom DNS server. To move the virtual network, set it to Default (Azure-provided) DNS server. After the move, reconfigure the custom DNS server.

App Service limitations

The limitations for moving App Service resources differ based on whether you're moving the resources within a subscription or to a new subscription.

The limitations described in these sections apply to uploaded certificates, not App Service Certificates. You can move App Service Certificates to a new resource group or subscription without limitations. If you have multiple web apps that use the same App Service Certificate, first move all the web apps, then move the certificate.

Moving within the same subscription

When moving a Web App *within the same subscription*, you can't move the uploaded SSL certificates. However, you can move a Web App to the new resource group without moving its uploaded SSL 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 uploaded certificate from the Web App.
2. Move the Web App.
3. Upload the certificate to the moved Web App.

Moving 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, it must be moved back to that original resource group first, and then it can be moved across subscriptions.

Classic deployment limitations

The options for moving resources deployed through the classic model differ based on whether you're moving the resources within a subscription or to a new subscription.

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.

New subscription

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 contain 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, perform 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/validateSubscriptionMoveAvailability?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.ClassicCompute/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.

Recovery Services limitations

Move isn't enabled for Storage, Network, or Compute resources used to set up disaster recovery with Azure Site Recovery.

For example, suppose you have set up replication of your on-premises machines to a storage account (Storage1) and want the protected machine to come up after failover to Azure as a virtual machine (VM1) attached to a virtual network (Network1). You can't move any of these Azure resources - Storage1, VM1, and Network1 - across resource groups within the same subscription or across subscriptions.

To move a VM enrolled in **Azure backup** between resource groups:

1. Temporarily stop backup and retain backup data
2. Move the VM to the target resource group
3. Reprotect it under the same/new vault. Users can restore from the available restore points created before the move operation. If the user moves the backed-up VM across subscriptions, step 1 and step 2 remain the same. In step 3, user needs to protect the VM under a new vault present/ created in the target subscription. Recovery Services vault doesn't support cross subscription backups.

HDInsight limitations

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.

Search limitations

You can't move multiple Search resources placed in different regions all at once. In such a case, you need to move them separately.

Load Balancer limitations

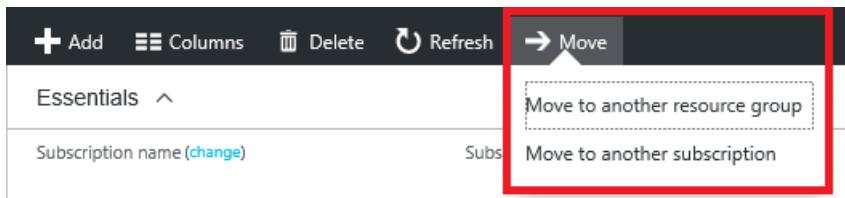
Basic SKU Load Balancer can be moved. Standard SKU Load Balancer can't be moved.

Public IP limitations

Basic SKU Public IP can be moved. Standard SKU Public IP can't be moved.

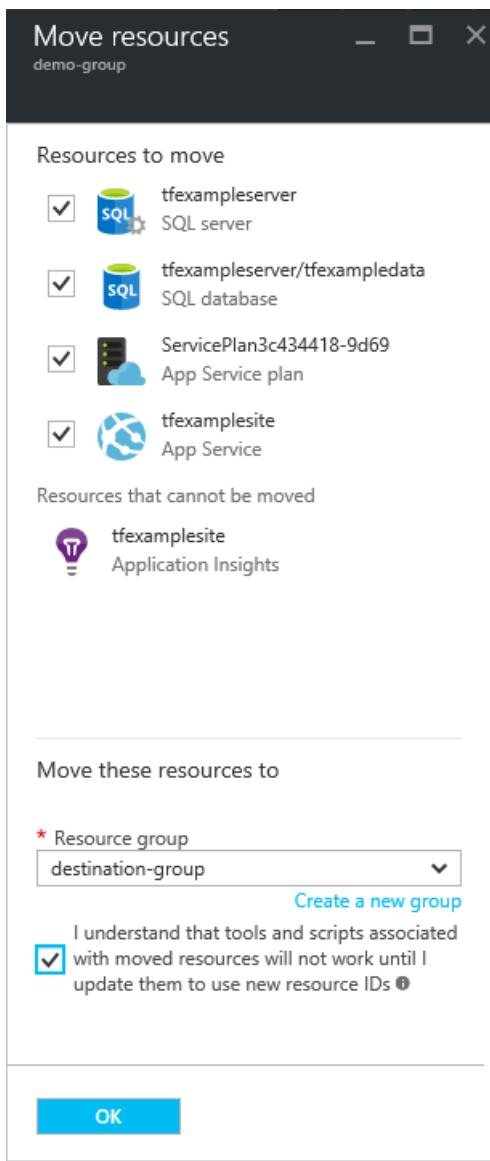
Use portal

To move resources, select the resource group containing those resources, and then select the **Move** button.



Select whether you're moving the resources to a new resource group or a new subscription.

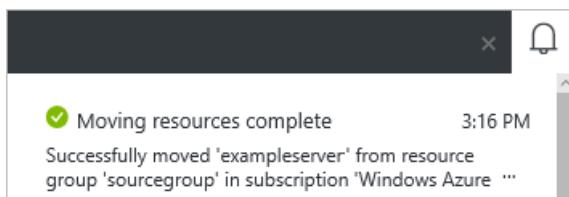
Select the resources to move and the destination resource group. Acknowledge that you need to update scripts for these resources and select **OK**. If you selected the edit subscription icon in the previous step, you must also select the destination subscription.



In **Notifications**, you see that the move operation is running.



When it has completed, you're notified of the result.



Use PowerShell

To move existing resources to another resource group or subscription, use the [Move-AzureRmResource](#) command. The following example shows how to move multiple resources to a new resource group.

```
$webapp = Get-AzureRmResource -ResourceGroupName OldRG -ResourceName ExampleSite
$plan = Get-AzureRmResource -ResourceGroupName OldRG -ResourceName ExamplePlan
Move-AzureRmResource -DestinationResourceGroupName NewRG -ResourceId $webapp.ResourceId, $plan.ResourceId
```

To move to a new subscription, include a value for the `--destination-subscription-id` parameter.

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 multiple 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.

Use REST API

To move existing resources to another resource group or subscription, run:

```
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. For more information about the move REST operation, see [Move resources](#).

Next steps

- To learn about PowerShell cmdlets for managing your subscription, see [Using Azure PowerShell with Resource Manager](#).
- To learn about Azure CLI commands for managing your subscription, see [Using the Azure CLI with Resource Manager](#).
- To learn about portal features for managing your subscription, see [Using the Azure portal to manage resources](#).
- To learn about applying a logical organization to your resources, see [Using tags to organize your resources](#).

Programmatically create Azure Enterprise subscriptions (preview)

7/30/2018 • 6 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](#).

Prerequisites

You must have an Owner or Contributor 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](#) (log-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:

```
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. Use the `id` as the `enrollmentAccount` value that you use to create the subscription in the next step.

Create subscriptions under a specific enrollment account

The following example creates a request to create subscription named *Dev Team Subscription* and subscription offer is *MS-AZR-0017P* (regular EA). The enrollment account is `747ddfe5-xxxx-xxxx-xxxx-xxxxxxxxxxxx` (placeholder value, this value is a GUID), which is the enrollment account for SignUpEngineering@contoso.com. It also optionally adds two users as RBAC Owners for the subscription.

- [REST](#)
- [PowerShell](#)
- [Azure CLI](#)

Use the `id` of the `enrollmentAccount` in the path of the request to create subscription.

```
POST https://management.azure.com/providers/Microsoft.Billing/enrollmentAccounts/747ddfe5-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/providers/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
--------------	----------	------	-------------

ELEMENT NAME	REQUIRED	TYPE	DESCRIPTION
<code>displayName</code>	No	String	The display name of the subscription. If not specified, it's set to the name of the offer, like "Microsoft Azure Enterprise."
<code>offerType</code>	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).
<code>owners</code>	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 50 subscriptions per account. After that, subscriptions can only be created by using Account Center.
- There needs to be at least one EA or EA Dev/Test subscriptions under the account, which means the Account Owner has gone through manual sign-up at least once.
- Users who aren't Account Owners, but were added to an enrollment account via RBAC, can't create subscriptions using 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)

7/31/2018 • 2 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.

To create a subscription, see [Programmatically create Azure Enterprise subscriptions \(preview\)](#).

Delegate access to an enrollment account using RBAC

To give another user or service principal the ability to create subscriptions against a specific account, [give them an RBAC Owner role at the scope of the enrollment account](#). The following example gives a user in the tenant with `principalId` of `<userObjectId>` (for SignUpEngineering@contoso.com) an Owner role on the enrollment account. To find the enrollment account ID and principal ID, see [Programmatically create Azure Enterprise subscriptions \(preview\)](#).

- [REST](#)
- [PowerShell](#)
- [Azure CLI](#)

```
PUT https://management.azure.com/providers/Microsoft.Billing/enrollmentAccounts/747ddfe5-xxxx-xxxx-xxxx-xxxxxxxxxx/providers/Microsoft.Authorization/roleAssignments/<roleAssignmentGuid>?api-version=2015-07-01

{
  "properties": {
    "roleDefinitionId": "/providers/Microsoft.Billing/enrollmentAccounts/providers/Microsoft.Authorization/roleDefinitions/<ownerRoleId>",
    "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/<ownerRoleDefinitionId>",
    "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/<ownerRoleDefinitionId>",
  "type": "Microsoft.Authorization/roleAssignments",
  "name": "<roleAssignmentGuid>"
}
}
```

Once a user becomes an RBAC Owner for your enrollment account, they can programmatically create subscriptions under it. A subscription created by a delegated user still has the original Account Owner as Service Admin, but it also has the delegated user as an Owner by default.

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'"
```

NOTE

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](#)

Create management groups for resource organization and management

8/2/2018 • 2 minutes to read • [Edit Online](#)

Management groups are containers that help you manage access, policy, and compliance across multiple subscriptions. Create these containers to build an effective and efficient hierarchy that can be used with [Azure Policy](#) and [Azure Role Based Access Controls](#). For more information on management groups, see [Organize your resources with Azure management groups](#).

The first management group created in the directory could take up to 15 minutes to complete. There are processes that run the first time to set up the management groups service within Azure for your directory. You receive a notification when the process is complete.

Create a management group

You can create the management group by using the portal, PowerShell, or Azure CLI.

Create in portal

1. Log into the [Azure portal](#).
2. Select **All services > Management groups**.
3. On the main page, select **New Management group**.

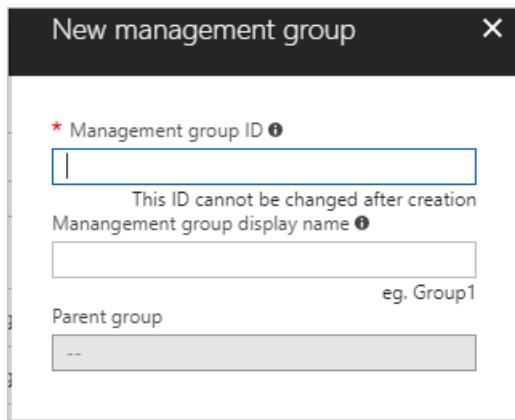
The screenshot shows the Azure portal's 'Management groups' page. At the top, there's a breadcrumb navigation: Home > Management groups. Below that, it says 'Management groups contoso'. There are two buttons: '+ New management group' and 'Refresh'. A search bar is present. The main area shows a list of items under 'Root Management G... > Contoso Redmond'. The list includes:

NAME	ID	TYPE	MY ROLE	...
Azure Policy	<MG ID>	Management Group	Owner	...
Contoso IT	<MG ID>	Management Group	Owner	...
Contoso Marketing	<MG ID>	Management Group	Owner	...
Global	<MG ID>	Management Group	Owner	...
Storefront	<MG ID>	Management Group	Owner	...
Azure Test Sub	<MG ID>	Subscription	Owner	...
BillNotifications	<MG ID>	Subscription	Owner	...
Groups	<MG ID>	Subscription	Owner	...
Legacy Groups Classic	<MG ID>	Subscription	Owner	...
mye2esubprod1-C	<MG ID>	Subscription	Owner	...

To the right of the list, there's a tooltip with a small icon of three people connected by lines. The text reads: 'Using management groups helps you manage access, policy, and compliance by grouping multiple subscriptions together. [Learn more](#)'.

4. Fill in the management group ID field.

- The **Management Group ID** is the directory unique identifier that is used to submit commands on this management group. This identifier is not editable after creation as it is used throughout the Azure system to identify this group.
- The display name field is the name that is displayed within the Azure portal. A separate display name is an optional field when creating the management group and can be changed at any time.



5. Select **Save**

Create in PowerShell

Within PowerShell, you use the Add-AzureRmManagementGroups cmdlets:

```
C:\> New-AzureRmManagementGroup -GroupName Contoso
```

The **GroupName** is a unique identifier being created. This ID is used by other commands to reference this group and it cannot be changed later.

If you wanted the management group to show a different name within the Azure portal, you would add the **DisplayName** parameter with the string. For example, if you wanted to create a management group with the GroupName of Contoso and the display name of "Contoso Group", you would use the following cmdlet:

```
C:\> New-AzureRmManagementGroup -GroupName Contoso -DisplayName "Contoso Group" -ParentId ContosoTenant
```

Use the **ParentId** parameter to have this management group be created under a different management.

Create in Azure CLI

On Azure CLI, you use the az account management-group create command.

```
C:\> az account management-group create --group-name <YourGroupName>
```

Next steps

To Learn more about management groups, see:

- [Organize your resources with Azure management groups](#)
- [How to change, delete, or manage your management groups](#)
- [Install the Azure Powershell module](#)
- [Review the REST API Spec](#)
- [Install the Azure CLI Extension](#)

Manage your resources with management groups

8/2/2018 • 5 minutes to read • [Edit Online](#)

Management groups are containers that help you manage access, policy, and compliance across multiple subscriptions. You can change, delete, and manage these containers to have hierarchies that can be used with [Azure Policy](#) and [Azure Role Based Access Controls \(RBAC\)](#). To learn more about management groups, see [Organize your resources with Azure management groups](#).

To make changes to a management group, you must have an Owner or Contributor role on the management group. To see what permissions you have, select the management group and then select **IAM**. To learn more about RBAC Roles, see [Manage access and permissions with RBAC](#).

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](#).

Change the name of a management group

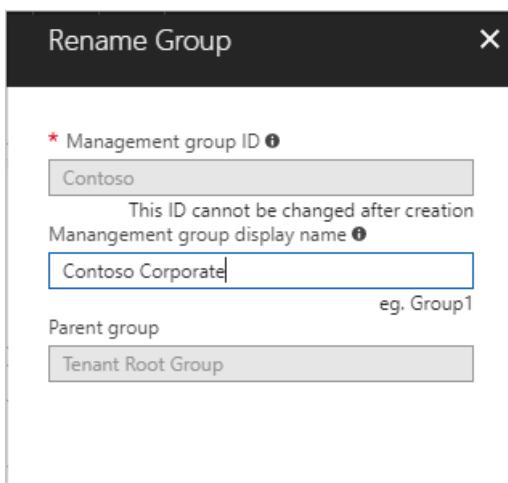
You can change the name of the management group by using the portal, PowerShell, or Azure CLI.

Change the name in the portal

1. Log into the [Azure portal](#)
2. Select **All services > Management groups**
3. Select the management group you would like to rename.
4. Select the **Rename group** option at the top of the page.



5. When the menu opens, enter the new name you would like to have displayed.



6. Select **Save**.

Change the name in PowerShell

To update the display name use **Update-AzureRmManagementGroup**. For example, to change a management groups name from "Contoso IT" to "Contoso Group", you run the following command:

```
C:\> Update-AzureRmManagementGroup -GroupName ContosoIt -DisplayName "Contoso Group"
```

Change the name in Azure CLI

For Azure CLI, use the update command.

```
az account management-group update --name Contoso --display-name "Contoso Group"
```

Delete a management group

To delete a management group, the following requirements must be met:

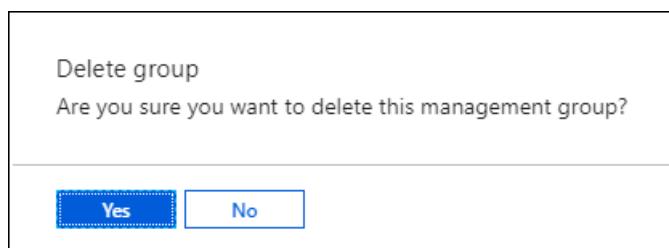
1. There are no child management groups or subscriptions under the management group.
 - To move a subscription out of a management group, see [Move subscription to another management group](#).
 - To move a management group to another management group, see [Move management groups in the hierarchy](#).
2. You have write permissions on the management group Owner or Contributor role on the management group.
To see what permissions you have, select the management group and then select **IAM**. To learn more on RBAC Roles, see [Manage access and permissions with RBAC](#).

Delete in the portal

1. Log into the [Azure portal](#)
2. Select **All services > Management groups**
3. Select the management group you would like to delete.
4. Select **Delete**.
 - If the icon is disabled, hovering your mouse selector over the icon shows you the reason.

The screenshot shows the Azure portal's Management Groups page. A management group named 'Newgroup1' is selected. The top navigation bar includes 'Rename Group', 'Delete' (which is highlighted with a red box), 'Move', 'Add management group', 'Add subscription', and 'Refresh'. On the right side, it displays the 'Parent management group' as 'Contoso Redmond' and 'Child management groups' as '0'. Below the main content area, there is a search bar labeled 'Search by name or ID' and a table with columns 'NAME' and 'ID'. The table shows one entry: 'No result'.

5. There's a window that opens confirming you want to delete the management group.



6. Select **Yes**

Delete in PowerShell

Use the **Remove-AzureRmManagementGroup** command within PowerShell to delete management groups.

```
Remove-AzureRmManagementGroup -GroupName Contoso
```

Delete in Azure CLI

With Azure CLI, use the command `az account management-group delete`.

```
az account management-group delete --name Contoso
```

View management groups

You can view any management group you have a direct or inherited RBAC role on.

View in the portal

1. Log into the [Azure portal](#)
2. Select **All services > Management groups**
3. The management group hierarchy page loads where you can explore all the management groups and subscriptions you have access to. Selecting the group name takes you down a level in the hierarchy. The navigation works the same as a file explorer does.

The screenshot shows the Azure Management Groups blade. At the top, there's a breadcrumb trail: Home > Management groups > contoso -. Below that is a search bar with placeholder text 'Search by name or ID'. To the right of the search bar is a help icon with the text: 'Using management groups helps you manage access, policy, and compliance by grouping multiple subscriptions together. [Learn more.](#)'

The main area displays a tree view of management groups under 'Root Management G... > Contoso Redmond'. The tree starts with 'Contoso Redmond' at the root, which branches into 'Azure Policy', 'Contoso IT', 'Contoso Marketing', 'Global', 'Storefront', 'Azure Test Sub', 'BillNotifications', 'Groups', 'Legacy Groups Classic', and 'mye2esubprod1-C'. Each item in the tree has a small icon to its left and a '(details)' link next to its name.

NAME	ID	TYPE	MY ROLE
Azure Policy	<MG ID>	Management Group	Owner
Contoso IT	<MG ID>	Management Group	Owner
Contoso Marketing	<MG ID>	Management Group	Owner
Global	<MG ID>	Management Group	Owner
Storefront	<MG ID>	Management Group	Owner
Azure Test Sub	<MG ID>	Subscription	Owner
BillNotifications	<MG ID>	Subscription	Owner
Groups	<MG ID>	Subscription	Owner
Legacy Groups Classic	<MG ID>	Subscription	Owner
mye2esubprod1-C	<MG ID>	Subscription	Owner

4. To see the details of the management group, select the **(details)** link next to the title of the management group. If this link isn't available, you don't have permissions to view that management group.

View in PowerShell

You use the `Get-AzureRmManagementGroup` command to retrieve all groups.

```
Get-AzureRmManagementGroup
```

For a single management group's information, use the `-GroupName` parameter

```
Get-AzureRmManagementGroup -GroupName Contoso
```

[View in Azure CLI](#)

You use the list command to retrieve all groups.

```
az account management-group list
```

For a single management group's information, use the show command

```
az account management-group show --name Contoso
```

Move subscriptions in the hierarchy

One reason to create a management group is to bundle subscriptions together. Only management groups and subscriptions can be made children of another management group. A subscription that moves to a management group inherits all user access and policies from the parent management group.

To move the subscription, there are a couple permissions you must have:

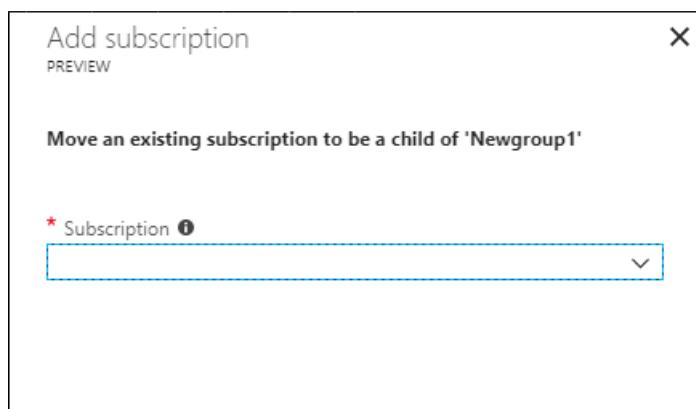
- "Owner" role on the child subscription.
- "Owner" or "Contributor" role on the new parent management group.
- "Owner" or "Contributor" role on the old parent management group.

To see what permissions you have, select the management group and then select **IAM**. To learn more on RBAC Roles, see [Manage access and permissions with RBAC](#).

Move subscriptions in the portal

Add an existing Subscription to a management group

1. Log into the [Azure portal](#)
2. Select **All services > Management groups**
3. Select the management group you're planning to be the parent.
4. At the top of the page, select **Add subscription**.
5. Select the subscription in the list with the correct ID.



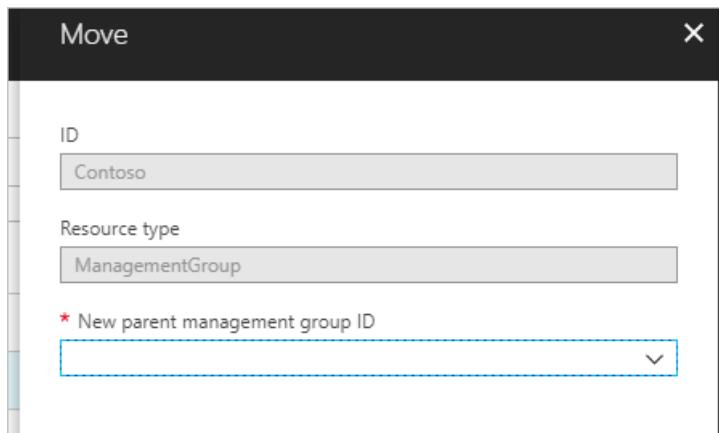
6. Select "Save"

Remove a subscription from a management group

1. Log into the [Azure portal](#)
2. Select **All services > Management groups**
3. Select the management group you're planning that is the current parent.
4. Select the ellipse at the end of the row for the subscription in the list you want to move.

TYPE	MY ROLE	
Management Group		...
Management Group	Move	→
Management Group	Owner	...
Management Group	Owner	...

5. Select **Move**
6. On the menu that opens, select the **Parent management group**.



7. Select **Save**

Move subscriptions in PowerShell

To move a subscription in PowerShell, you use the `Add-AzureRmManagementGroupSubscription` command.

```
New-AzureRmManagementGroupSubscription -GroupName Contoso -SubscriptionId 12345678-1234-1234-1234-123456789012
```

To remove the link between a subscription and the management group use the `Remove-AzureRmManagementGroupSubscription` command.

```
Remove-AzureRmManagementGroupSubscription -GroupName Contoso -SubscriptionId 12345678-1234-1234-1234-123456789012
```

Move subscriptions in Azure CLI

To move a subscription in CLI, you use the `az account management-group subscription add` command.

```
az account management-group subscription add --name Contoso --subscription 12345678-1234-1234-1234-123456789012
```

To remove the subscription from the management group, use the `az account management-group subscription remove` command.

```
az account management-group subscription remove --name Contoso --subscription 12345678-1234-1234-1234-123456789012
```

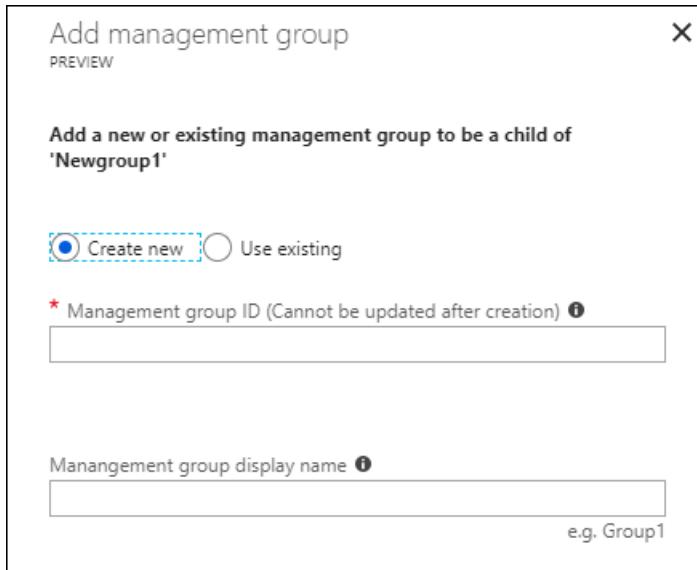
Move management groups in the hierarchy

When you move a parent management group, all the child resources that include management groups,

subscriptions, resource groups, and resources move with the parent.

Move management groups in the portal

1. Log into the [Azure portal](#)
2. Select **All services > Management groups**
3. Select the management group you're planning to be the parent.
4. At the top of the page, select **Add management group**.
5. In the menu that opens, select if you want a new or use an existing management group.
 - Selecting new will create a new management group.
 - Selecting an existing will present you with a drop-down of all the management groups you can move to this management group.



6. Select **Save**

Move management groups in PowerShell

Use the `Update-AzureRmManagementGroup` command in PowerShell to move a management group under a different group.

```
Update-AzureRmManagementGroup -GroupName Contoso -ParentName ContosoIT
```

Move management groups in Azure CLI

Use the `update` command to move a management group with Azure CLI.

```
az account management-group update --name Contoso --parent "Contoso Tenant"
```

Next steps

To Learn more about management groups, see:

- [Organize your resources with Azure management groups](#)
- [Create management groups to organize Azure resources](#)
- [Install the Azure Powershell module](#)
- [Review the REST API Spec](#)
- [Install the Azure CLI Extension](#)

Use Azure PowerShell to create a service principal with a certificate

8/3/2018 • 5 minutes to read • [Edit Online](#)

When you have an app or script that needs to access resources, you can set up an identity for the app and authenticate the app with its own credentials. This identity is known as a service principal. This approach enables you to:

- Assign permissions to the app identity that are different than your own permissions. Typically, these permissions are restricted to exactly what the app needs to do.
- Use a certificate for authentication when executing an unattended script.

IMPORTANT

Instead of creating a service principal, consider using Azure AD Managed Service Identity for your application identity. Azure AD MSI is a public preview feature of Azure Active Directory that simplifies creating an identity for code. If your code runs on a service that supports Azure AD MSI and accesses resources that support Azure Active Directory authentication, Azure AD MSI is a better option for you. To learn more about Azure AD MSI, including which services currently support it, see [Managed Service Identity for Azure resources](#).

This article shows you how to create a service principal that authenticates with a certificate. To set up a service principal with password, see [Create an Azure service principal with Azure PowerShell](#).

You must have the [latest version](#) of PowerShell for this article.

Required permissions

To complete this article, you must have sufficient permissions in both your Azure Active Directory and Azure subscription. Specifically, you must be able to create an app in the Azure Active Directory, and assign the service principal to a role.

The easiest way to check whether your account has adequate permissions is through the portal. See [Check required permission](#).

Create service principal with self-signed certificate

The following example covers a simple scenario. It uses [New-AzureRmADServicePrincipal](#) to create a service principal with a self-signed certificate, and uses [New-AzureRmRoleAssignment](#) to assign the [Contributor](#) role to the service principal. The role assignment is scoped to your currently selected Azure subscription. To select a different subscription, use [Set-AzureRmContext](#).

```

$cert = New-SelfSignedCertificate -CertStoreLocation "cert:\CurrentUser\My" ` 
-Subject "CN=exampleappScriptCert" ` 
-KeySpec KeyExchange
$keyValue = [System.Convert]::ToBase64String($cert.GetRawCertData())

$sp = New-AzureRMServicePrincipal -DisplayName exampleapp ` 
-CertValue $keyValue ` 
-EndDate $cert.NotAfter ` 
-StartDate $cert.NotBefore
Sleep 20
New-AzureRmRoleAssignment -RoleDefinitionName Contributor -ServicePrincipalName $sp.ApplicationId

```

The example sleeps for 20 seconds to allow some time for the new service principal to propagate throughout Azure Active Directory. If your script doesn't wait long enough, you'll see an error stating: "Principal {ID} does not exist in the directory {DIR-ID}." To resolve this error, wait a moment then run the **New-AzureRmRoleAssignment** command again.

You can scope the role assignment to a specific resource group by using the **ResourceGroupName** parameter. You can scope to a specific resource by also using the **ResourceType** and **ResourceName** parameters.

If you **do not have Windows 10 or Windows Server 2016**, you need to download the [Self-signed certificate generator](#) from Microsoft Script Center. Extract its contents and import the cmdlet you need.

```

# Only run if you could not use New-SelfSignedCertificate
Import-Module -Name c:\ExtractedModule\New-SelfSignedCertificateEx.ps1

```

In the script, substitute the following two lines to generate the certificate.

```

New-SelfSignedCertificateEx -StoreLocation CurrentUser ` 
-Subject "CN=exampleapp" ` 
-KeySpec "Exchange" ` 
-FriendlyName "exampleapp"
$cert = Get-ChildItem -path Cert:\CurrentUser\my | where {$PSitem.Subject -eq 'CN=exampleapp'}

```

Provide certificate through automated PowerShell script

Whenever you sign in as a service principal, you need to provide the tenant ID of the directory for your AD app. A tenant is an instance of Azure Active Directory.

```

$TenantId = (Get-AzureRmSubscription -SubscriptionName "Contoso Default").TenantId
$ApplicationId = (Get-AzureRmADApplication -DisplayNameStartWith exampleapp).ApplicationId

$Thumbprint = (Get-ChildItem cert:\CurrentUser\My\ | Where-Object {$_.Subject -match "CN=exampleappScriptCert"}).Thumbprint
Connect-AzureRmAccount -ServicePrincipal ` 
-CertificateThumbprint $Thumbprint ` 
-ApplicationId $ApplicationId ` 
-TenantId $TenantId

```

Create service principal with certificate from Certificate Authority

The following example uses a certificate issued from a Certificate Authority to create service principal. The assignment is scoped to the specified Azure subscription. It adds the service principal to the [Contributor](#) role. If an error occurs during the role assignment, it retries the assignment.

```

Param (
    [Parameter(Mandatory=$true)]
    [String] $ApplicationDisplayName,
    [Parameter(Mandatory=$true)]
    [String] $SubscriptionId,
    [Parameter(Mandatory=$true)]
    [String] $CertPath,
    [Parameter(Mandatory=$true)]
    [String] $CertPlainPassword
)

Connect-AzureRmAccount
Import-Module AzureRM.Resources
Set-AzureRmContext -Subscription $SubscriptionId

$CertPassword = ConvertTo-SecureString $CertPlainPassword -AsPlainText -Force

$PFXCert = New-Object -TypeName System.Security.Cryptography.X509Certificates.X509Certificate2 -ArgumentList
@($CertPath, $CertPassword)
$keyValue = [System.Convert]::ToBase64String($PFXCert.GetRawCertData())

$ServicePrincipal = New-AzureRmADServicePrincipal -DisplayName $ApplicationDisplayName
New-AzureRmADSpCredential -ObjectId $ServicePrincipal.Id -CertValue $keyValue -StartDate $PFXCert.NotBefore -EndDate $PFXCert.NotAfter
Get-AzureRmADServicePrincipal -ObjectId $ServicePrincipal.Id

$NewRole = $null
$Retries = 0;
While ($NewRole -eq $null -and $Retries -le 6)
{
    # Sleep here for a few seconds to allow the service principal application to become active (should only take a couple of seconds normally)
    Sleep 15
    New-AzureRMRoleAssignment -RoleDefinitionName Contributor -ServicePrincipalName
    $ServicePrincipal.ApplicationId | Write-Verbose -ErrorAction SilentlyContinue
    $NewRole = Get-AzureRMRoleAssignment -ObjectId $ServicePrincipal.Id -ErrorAction SilentlyContinue
    $Retries++;
}

$NewRole

```

Provide certificate through automated PowerShell script

Whenever you sign in as a service principal, you need to provide the tenant ID of the directory for your AD app. A tenant is an instance of Azure Active Directory.

```

Param (
    [Parameter(Mandatory=$true)]
    [String] $CertPath,
    [Parameter(Mandatory=$true)]
    [String] $CertPlainPassword,
    [Parameter(Mandatory=$true)]
    [String] $ApplicationId,
    [Parameter(Mandatory=$true)]
    [String] $TenantId
)

$CertPassword = ConvertTo-SecureString $CertPlainPassword -AsPlainText -Force
$PFXCert = New-Object `-
    -TypeName System.Security.Cryptography.X509Certificates.X509Certificate2 `-
    -ArgumentList @($CertPath, $CertPassword)
$Thumbprint = $PFXCert.Thumbprint

Connect-AzureRmAccount -ServicePrincipal `-
    -CertificateThumbprint $Thumbprint `-
    -ApplicationId $ApplicationId `-
    -TenantId $TenantId

```

The application ID and tenant ID aren't sensitive, so you can embed them directly in your script. If you need to retrieve the tenant ID, use:

```
(Get-AzureRmSubscription -SubscriptionName "Contoso Default").TenantId
```

If you need to retrieve the application ID, use:

```
(Get-AzureRmADApplication -DisplayNameStartWith {display-name}).ApplicationId
```

Change credentials

To change the credentials for an AD app, either because of a security compromise or a credential expiration, use the [Remove-AzureRmADAppCredential](#) and [New-AzureRmADAppCredential](#) cmdlets.

To remove all the credentials for an application, use:

```
Get-AzureRmADApplication -DisplayName exampleapp | Remove-AzureRmADAppCredential
```

To add a certificate value, create a self-signed certificate as shown in this article. Then, use:

```
Get-AzureRmADApplication -DisplayName exampleapp | New-AzureRmADAppCredential `-
    -CertValue $keyValue `-
    -EndDate $cert.NotAfter `-
    -StartDate $cert.NotBefore
```

Debug

You may get the following errors when creating a service principal:

- **"Authentication_Unauthorized"** or **"No subscription found in the context."** - You see this error when

your account does not have the [required permissions](#) on the Azure Active Directory to register an app. Typically, you see this error when only admin users in your Azure Active Directory can register apps, and your account is not an admin. Ask your administrator to either assign you to an administrator role, or to enable users to register apps.

- Your account "**does not have authorization to perform action** '`Microsoft.Authorization/roleAssignments/write`' over scope '/subscriptions/{guid}'." - You see this error when your account does not have sufficient permissions to assign a role to an identity. Ask your subscription administrator to add you to User Access Administrator role.

Next steps

- To set up a service principal with password, see [Create an Azure service principal with Azure PowerShell](#).
- For detailed steps on integrating an application into Azure for managing resources, see [Developer's guide to authorization with the Azure Resource Manager API](#).
- For a more detailed explanation of applications and service principals, see [Application Objects and Service Principal Objects](#).
- For more information about Azure Active Directory authentication, see [Authentication Scenarios for Azure AD](#).

Use portal to create an Azure Active Directory application and service principal that can access resources

7/26/2018 • 5 minutes to read • [Edit Online](#)

When you have code that needs to access or modify resources, you must set up an Azure Active Directory (AD) application. You can then assign the required permissions to the AD application. This approach is preferable to running the app under your own credentials because you can assign permissions to the app identity that are different than your own permissions. Typically, these permissions are restricted to exactly what the app needs to do.

This article shows you how to perform these steps through the portal. It focuses on a single-tenant application where the application is intended to run within only one organization. You typically use single-tenant applications for line-of-business applications that run within your organization.

IMPORTANT

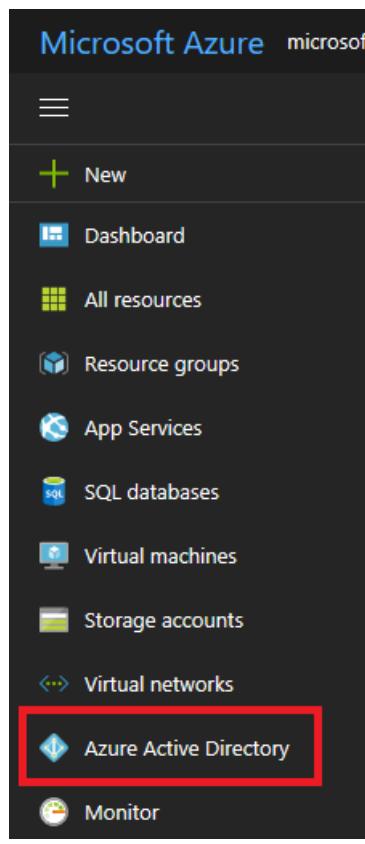
Instead of creating a service principal, consider using Azure AD Managed Service Identity for your application identity. Azure AD MSI is a public preview feature of Azure Active Directory that simplifies creating an identity for code. If your code runs on a service that supports Azure AD MSI and accesses resources that support Azure Active Directory authentication, Azure AD MSI is a better option for you. To learn more about Azure AD MSI, including which services currently support it, see [Managed Service Identity for Azure resources](#).

Required permissions

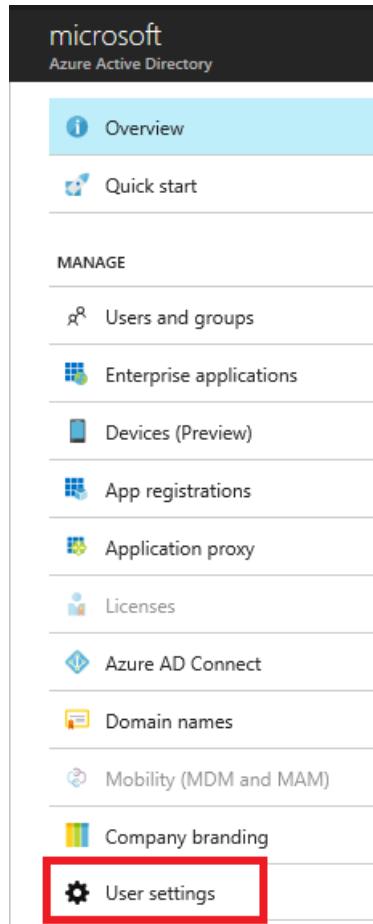
To complete this article, you must have sufficient permissions to register an application with your Azure AD tenant, and assign the application to a role in your Azure subscription. Let's make sure you have the right permissions to perform those steps.

Check Azure Active Directory permissions

1. Select **Azure Active Directory**.



2. In Azure Active Directory, select **User settings**.



3. Check the **App registrations** setting. If set to **Yes**, non-admin users can register AD apps. This setting means any user in the Azure AD tenant can register an app. You can proceed to [Check Azure subscription permissions](#).

Enterprise applications

Users can consent to apps accessing company data on their behalf Yes No

Users can add gallery apps to their Access Panel Yes No

App registrations

Users can register applications Yes No

External users

Guest users permissions are limited Yes No

Admins and users in the guest inviter role can invite Yes No

Members can invite Yes No

Guests can invite Yes No

- If the app registrations setting is set to **No**, only **global administrators** can register apps. Check whether your account is an admin for the Azure AD tenant. Select **Overview** and look at your user information. If your account is assigned to the User role, but the app registration setting (from the preceding step) is limited to admin users, ask your administrator to either assign you to the global administrator role, or to enable users to register apps.

Overview Switch directory Delete directory

microsoft.onmicrosoft.com
microsoft
Azure AD for Office 365

Sign-ins

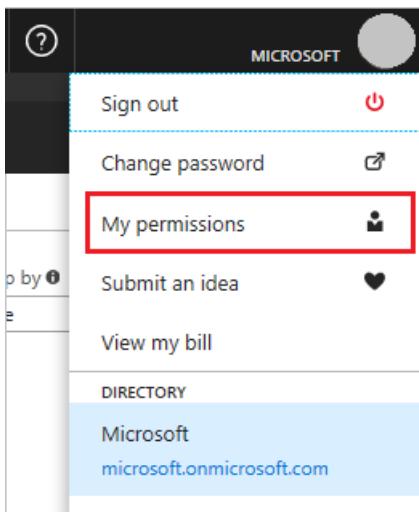
Your role
User More info >

Check Azure subscription permissions

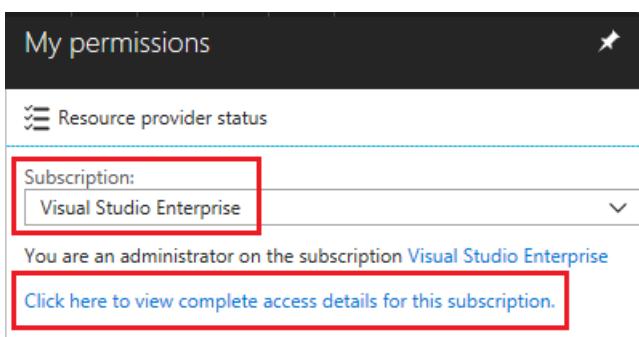
In your Azure subscription, your account must have `Microsoft.Authorization/*/Write` access to assign an AD app to a role. This action is granted through the **Owner** role or **User Access Administrator** role. If your account is assigned to the **Contributor** role, you do not have adequate permission. You receive an error when attempting to assign the service principal to a role.

To check your subscription permissions:

- Select your account in the upper right corner, and select **My permissions**.



- From the drop-down list, select the subscription. Select **Click here to view complete access details for this subscription**.

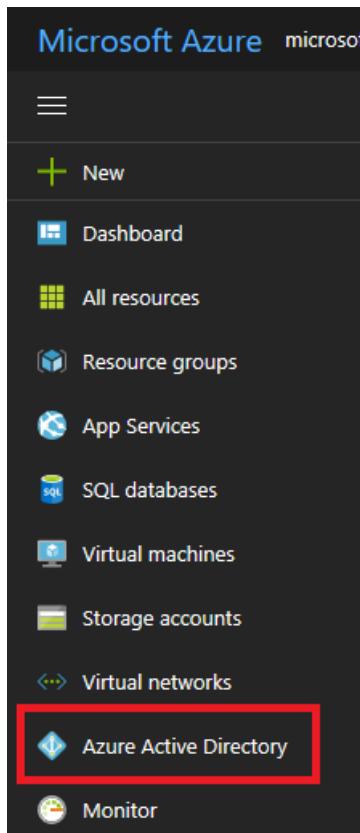


- View your assigned roles, and determine if you have adequate permissions to assign an AD app to a role. If not, ask your subscription administrator to add you to User Access Administrator role. In the following image, the user is assigned to the Owner role, which means that user has adequate permissions.

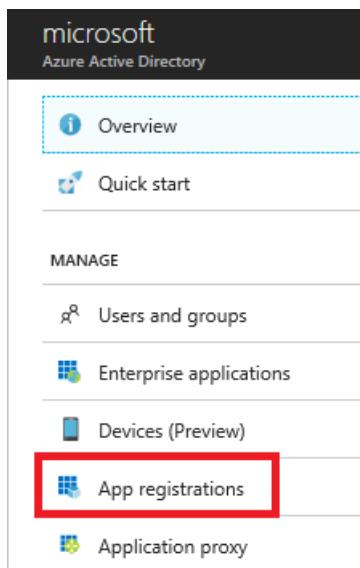
+ Add				Remove		Roles	Refresh	Help
Name	Type	Role	Scope					
Search by name or email	All	2 selected	All scopes					
2 items (1 Users, 1 Service Principals)								
NAME	TYPE	ROLE	SCOPE					
OWNER								
<input checked="" type="checkbox"/> Example User example@contoso.org	User	Owner, Service administrator	All scopes					

Create an Azure Active Directory application

- Log in to your Azure Account through the [Azure portal](#).
- Select **Azure Active Directory**.



3. Select **App registrations**.



4. Select **New application registration**.



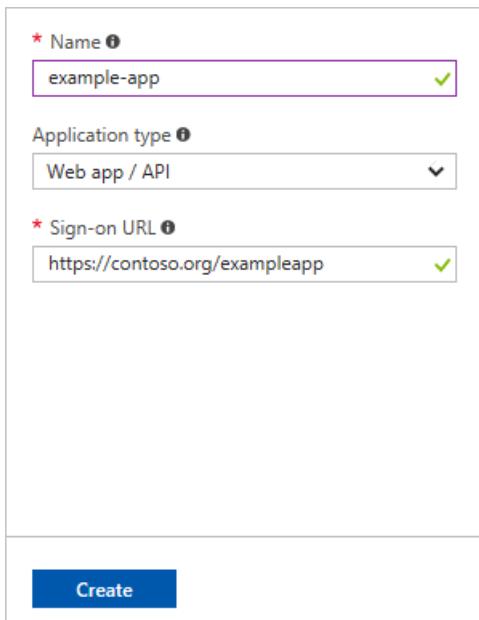
5. Provide a name and URL for the application. Select **Web app / API** for the type of application you want to create. You cannot create credentials for a **Native application**; therefore, that type does not work for an automated application. After setting the values, select **Create**.

*** Name** example-app ✓

Application type Web app / API

*** Sign-on URL** https://contoso.org/exampleapp ✓

Create



You have created your application.

Get application ID and authentication key

When programmatically logging in, you need the ID for your application and an authentication key. To get those values, use the following steps:

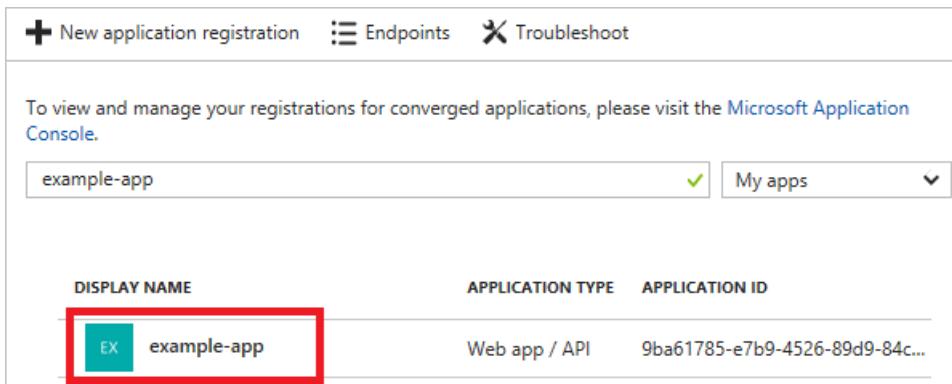
1. From **App registrations** in Azure Active Directory, select your application.

+ New application registration **Endpoints** **Troubleshoot**

To view and manage your registrations for converged applications, please visit the [Microsoft Application Console](#).

example-app ✓ My apps

DISPLAY NAME	APPLICATION TYPE	APPLICATION ID
EX example-app	Web app / API	9ba61785-e7b9-4526-89d9-84c41125cf20...



2. Copy the **Application ID** and store it in your application code. Some [sample applications](#) refer to this value as the client ID.

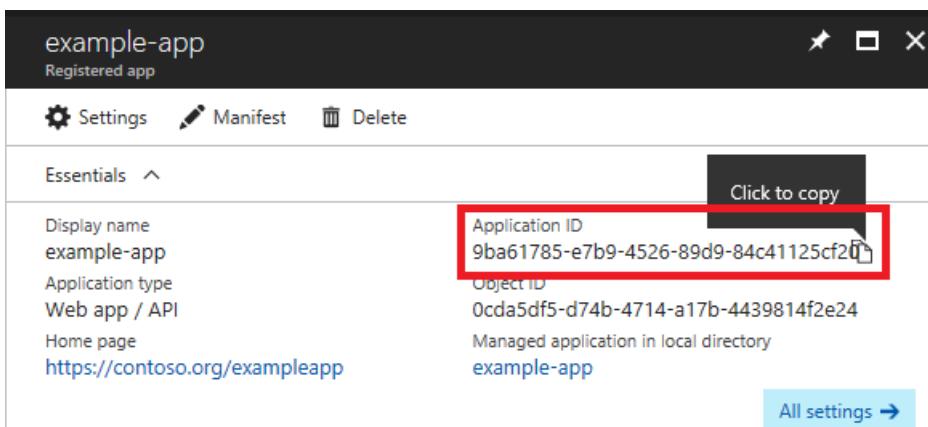
example-app
Registered app

Settings **Manifest** **Delete**

Essentials **Click to copy**

Display name	Application ID
example-app	9ba61785-e7b9-4526-89d9-84c41125cf20
Application type	Object ID
Web app / API	0cda5df5-d74b-4714-a17b-4439814f2e24
Home page	Managed application in local directory
https://contoso.org/exampleapp	example-app

All settings →



3. To generate an authentication key, select **Settings**.

The screenshot shows the Azure portal interface for a registered app named 'exampleapp'. At the top, there's a dark header with the app name and 'Registered app' status. Below it is a navigation bar with three items: 'Settings' (highlighted with a red box), 'Manifest', and 'Delete'. The main content area is titled 'Settings'.

4. To generate an authentication key, select **Keys**.

This screenshot shows the 'Settings' blade for the 'exampleapp' app. Under the 'GENERAL' section, there are links for 'Properties', 'Reply URLs', and 'Owners'. In the 'API ACCESS' section, there are links for 'Required permissions' and 'Keys'. The 'Keys' link is highlighted with a red box.

5. Provide a description of the key, and a duration for the key. When done, select **Save**.

This screenshot shows the 'Keys' blade. At the top, there are 'Save' and 'Discard' buttons. Below them is a table with columns: DESCRIPTION, EXPIRES, and VALUE. A new row is being added, with 'first key' in the DESCRIPTION field and 'In 1 year' in the EXPIRES field. The entire row is highlighted with a red box. A tooltip 'Value will be displayed on save' is visible over the VALUE column.

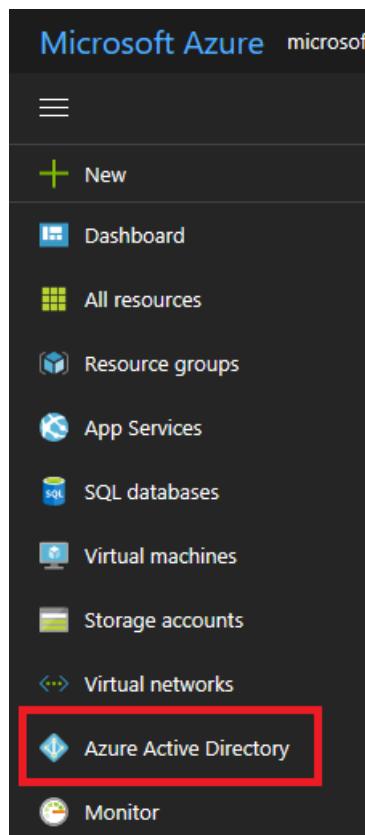
After saving the key, the value of the key is displayed. Copy this value because you are not able to retrieve the key later. You provide the key value with the application ID to log in as the application. Store the key value where your application can retrieve it.

This screenshot shows the 'Keys' blade after saving the key. A yellow warning bar at the top says '⚠️ Copy the key value. You won't be able to retrieve after you leave this blade.' Below it is a table with the same columns: DESCRIPTION, EXPIRES, and VALUE. The first row now shows 'first key' in the DESCRIPTION column, '9/8/2018' in the EXPIRES column, and the key value 'syH8cFAWotOvXIZPQXIVhQhkyNeWDCW8rXHaIYWDsvs=' in the VALUE column. This value is also highlighted with a red box.

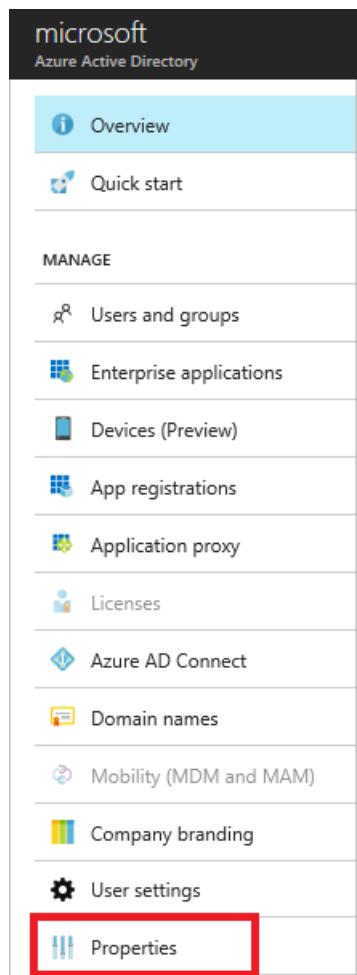
Get tenant ID

When programmatically logging in, you need to pass the tenant ID with your authentication request.

1. Select **Azure Active Directory**.



2. To get the tenant ID, select **Properties** for your Azure AD tenant.



3. Copy the **Directory ID**. This value is your tenant ID.

Save Discard

* Name
Microsoft

Country or region
United States

Location
Asia, United States, Europe datacenters

Notification language
English

Global admin can manage Azure Subscriptions

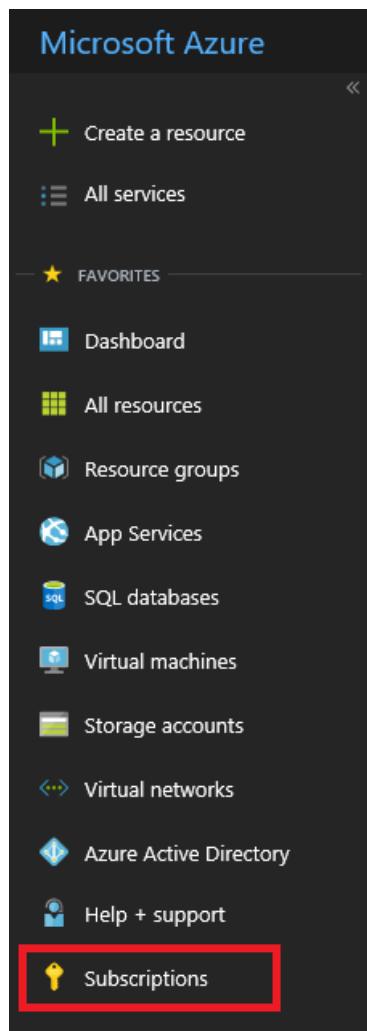
Directory ID
xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx 

Assign application to role

To access resources in your subscription, you must assign the application to a role. Decide which role represents the right permissions for the application. To learn about the available roles, see [RBAC: Built in Roles](#).

You can set the scope at the level of the subscription, resource group, or resource. Permissions are inherited to lower levels of scope. For example, adding an application to the Reader role for a resource group means it can read the resource group and any resources it contains.

1. Navigate to the level of scope you wish to assign the application to. For example, to assign a role at the subscription scope, select **Subscriptions**. You could instead select a resource group or resource.



2. Select the particular subscription (resource group or resource) to assign the application to.

This screenshot shows the 'Subscriptions' page in the Azure portal. At the top left is a 'Microsoft' logo. Below it is a 'Subscriptions' header with a 'Add' button. Under 'My role' is a dropdown menu set to 'All' with an 'Apply' button. A search bar below it contains the placeholder 'Search to filter items...'. The main area is titled 'SUBSCRIPTION' and lists one item: 'Visual Studio Enterprise' with a blue gear icon. This item is also highlighted with a red rectangular border.

3. Select **Access Control (IAM)**.

This screenshot shows the 'Subscription Overview' page for 'Visual Studio Enterprise'. At the top left is a yellow key icon. The title 'Visual Studio Enterprise Subscription' is followed by a 'Search (Ctrl+ /)' input field. Below the search bar are three navigation links: 'Overview' (highlighted with a red box), 'Access control (IAM)', and 'Diagnose and solve problems'.

4. Select **Add**.

The screenshot shows the 'Access control (IAM)' section of the Visual Studio Enterprise dashboard. On the left, there's a navigation bar with 'Overview', 'Access control (IAM)', and 'Diagnose and solve problems'. The 'Access control (IAM)' item is selected and highlighted with a blue dashed border. On the right, there's a search bar labeled 'Search (Ctrl+ /)' and a row of buttons: '+ Add' (highlighted with a red box), 'Remove', and 'Roles'. Below these are fields for 'Name' (with placeholder 'Search by name or email') and 'Scope' (set to 'All scopes').

5. Select the role you wish to assign to the application. In order to allow the application execute actions like **reboot**, **start** and **stop** instances, you must have to select the role **Contributor**. The following image shows the **Reader** role.

The screenshot shows the 'Add permissions' dialog. It has two main sections: 'Role' and 'Select'. The 'Role' section contains a dropdown menu where 'Reader' is selected. The 'Select' section contains a search bar with 'example-app' typed into it, and below it, a list of results where 'example-app' is shown again, with its icon and name highlighted with a red box.

6. By default, Azure Active Directory applications aren't displayed in the available options. To find your application, you must provide the name of it in the search field. Select it.

The screenshot shows the 'Add permissions' dialog with the 'Select' section expanded. The search bar contains 'example-app'. Below it, a list of applications is shown, with 'example-app' listed and its icon and name highlighted with a red box.

7. Select **Save** to finish assigning the role. You see your application in the list of users assigned to a role for that scope.

Next steps

- To set up a multi-tenant application, see [Developer's guide to authorization with the Azure Resource Manager API](#).
- To learn about specifying security policies, see [Azure Role-based Access Control](#).
- For a list of available actions that can be granted or denied to users, see [Azure Resource Manager Resource Provider operations](#).

Use Resource Manager authentication API to access subscriptions

8/2/2018 • 14 minutes to read • [Edit Online](#)

Introduction

If you are 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 on behalf of 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.

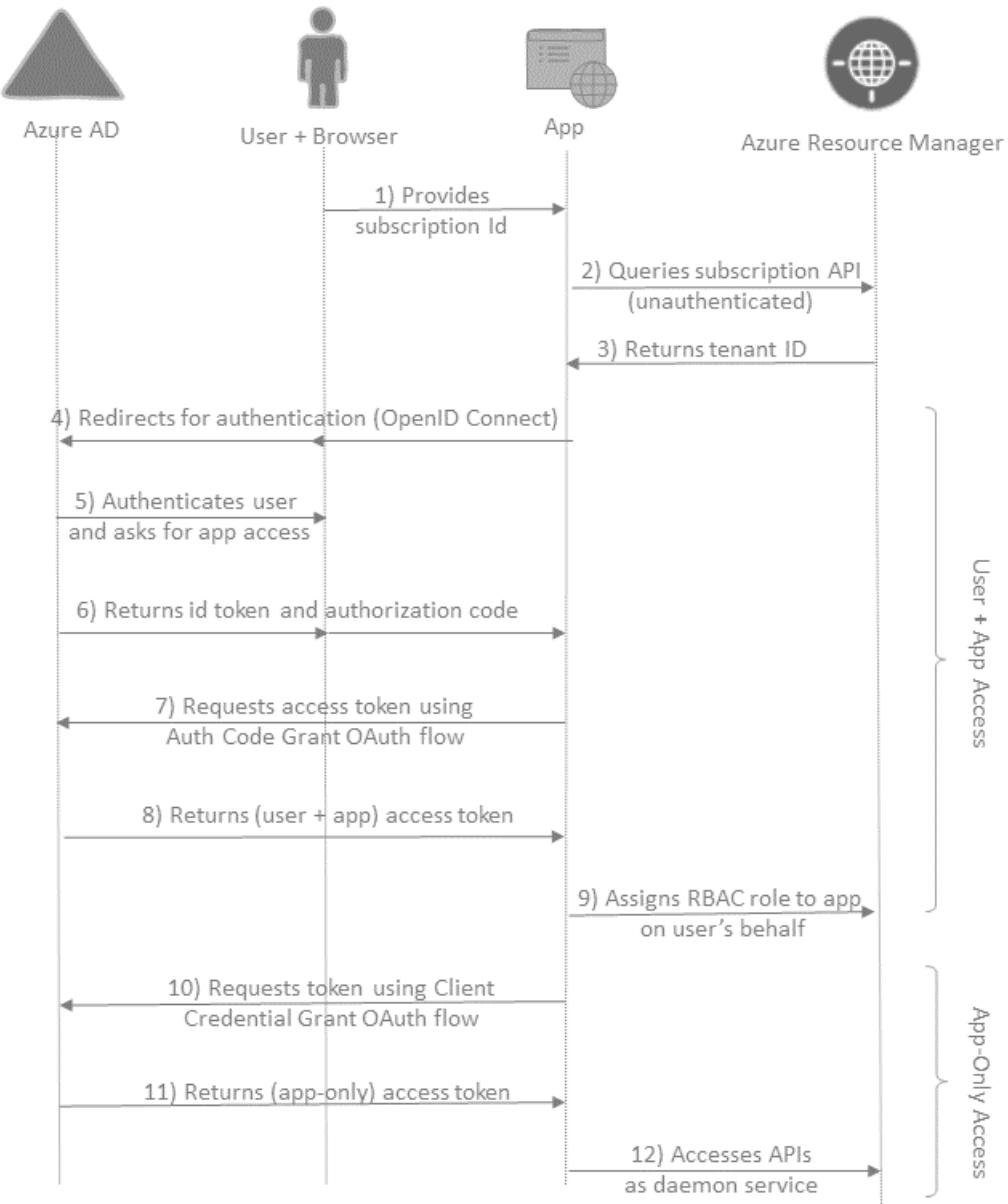
This article provides step-by-step instructions to create an app that employs both these authorization methods. It shows how to perform each step with REST API or C#. The complete ASP.NET MVC application is available at <https://github.com/dushyantgill/VipSwapper/tree/master/CloudSense>.

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.



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) [?](#)

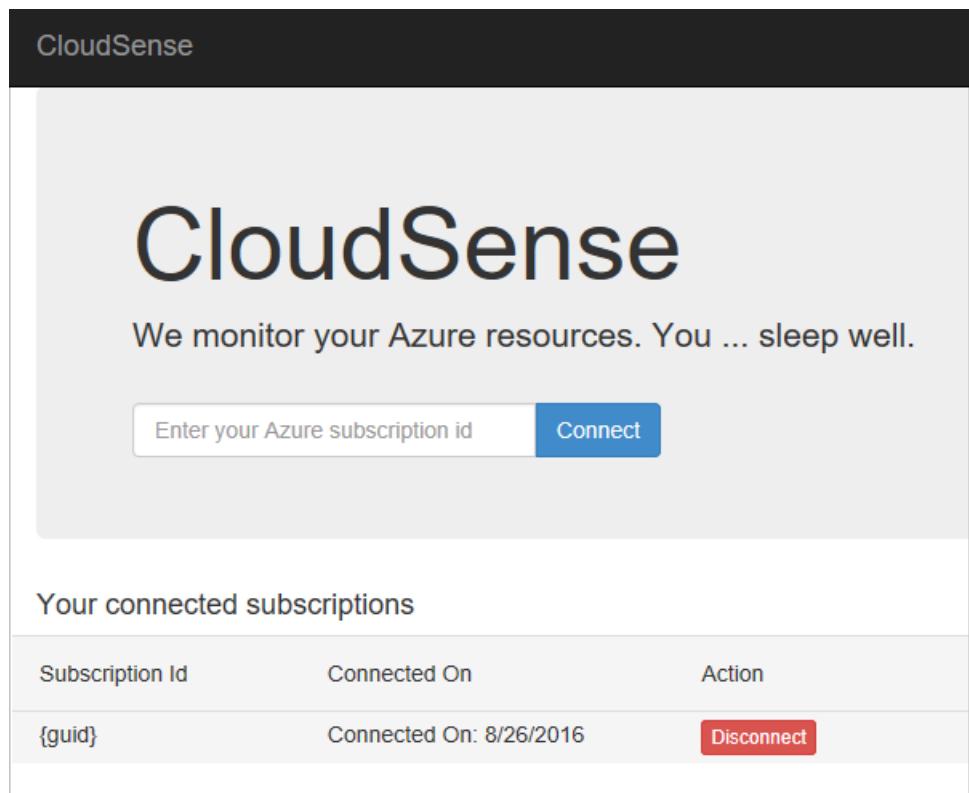
You're signed in as: example@contoso.org

[Show details](#)

Accept

Cancel

Manage your connected subscriptions:



The screenshot shows the CloudSense monitoring portal. At the top, it says "CloudSense". Below that is a large "CloudSense" logo with the tagline "We monitor your Azure resources. You ... sleep well." A text input field contains "Enter your Azure subscription id" and a blue "Connect" button. Below this is a section titled "Your connected subscriptions" with a table. The table has columns: "Subscription Id", "Connected On", and "Action". It shows one entry: "{guid}" connected on 8/26/2016, with a red "Disconnect" button.

Subscription Id	Connected On	Action
{guid}	Connected On: 8/26/2016	<button>Disconnect</button>

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 on behalf of the user.

Because your app accesses other subscription, you must configure it as a multi-tenant application. To pass validation, provide a domain associated with your Azure Active Directory. To see the domains associated with your Azure Active Directory, sign in to the portal.

The following example shows how to register the app by using Azure PowerShell. You must have the latest version (August 2016) of Azure PowerShell for this command to work.

```
$app = New-AzureRmADApplication -DisplayName "{app name}" -HomePage "https://your domain}/{app name}" -IdentifierUris "https://your domain}/{app name}" -Password "{your password}" -AvailableToOtherTenants $true
```

To sign in as the AD application, you need the application ID and password. To see the application ID that is returned from the previous command, use:

```
$app.ApplicationId
```

The following example shows how to register the app by using Azure CLI.

```
az ad app create --display-name {app name} --homepage https://your domain}/{app name} --identifier-uris https://your domain}/{app name} --password {your password} --available-to-other-tenants true
```

The results include the AppId, which you need when authenticating as the application.

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 has not 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 perform authentication 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-c1873de9394&response_mode=query&response_type=code&redirect_uri=http%3a%2f%2fwww.vipswapper.com%2fcardsense%2fAccount%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 on behalf of 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%2fccloudsense%2fAccount%2fSignIn&resource=https%3a%2f%2fgraph.windows.net%2f&scope=openid+profile&nonce=63567Dc4MDAw&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 is 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%3a62080%2fAccount%2fSignIn&client_id=a0448380-c346-4f9f-b897-c18733de9394&client_secret=o1na84E8****goSc0g%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. The claim types for the token are shown in [JWT token claims](#). 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%3A62080%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": "https://management.core.windows.net/", "access_token": "eyJ0eXAiOiJKV1Q****M7Cw6JWtfY2lGc5A", "refresh_token": "AAABAAAiL9Kn2Z****55j-sjnyYgAA", "scope": "user_impersonation", "id_token": "eyJ0eXAiOiJKV1Q****-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 on behalf of the user. The lifetime of access tokens issued by Azure AD is one hour. It is 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=oIa84E8****goSc0g%3D
```

Although refresh tokens can be used to get new access tokens for Azure Resource Manager, they are not 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 on behalf of 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. 83cf939-2402-4581-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 eyJ0eXAiOiJKV1QiLCJhbGwiOiJzdWIiLCJ1c2VyX2lkIjoiMjAxNzEwOTUyIiwidmFsdWUi  
On: 2015-07-01T12:00:00Z  
Expires: 2015-07-01T12:00:00Z  
Type: access_token
```

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": []}]}  
  
Content-Type: application/json; charset=utf-8
```

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 perform 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*****0g%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": "https://graph.windows.net/", "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Ik1uQ19WWmNBVGZNNXBPWw1KSE1iYTlnb0VLWSIsImtpZCI6Ik1uQ19WWmNBVGZNNXBPWw1KSE1iYTlnb0VLWSJ9.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": "9b5018d4-6951-42ed-8a92-f11ec283ccce", "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 search then iterate over the result to find the desired 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 eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adGwuDw
```

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/resources/read",
              "Microsoft.Resources/subscriptions/resourceGroups/read",
              "Microsoft.Resources/subscriptions/resourceGroups/resources/read",
              "Microsoft.Resources/subscriptions/resourceGroups/deployments/*",
              "Microsoft.Insights/alertRules/*",
              "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.Resources/subscriptions/resourceGroups/resources/read",
              "Microsoft.Resources/subscriptions/resourceGroups/deploymentS/*",
              "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 do not 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

ROLE	GUID
Contributor	b24988ac-6180-42a0-ab88-20f7382dd24c
Virtual Machine Contributor	d73bb868-a0df-4d4d-bd69-98a00b01fcdb
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*****Flw01mM7Cw6JWtfY2lGc5
Content-Type: application/json
Content-Length: 230

{"properties": {"roleDefinitionId":"/subscriptions/09cbd307-aa71-4aca-b346-5f253e6e3ebb/providers/Microsoft.Authorization/roleDefinitions/acdd72a7-3385-48ef-bd42-f606fba81ae7","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
acdd72a7-3385-48ef-bd42-f606fba81ae7	the ID of the reader 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/acdd72a7-3385-48ef-bd42-f606fba81ae7","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 has the desired access on the subscription, perform 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 has the desired access on 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 are able to 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.

Lock resources to prevent unexpected changes

8/2/2018 • 5 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.

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 do not restrict how resources perform their own functions. Resource changes are restricted, but resource operations are not restricted. For example, a **ReadOnly** lock on a SQL Database prevents you from deleting or modifying the database, but it does not prevent you from creating, updating, or deleting data in the database. Data transactions are permitted because those operations are not sent to <https://management.azure.com>.

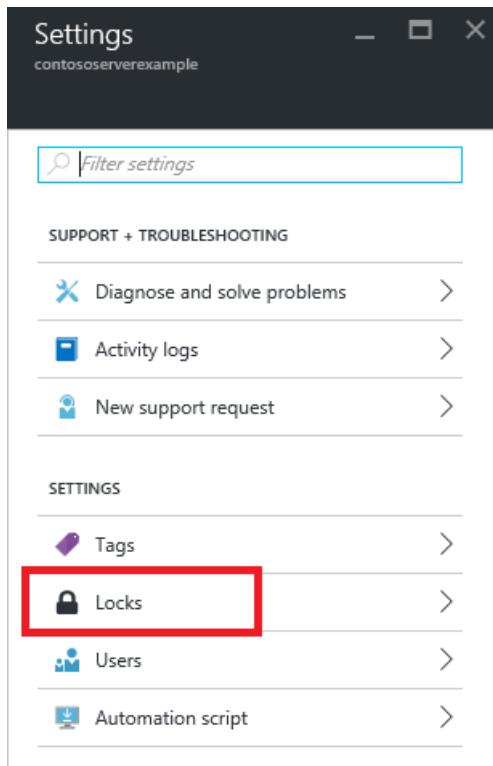
Applying **ReadOnly** can lead to unexpected results because some operations that seem like read operations actually require additional actions. For example, placing a **ReadOnly** lock on a storage account prevents all users from listing the keys. The list keys operation is handled through a POST request because the returned keys are available for write operations. For another example, placing 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.

Who can create or delete locks in your organization

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.

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.

The screenshot shows the 'Management locks' page. At the top, there are buttons for '+ Add', 'Resource group', 'Subscription', and 'Refresh'. Below that is a table header with columns: LOCK NAME, LOCK TYPE, SCOPE, and NOTES. A message below the table says 'This resource has no locks.'

3. Give the lock a name and lock level. Optionally, you can add notes that describe the lock.

The screenshot shows the 'Add lock' dialog box. It has fields for 'Lock name' (containing 'DatabaseServerLock') and 'Lock type' (set to 'Delete'). There's also a 'Notes' section with the text 'Prevent deleting the database server'. At the bottom are 'OK' and 'Cancel' buttons.

4. To delete the lock, select the ellipsis and **Delete** from the available options.

The screenshot shows the 'Management locks' blade in the Azure portal. The title bar includes the text 'Management locks' and 'contososerverexample'. The top navigation bar has buttons for 'Add', 'Resource group', 'Subscription', and 'Refresh'. Below the navigation is a table header with columns: 'LOCK NAME', 'LOCK TYPE', 'SCOPE', and 'NOTES'. A single row is visible in the table, representing a lock named 'DatabaseS...'. The 'NOTES' column contains the text 'Prevent deleting the database server'. To the right of the notes, there is a blue button with three dots ('...'), which is highlighted with a red box.

LOCK NAME	LOCK TYPE	SCOPE	NOTES
DatabaseS...	Delete	This resource	Prevent deleting the database server

Template

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.

```
{
    "$schema": "http://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."
            }
        }
    ]
}
}
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroup -Name sitegroup -Location southcentralus
New-AzureRmResourceGroupDeployment -ResourceGroupName sitegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-manager/lock.json -
hostingPlanName plan0103
```

To deploy this example template with Azure CLI, use:

```
az group create --name sitegroup --location southcentralus
az group deployment create --resource-group sitegroup --template-uri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-manager/lock.json --
parameters hostingPlanName=plan0103
```

PowerShell

You lock deployed resources with Azure PowerShell by using the [New-AzureRmResourceLock](#) command.

To lock a resource, provide the name of the resource, its resource type, and its resource group name.

```
New-AzureRmResourceLock -LockLevel CanNotDelete -LockName LockSite -ResourceName examplesite -ResourceType
Microsoft.Web/sites -ResourceGroupName exempleresourcegroup
```

To lock a resource group, provide the name of the resource group.

```
New-AzureRmResourceLock -LockName LockGroup -LockLevel CanNotDelete -ResourceGroupName exempleresourcegroup
```

To get information about a lock, use [Get-AzureRmResourceLock](#). To get all the locks in your subscription, use:

```
Get-AzureRmResourceLock
```

To get all locks for a resource, use:

```
Get-AzureRmResourceLock -ResourceName examplesite -ResourceType Microsoft.Web/sites -ResourceGroupName
exempleresourcegroup
```

To get all locks for a resource group, use:

```
Get-AzureRmResourceLock -ResourceGroupName exempleresourcegroup
```

To delete a lock, use:

```
$lockId = (Get-AzureRmResourceLock -ResourceGroupName exempleresourcegroup -ResourceName examplesite -
 ResourceType Microsoft.Web/sites).LockId
Remove-AzureRmResourceLock -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 exempleresourcegroup --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 exempleresourcegroup
```

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 **2015-01-01**.

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](#)
- To change which resource group a resource resides in, see [Move resources to new resource group](#)
- 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](#).

View activity logs to audit actions on resources

5/21/2018 • 3 minutes to read • [Edit Online](#)

Through activity logs, you can determine:

- what operations were taken on the resources in your subscription
- who initiated the operation (although operations initiated by a backend service do not return a user as the caller)
- when the operation occurred
- the status of the operation
- the values of other properties that might help you research the operation

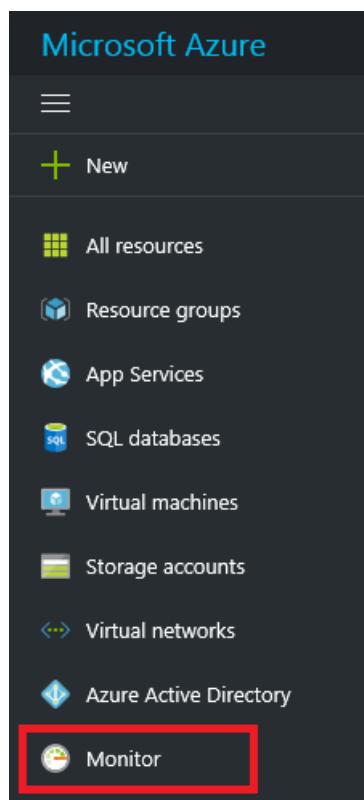
The activity log contains all write operations (PUT, POST, DELETE) performed on your resources. It does not include read operations (GET). For a list of resource actions, see [Azure Resource Manager Resource Provider operations](#). You can use the audit logs to find an error when troubleshooting or to monitor how a user in your organization modified a resource.

Activity logs are retained for 90 days. You can query for any range of dates, as long as the starting date is not 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](#).

Portal

1. To view the activity logs through the portal, select **Monitor**.



Or, to automatically filter the activity log for a particular resource or resource group, select **Activity log**. Notice that the activity log is automatically filtered by the selected resource.

Storage account

Activity log

Search (Ctrl+ /)

Columns Export

Select query ...

* Subscription: Windows Azure MSDN

Resource group: demo-group

Resource: storagedemo

Timespan: Last 1 hour

Event category: All categories

Event severity: 4 selected

Apply Reset

Insights (Last 24 hours): 1 failed dep, 1 outage notifications

2. In the **Activity Log**, you see a summary of recent operations.

Activity log

Columns Export

Select query ...

* Subscription: Windows Azure MSDN - Visual...

Resource group: All resource groups

Resource: All resources

Timespan: Last 1 hour

Event category: All categories

Event severity: 4 selected

Apply Reset

Insights (Last 24 hours): 5 items

Query returned 5 items. Click here to download all the items as csv.

OPERATION NAME	STATUS	TIME	TIME STAMP
▶ ❌ Write Deployments	Failed	Just now	Mon Aug 22...
ℹ️ Validate	Succeeded	Just now	Mon Aug 22...
ℹ️ Register	Succeeded	3 min ago	Mon Aug 22...
▶ ℹ️ Delete resource group	Succeeded	3 min ago	Mon Aug 22...
ℹ️ Register	Succeeded	3 min ago	Mon Aug 22...

3. To restrict the number of operations displayed, select different conditions. For example, the following image shows the **Timespan** and **Event initiated by** fields changed to view the actions taken by a particular user or application for the past month. Select **Apply** to view the results of your query.

Activity log

Columns Export

Select query ...

* Subscription: Windows Azure MSDN

Resource group: All resource groups

Resource: All resources

Resource type: All resource types

Timespan: Last month

Event category: All categories

Event severity: 4 selected

Event initiated by: CloudSense

Apply Reset

Insights (Last 24 hours): 1 failed dep

4. If you need to run the query again later, select **Save** and give the query a name.

Activity log

Columns Export

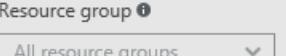
Select query ...  

5. To quickly run a query, you can select one of the built-in queries, such as failed deployments.

Insights (Last 24 hours): **1 failed deployment** | 0 role assignments | 1 error

The selected query automatically sets the required filter values.

Select query ...    Insights (Last 24 hours): **1 failed deployment** | 0 role assignments |

* Subscription  Resource group  Resource  Resource type 

2 selected All resource groups All resources Deployment (deployments)

Timespan  Event category  * Event severity  Event initiated by 

Last 24 hours All categories Error All

Apply Reset

Query returned 1 items. [Click here to download all the items as csv.](#)

OPERATION NAME	STATUS	TIME	TIME STAMP	SUBSCR
! Validate	Failed	4 min ago	Mon Jan 09 2...	Third I

6. Select one of the operations to see a summary of the event.

OPERATION NAME

! Validate

Validate

Summary JSON

Operation name
Validate

Time stamp
Mon Jan 09 2017 14:14:39 GMT-0800 (Pacific Standard Time)

Event initiated by

Error code
KeyVaultParameterReferenceNotFound

Message
The specified KeyVault '/subscriptions/ ... could not be found.

PowerShell

1. To retrieve log entries, run the **Get-AzureRmLog** command. You provide additional parameters to filter the list of entries. If you do not specify a start and end time, entries for the last hour are returned. For example, to retrieve the operations for a resource group during the past hour run:

```
Get-AzureRmLog -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-AzureRmLog -ResourceGroup ExampleGroup -StartTime 2015-08-28T06:00 -EndTime 2015-09-10T06:00
```

Or, you can use date functions to specify the date range, such as the last 14 days.

```
Get-AzureRmLog -ResourceGroup ExampleGroup -StartTime (Get-Date).AddDays(-14)
```

2. 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, if you are trying to research how a web app was stopped, you could run the following command:

```
Get-AzureRmLog -ResourceGroup ExampleGroup -StartTime (Get-Date).AddDays(-14) | Where-Object  
OperationName -eq Microsoft.Web/sites/stop/action
```

Which for this example shows that a stop action was performed by someone@contoso.com.

```
Authorization      :  
Scope      : /subscriptions/xxxxx/resourcegroups/ExampleGroup/providers/Microsoft.Web/sites/ExampleSite  
Action     : Microsoft.Web/sites/stop/action  
Role       : Subscription Admin  
Condition   :  
Caller      : someone@contoso.com  
CorrelationId  : 84beae59-92aa-4662-a6fc-b6fecc0ff8da  
EventSource    : Administrative  
EventTimestamp : 8/28/2015 4:08:18 PM  
OperationName  : Microsoft.Web/sites/stop/action  
ResourceGroupName : ExampleGroup  
ResourceId     :  
/subscriptions/xxxxx/resourcegroups/ExampleGroup/providers/Microsoft.Web/sites/ExampleSite  
Status       : Succeeded  
SubscriptionId : xxxxx  
SubStatus     : OK
```

3. You can look up the actions taken by a particular user, even for a resource group that no longer exists.

```
Get-AzureRmLog -ResourceGroup deletedgroup -StartTime (Get-Date).AddDays(-14) -Caller  
someone@contoso.com
```

4. You can filter for failed operations.

```
Get-AzureRmLog -ResourceGroup ExampleGroup -Status Failed
```

5. You can focus on one error by looking at the status message for that entry.

```
((Get-AzureRmLog -Status Failed -ResourceGroup ExampleGroup -  
DetailedOutput).Properties[1].Content["statusMessage"] | ConvertFrom-Json).error
```

Which returns:

```
code          message
----          -----
DnsRecordInUse DNS record dns.westus.cloudapp.azure.com is already used by another public IP.
```

Azure CLI

To retrieve log entries, run the [az monitor activity-log list](#) command.

```
az monitor activity-log list --resource-group <group name>
```

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 operations with Azure Resource Manager

5/21/2018 • 4 minutes to read • [Edit Online](#)

You can view the operations for a deployment through the Azure portal. You may be most interested in viewing the operations when you have received an error during deployment so this article focuses on viewing operations that have failed. The portal provides an interface that enables you to easily find the errors and determine potential fixes.

You can troubleshoot your deployment by looking at either the audit logs, or the deployment operations. This article shows both methods. For help with resolving particular deployment errors, see [Resolve common errors when deploying resources to Azure with Azure Resource Manager](#).

Portal

To see the deployment operations, use the following steps:

1. For the resource group involved in the deployment, notice the status of the last deployment. You can select this status to get more details.

The screenshot shows the Azure portal's resource group management interface. At the top, it displays 'ExampleGroup' under 'Resource group'. Below the header are three buttons: 'Settings', 'Add', and 'Delete'. The main content area is titled 'Essentials ^'. It shows the 'Subscription name' as 'Windows Azure MSDN - Visual Studio Ulti...'. Underneath, there is a section for 'Last deployment' which is highlighted with a red box. Inside this box, the date '3/16/2016 (Failed)' is visible.

2. You see the recent deployment history. Select the deployment that failed.

The screenshot shows the 'Deployment history' page for the 'examplegroup' resource group. The title bar includes icons for back, forward, and search, along with the text 'Deployment history' and 'examplegroup'. The main content area shows a single deployment entry. It features a red exclamation mark icon, the template name 'Microsoft.Template', and the timestamp '6/14/2016 12:34:23 PM'.

3. Select the link to see a description of why the deployment failed. In the image below, the DNS record is not unique.

The screenshot shows the Microsoft.Template deployment page. At the top, there are buttons for Delete, Cancel, Refresh, Redeploy, and View template. A red banner at the top says "Failed. Click here for details →". Below the banner, there's a summary table:

Summary	
DEPLOYMENT DATE	6/14/2016 12:15:04 PM
STATUS	Failed
RESOURCE GROUP	examplegroup

To the right of the summary, an "Error details" panel is open, displaying the following message:

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)

- DNS record
dns.centralus.cloudapp.azure.com is already used by another public IP. (Code: DnsRecordInUse)

This error message should be enough for you to begin troubleshooting. However, if you need more details about which tasks were completed, you can view the operations as shown in the following steps.

4. You can view all the deployment operations. Select any operation to see more details.

Operation details				
RESOURCE	TYPE	STATUS	TIMESTAMP	
tfstorage9856	Microsoft.Storage/storageAcco...	OK	2016-06-14T21:18...	
myPublicIP	Microsoft.Network/publicIPAdd...	BadRequest	2016-06-14T21:18...	
myVNET	Microsoft.Network/virtualNetw...	OK	2016-06-14T21:18...	
myAvSet	Microsoft.Compute/availabilityS...	OK	2016-06-14T21:18...	

In this case, you see that the storage account, virtual network, and availability set were successfully created. The public IP address failed, and other resources were not attempted.

5. You can view events for the deployment by selecting **Events**.

The screenshot shows the Microsoft.Template deployment page. At the top, there are buttons for Delete, Cancel, Refresh, Redeploy, and View template. A red banner at the top says "Failed. Click here for details →". Below the banner, there's a summary table:

Summary	
DEPLOYMENT DATE	6/14/2016 2:18:21 PM
STATUS	Failed
RESOURCE GROUP	examplegroup
RELATED	Events

6. You see all the events for the deployment and select any one for more details. Notice the correlation IDs. This value can be helpful when working with technical support to troubleshoot a deployment.

The screenshot shows a Microsoft Azure Resource Explorer window with the title "Microsoft.Resources/deployments/write". The main content area displays deployment details:

LEVEL	Error
STATUS	Failed
TIME	Tuesday, June 14, 2016 2:18:21 PM
CALLER	someone@example.com
CORRELATION IDS	a1ffc1c1-5b71-487f-9306-4257f6a7444c

Below this is a table titled "EVENT" showing deployment operations:

EVENT	LEVEL	STATUS	TIME
Microsoft.Resources/de...	>Error	Failed	5...
Microsoft.Storage/stora...	Informational	Succeeded	5...
Microsoft.Network/publi...	Error	Failed	5...
Microsoft.Network/publi...	Error	Failed	5...
Microsoft.Network/virtu...	Informational	Succeeded	5...
Microsoft.Compute/avail...	Informational	Succeeded	5...

PowerShell

1. To get the overall status of a deployment, use the **Get-AzureRmResourceGroupDeployment** command.

```
Get-AzureRmResourceGroupDeployment -ResourceGroupName ExampleGroup
```

Or, you can filter the results for only those deployments that have failed.

```
Get-AzureRmResourceGroupDeployment -ResourceGroupName ExampleGroup | Where-Object ProvisioningState -eq Failed
```

2. 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-AzureRmResourceGroupDeploymentOperation**.

```
Get-AzureRmResourceGroupDeploymentOperation -ResourceGroupName ExampleGroup -DeploymentName vmDeployment
```

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=2016-06-14T21:55:15.0156208Z;
                  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-AzureRmResourceGroupDeploymentOperation -DeploymentName Microsoft.Template -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            : 2016-06-14T21:54:55.1468068Z
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 will 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-AzureRmResourceGroupDeploymentOperation -DeploymentName Microsoft.Template -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. The request content is what you sent to Azure during deployment (for example, create a VM, OS disk, and other resources). The response content is what Azure sent back from your deployment request. During deployment, you can use **DeploymentLogLevel** parameter to specify that the request and/or response are retained in the log.

You get that information from the log, and save it locally by using the following PowerShell commands:

```
(Get-AzureRmResourceGroupDeploymentOperation -DeploymentName "TestDeployment" -ResourceGroupName "Test-RG").Properties.request | ConvertTo-Json | Out-File -FilePath <PathToFile>
```

```
(Get-AzureRmResourceGroupDeploymentOperation -DeploymentName "TestDeployment" -ResourceGroupName "Test-RG").Properties.response | ConvertTo-Json | Out-File -FilePath <PathToFile>
```

Azure CLI

1. Get the overall status of a deployment with the **azure group deployment show** command.

```
az group deployment show -g ExampleGroup -n ExampleDeployment
```

2. One of the returned values is the **correlationId**. This value 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
```

3. To see the operations for a deployment, use:

```
az group deployment operation list -g ExampleGroup -n ExampleDeployment
```

REST

1. Get information about a deployment with 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 http://aka.ms/arm-debug for usage details.",  
      "details": [{"code": "Conflict", "message": "\r\n        \"error\": {\r\n          \"message\": \"Conflict\"\r\n        }\r\n      "}]  
    }  
  }  
}
```

2. Get information about deployments with [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

8/2/2018 • 9 minutes to read • [Edit Online](#)

This article describes some common Azure deployment errors you may encounter, and provides information to resolve the errors. If you cannot find the error code for your deployment error, see [Find error code](#).

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 does not have resources available or cannot 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 does not have sufficient access to complete the deployment. Check the role your account belongs to, and its access for the deployment scope.	Azure Role-Based Access Control
BadRequest	You sent deployment values that do not match what is expected by Resource Manager. Check the inner status message for help with troubleshooting.	Template reference and Supported locations
Conflict	You are requesting an operation that is not permitted 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 does not 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

Error code	Mitigation	More information
DeploymentQuotaExceeded	<p>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-AzureRmResourceGroupDeployment in PowerShell. Deleting an entry from the deployment history does not affect the deployed resources.</p>	
DnsRecordInUse	<p>The DNS record name must be unique. Either provide a different name, or modify the existing record.</p>	
ImageNotFound	<p>Check VM image settings.</p>	
InUseSubnetCannotBeDeleted	<p>You may encounter this error when attempting to update a resource, but the request is processed by deleting and creating the resource. Make sure to specify all unchanged values.</p>	Update resource
InvalidAuthenticationTokenTenant	<p>Get access token for the appropriate tenant. You can only get the token from the tenant that your account belongs to.</p>	
InvalidContentLink	<p>You have most likely attempted to link to a nested template that is not 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.</p>	Linked templates
InvalidParameter	<p>One of the values you provided for a resource does not 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.</p>	
InvalidRequestContent	<p>Your deployment values either include values that are not expected or are missing required values. Confirm the values for your resource type.</p>	Template reference
InvalidRequestFormat	<p>Enable debug logging when executing the deployment, and verify the contents of the request.</p>	Debug logging
InvalidResourceNamespace	<p>Check the resource namespace you specified in the type property.</p>	Template reference

ERROR CODE	MITIGATION	MORE INFORMATION
InvalidResourceReference	The resource either does not 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
Invalid ResourceType	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 are deploying to.	
LinkedInvalidPropertyId	The resource ID for a resource is not 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
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

ERROR CODE	MITIGATION	MORE INFORMATION
ParentResourceNotFound	Make sure a parent resource exists before creating the child resources.	Resolve parent resource
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
ReservedResourceName	Provide a resource name that does not 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 cannot be resolved. Verify that your use of the reference function includes the parameters required for your scenario.	Resolve references
ResourceQuotaExceeded	The deployment is attempting 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 have selected.	Resolve SKU
StorageAccountAlreadyExists	Provide a unique name for the storage account.	Resolve storage account name

ERROR CODE	MITIGATION	MORE INFORMATION
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 are attempting to use.	
SubnetsNotInSameVnet	A virtual machine can only have one virtual network. When deploying multiple 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 do not 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 [?](#)

Performance [?](#)
 Standard Premium

Replication [?](#)

* Storage service encryption (blobs and files) [?](#)
 Disabled Enabled

* Secure transfer required [?](#)
 Disabled Enabled

* Subscription

* Resource group [?](#)
 Create new Use existing

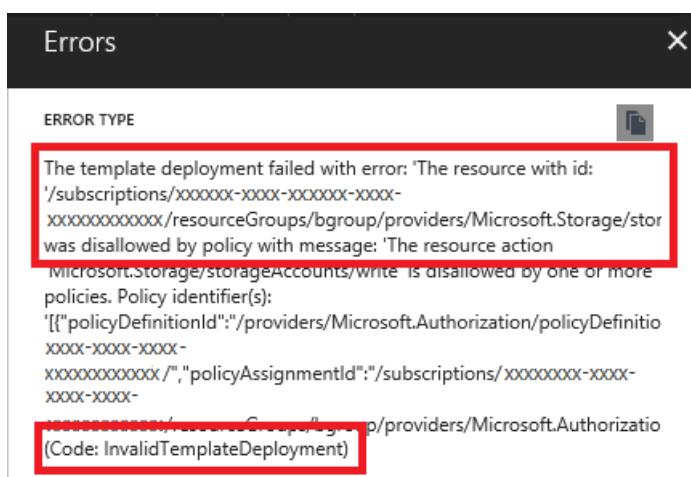
* Location

There were validation errors. Click here to view details.

Pin to dashboard

Create [Automation options](#)

Select the message for more details. In the following image, you see an **InvalidTemplateDeployment** error and a message that indicates a policy blocked deployment.



Deployment errors

When the operation passes validation, but fails during deployment, you get a deployment error.

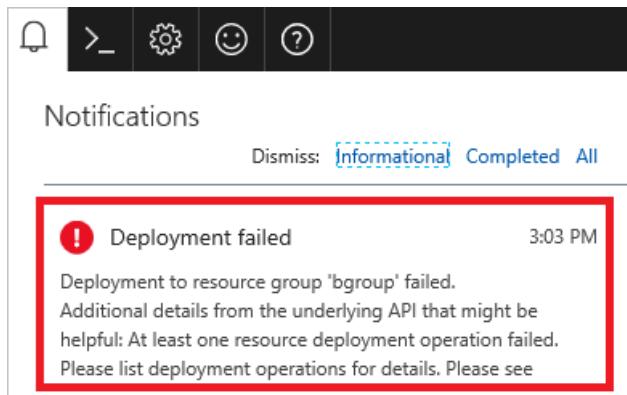
To see deployment error codes and messages with PowerShell, use:

```
(Get-AzureRmResourceGroupDeploymentOperation -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.



You see more details about the deployment. Select the option to find more information about the error.

A screenshot of the Azure portal's Deployment details page. At the top, there is a toolbar with buttons for "Delete", "Cancel", "Refresh", "Redeploy", and "View template". Below the toolbar, a red box highlights a button labeled "Failed. Click here for details →". The main content area is titled "Summary" and contains the following deployment details:

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.

Errors

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-4502-bd1c-d75a47b224b5 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-AzureRmResourceGroupDeployment ` 
-Name exampledeployment ` 
-ResourceGroupName examplegroup ` 
-TemplateFile c:\Azure\Templates\storage.json ` 
-DeploymentDebugLogLevel All
```

Examine the request content with the following cmdlet:

```
(Get-AzureRmResourceGroupDeploymentOperation ` 
-DeploymentName exampledeployment ` 
-ResourceGroupName examplegroup).Properties.request ` 
| ConvertTo-Json
```

Or, the response content with:

```
(Get-AzureRmResourceGroupDeploymentOperation ` 
-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 are 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 are 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 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

5/21/2018 • 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.

Troubleshoot deploying Linux virtual machine issues in Azure

5/11/2018 • 3 minutes to read • [Edit Online](#)

To troubleshoot virtual machine (VM) deployment issues in Azure, review the [top issues](#) for common failures and resolutions.

If you need more help at any point in this article, you can contact the Azure experts on [the MSDN Azure and Stack Overflow forums](#). Alternatively, you can file an Azure support incident. Go to the [Azure support site](#) and select **Get Support**.

Top issues

The following top issues may help resolve your issue. To start troubleshooting, review these steps:

- [The cluster cannot support the requested VM size](#)
- [The cluster does not have free resources](#)

The cluster cannot support the requested VM size

- Retry the request using a smaller VM size.
- If the size of the requested VM cannot be changed:
 - Stop all the VMs in the availability set. Click **Resource groups** > your resource group > **Resources** > your availability set > **Virtual Machines** > your virtual machine > **Stop**.
 - After all the VMs stop, create the VM in the desired size.
 - Start the new VM first, and then select each of the stopped VMs and click Start.

The cluster does not have free resources

- Retry the request later.
- If the new VM can be part of a different availability set
 - Create a VM in a different availability set (in the same region).
 - Add the new VM to the same virtual network.

How do I activate my monthly credit for Visual studio Enterprise (BizSpark)

To activate your monthly credit, see this [article](#).

Why can I not install the GPU driver for an Ubuntu NV VM?

Currently, Linux GPU support is only available on Azure NC VMs running Ubuntu Server 16.04 LTS. For more information, see [Set up GPU drivers for N-series VMs running Linux](#).

My drivers are missing for my Linux N-Series VM

Drivers for Linux-based VMs are located [here](#).

I can't find a GPU instance within my N-Series VM

To take advantage of the GPU capabilities of Azure N-series VMs running Windows Server 2016 or Windows Server 2012 R2, you must install NVIDIA graphics drivers on each VM after deployment. Driver setup information is available for [Windows VMs](#) and [Linux VMs](#).

Is N-Series VMs available in my region?

You can check the availability from the [Products available by region table](#), and pricing [here](#).

I am not able to see VM Size family that I want when resizing my VM.

When a VM is running, it is deployed to a physical server. The physical servers in Azure regions are grouped in clusters of common physical hardware. Resizing a VM that requires the VM to be moved to different hardware clusters is different depending on which deployment model was used to deploy the VM.

- VMs deployed in Classic deployment model, the cloud service deployment must be removed and redeployed to change the VMs to a size in another size family.
- VMs deployed in Resource Manager deployment model, you must stop all VMs in the availability set before changing the size of any VM in the availability set.

The listed VM size is not supported while deploying in Availability Set.

Choose a size that is supported on the availability set's cluster. It is recommended when creating an availability set to choose the largest VM size you think you need, and have that be your first deployment to the Availability set.

What Linux distributions/versions are supported on Azure?

You can find the list at Linux on [Azure-Endorsed Distributions](#).

Can I add an existing Classic VM to an availability set?

Yes. You can add an existing classic VM to a new or existing Availability Set. For more information see [Add an existing virtual machine to an availability set](#).

Next steps

If you need more help at any point in this article, you can contact the Azure experts on [the MSDN Azure and Stack Overflow forums](#).

Alternatively, you can file an Azure support incident. Go to the [Azure support site](#) and select **Get Support**.

Resolve errors for resource provider registration

5/21/2018 • 2 minutes to read • [Edit Online](#)

This article describes the errors you may encounter when using a resource provider that you have not previously used in your subscription.

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 are using, but not all. If you have not used a particular resource provider before, you may need to register that provider.

Cause

You receive these errors for one of three reasons:

1. The resource provider has not been registered for your subscription
2. API version not supported for the resource type
3. Location not supported for the resource type

Solution 1 - PowerShell

For PowerShell, use **Get-AzureRmResourceProvider** to see your registration status.

```
Get-AzureRmResourceProvider -ListAvailable
```

To register a provider, use **Register-AzureRmResourceProvider** and provide the name of the resource provider you wish to register.

```
Register-AzureRmResourceProvider -ProviderNamespace Microsoft.Cdn
```

To get the supported locations for a particular type of resource, use:

```
((Get-AzureRmResourceProvider -ProviderNamespace Microsoft.Web).ResourceTypes | Where-Object ResourceTypeName -eq sites).Locations
```

To get the supported API versions for a particular type of resource, use:

```
((Get-AzureRmResourceProvider -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
```

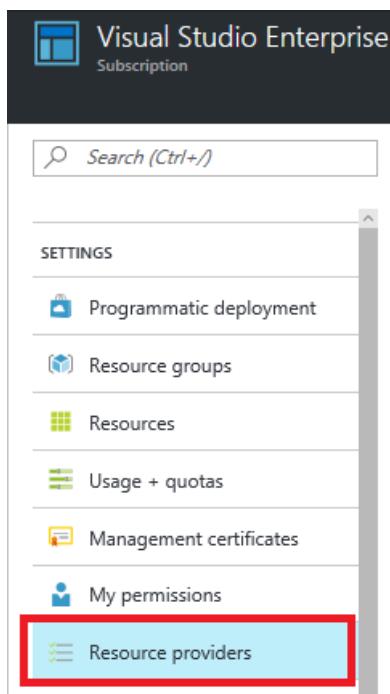
To see the supported locations and API versions for a resource type, use:

```
az provider show -n Microsoft.Web --query "resourceTypes[?resourceType=='sites'].locations"
```

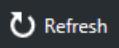
Solution 3 - Azure portal

You can see the registration status and register a resource provider namespace through the portal.

1. For your subscription, select **Resource providers**.



2. Look at the list of resource providers, and if necessary, select the **Register** link to register the resource provider of the type you are trying to deploy.



Search to filter resource providers...

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

6/6/2018 • 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.

The screenshot shows the Azure portal interface for a resource group named 'examplegroup'. At the top, there's a search bar and buttons for 'Add', 'Columns', and 'Delete'. Below that, the 'Essentials' section displays the 'Subscription name (change)' as 'Azure Internal Consumption'. Under 'Deployments', it shows '1 Succeeded' deployment. The left sidebar has links for 'Overview' (which is selected) and 'Activity log'.

2. Select a deployment from the history, and select **Events**.

The screenshot shows the deployment history for 'examplegroup'. A specific deployment for 'Microsoft.StorageAccount-2016121211...' dated 12/12/2016 11:40:09 AM is selected. On the right, the deployment details are shown: DEPLOYMENT DATE: 12/12/2016 11:40:09; STATUS: Succeeded; DURATION: 25 seconds; RESOURCE GROUP: examplegroup. The 'Events' tab is highlighted with a red box.

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	<i>ⓘ</i>	Succeeded	4 min ago
Microsoft.Storage/storageAccounts/write	<i>ⓘ</i>	Succeeded	4 min ago
Microsoft.Storage/storageAccounts/write	<i>ⓘ</i>	Succeeded	4 min ago
Microsoft.Storage/storageAccounts/write	<i>ⓘ</i>	Succeeded	4 min ago
Microsoft.Storage/storageAccounts/write	<i>ⓘ</i>	Accepted	4 min ago
Microsoft.Storage/storageAccounts/write	<i>ⓘ</i>	Accepted	4 min ago
Microsoft.Storage/storageAccounts/write	<i>ⓘ</i>	Accepted	4 min ago
Microsoft.Storage/storageAccounts/write	<i>ⓘ</i>	Started	4 min ago
Microsoft.Storage/storageAccounts/write	<i>ⓘ</i>	Started	4 min ago
Microsoft.Storage/storageAccounts/write	<i>ⓘ</i>	Started	4 min ago
Microsoft.Resources/deployments/write	<i>ⓘ</i>	Succeeded	4 min ago
Microsoft.Resources/deployments/write	<i>ⓘ</i>	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	<i>ⓘ</i>	Succeeded	Just now
Microsoft.Storage/storageAccounts/write	<i>ⓘ</i>	Succeeded	Just now
Microsoft.Storage/storageAccounts/write	<i>ⓘ</i>	Accepted	1 min ago
Microsoft.Storage/storageAccounts/write	<i>ⓘ</i>	Started	1 min ago
Microsoft.Storage/storageAccounts/write	<i>ⓘ</i>	Succeeded	1 min ago
Microsoft.Storage/storageAccounts/write	<i>ⓘ</i>	Accepted	2 min ago
Microsoft.Storage/storageAccounts/write	<i>ⓘ</i>	Started	2 min ago
Microsoft.Storage/storageAccounts/write	<i>ⓘ</i>	Succeeded	2 min ago
Microsoft.Storage/storageAccounts/write	<i>ⓘ</i>	Accepted	2 min ago
Microsoft.Storage/storageAccounts/write	<i>ⓘ</i>	Started	2 min ago
Microsoft.Resources/deployments/write	<i>ⓘ</i>	Succeeded	2 min ago
Microsoft.Resources/deployments/write	<i>ⓘ</i>	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

8/2/2018 • 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_CI_AS",  
                "edition": "Basic"  
            }  
        }  
    ],  
    "outputs": {}  
}
```

For more information, see [Define the order for deploying resources in Azure Resource Manager templates](#).

Troubleshoot Resource Manager deployment issues with creating a new Linux virtual machine in Azure

1/3/2018 • 4 minutes to read • [Edit Online](#)

When you try to create a new Azure Virtual Machine (VM), the common errors you encounter are provisioning failures or allocation failures.

- A provisioning failure happens when the OS image fails to load either due to incorrect preparatory steps or because of selecting the wrong settings during the image capture from the portal.
- An allocation failure results when the cluster or region either does not have resources available or cannot support the requested VM size.

If your Azure issue is not addressed in this article, visit the Azure forums on [MSDN and Stack Overflow](#). You can post your issue in these forums, or post to [@AzureSupport on Twitter](#). You also can submit an Azure support request. To submit a support request, on the [Azure support](#) page, select **Get support**.

Top issues

The following top issues may help resolve your issue. To start troubleshooting, review these steps:

- [The cluster cannot support the requested VM size](#)
- [The cluster does not have free resources](#)

For other VM deployment issues and questions, see [Troubleshoot deploying Linux virtual machine issues in Azure](#).

Collect activity logs

To start troubleshooting, collect the activity logs to identify the error associated with the issue. The following links contain detailed information on the process to follow.

[View deployment operations](#)

[View activity logs to manage Azure resources](#)

Issue: Custom image; provisioning errors

Provisioning errors arise if you upload or capture a generalized VM image as a specialized VM image or vice versa. The former will cause a provisioning timeout error and the latter will cause a provisioning failure. To deploy your custom image without errors, you must ensure that the type of the image does not change during the capture process.

The following table lists the possible combinations of generalized and specialized images, the error type you will encounter and what you need to do to fix the errors.

The following table lists the possible upload and capture combinations of Linux generalized and specialized OS images. The combinations that will process without any errors are indicated by a Y, and those that will throw errors are indicated by an N. The causes and resolutions for the different errors you will run into are given below the table.

OS	UPLOAD SPEC.	UPLOAD GEN.	CAPTURE SPEC.	CAPTURE GEN.
Linux gen.	N ¹	Y	N ³	Y
Linux spec.	Y	N ²	Y	N ⁴

Y: If the OS is Linux generalized, and it is uploaded and/or captured with the generalized setting, then there won't be any errors. Similarly, if the OS is Linux specialized, and it is uploaded and/or captured with the specialized setting, then there won't be any errors.

Upload Errors:

N¹: If the OS is Linux generalized, and it is uploaded as specialized, you will get a provisioning timeout error because the VM is stuck at the provisioning stage.

N²: If the OS is Linux specialized, and it is uploaded as generalized, you will get a provisioning failure error because the new VM is running with the original computer name, username and password.

Resolution:

To resolve both these errors, upload the original VHD, available on-prem, with the same setting as that for the OS (generalized/specialized). To upload as generalized, remember to run -deprovision first.

Capture Errors:

N³: If the OS is Linux generalized, and it is captured as specialized, you will get a provisioning timeout error because the original VM is not usable as it is marked as generalized.

N⁴: If the OS is Linux specialized, and it is captured as generalized, you will get a provisioning failure error because the new VM is running with the original computer name, username and password. Also, the original VM is not usable because it is marked as specialized.

Resolution:

To resolve both these errors, delete the current image from the portal, and [recapture it from the current VHDs](#) with the same setting as that for the OS (generalized/specialized).

Issue: Custom/ gallery/ marketplace image; allocation failure

This error arises in situations when the new VM request is pinned to a cluster that either cannot support the VM size being requested, or does not have available free space to accommodate the request.

Cause 1: The cluster cannot support the requested VM size.

Resolution 1:

- Retry the request using a smaller VM size.
- If the size of the requested VM cannot be changed:
 - Stop all the VMs in the availability set. Click **Resource groups** > *your resource group* > **Resources** > *your availability set* > **Virtual Machines** > *your virtual machine* > **Stop**.
 - After all the VMs stop, create the new VM in the desired size.
 - Start the new VM first, and then select each of the stopped VMs and click **Start**.

Cause 2: The cluster does not have free resources.

Resolution 2:

- Retry the request at a later time.

- If the new VM can be part of a different availability set
 - Create a new VM in a different availability set (in the same region).
 - Add the new VM to the same virtual network.

Next steps

If you encounter issues when you start a stopped Linux VM or resize an existing Linux VM in Azure, see

[Troubleshoot Resource Manager deployment issues with restarting or resizing an existing Linux Virtual Machine in Azure](#).

Troubleshoot deployment issues when creating a new Windows VM in Azure

6/15/2018 • 4 minutes to read • [Edit Online](#)

When you try to create a new Azure Virtual Machine (VM), the common errors you encounter are provisioning failures or allocation failures.

- A provisioning failure happens when the OS image fails to load either due to incorrect preparatory steps or because of selecting the wrong settings during the image capture from the portal.
- An allocation failure results when the cluster or region either does not have resources available or cannot support the requested VM size.

If your Azure issue is not addressed in this article, visit the Azure forums on [MSDN and Stack Overflow](#). You can post your issue in these forums, or post to [@AzureSupport on Twitter](#). You also can submit an Azure support request. To submit a support request, on the [Azure support](#) page, select **Get support**.

Top issues

The following top issues may help resolve your issue. To start troubleshooting, review these steps:

- [The cluster cannot support the requested VM size](#)
- [The cluster does not have free resources](#)

For other VM deployment issues and questions, see [Troubleshoot deploying Windows virtual machine issues in Azure](#).

Collect activity logs

To start troubleshooting, collect the activity logs to identify the error associated with the issue. The following links contain detailed information on the process to follow.

[View deployment operations](#)

[View activity logs to manage Azure resources](#)

Issue: Custom image; provisioning errors

Provisioning errors arise if you upload or capture a generalized VM image as a specialized VM image or vice versa. The former will cause a provisioning timeout error and the latter will cause a provisioning failure. To deploy your custom image without errors, you must ensure that the type of the image does not change during the capture process.

The following table lists the possible combinations of generalized and specialized images, the error type you will encounter and what you need to do to fix the errors.

The following table lists the possible upload and capture combinations of Windows generalized (gen.) and specialized (spec.) OS images. The combinations that will process without any errors are indicated by a Y, and those that will throw errors are indicated by an N. The causes and resolutions for the different errors you will run into are given below the table.

OS	UPLOAD SPEC.	UPLOAD GEN.	CAPTURE SPEC.	CAPTURE GEN.
Windows gen.	N ¹	Y	N ³	Y
Windows spec.	Y	N ²	Y	N ⁴

Y: If the OS is Windows generalized, and it is uploaded and/or captured with the generalized setting, then there won't be any errors. Similarly, if the OS is Windows specialized, and it is uploaded and/or captured with the specialized setting, then there won't be any errors.

Upload Errors:

N¹: If the OS is Windows generalized, and it is uploaded as specialized, you will get a provisioning timeout error with the VM stuck at the OOB screen.

N²: If the OS is Windows specialized, and it is uploaded as generalized, you will get a provisioning failure error with the VM stuck at the OOB screen because the new VM is running with the original computer name, username and password.

Resolution

To resolve both these errors, use [Add-AzureRmVhd to upload the original VHD](#), available on-premises, with the same setting as that for the OS (generalized/specialized). To upload as generalized, remember to run sysprep first.

Capture Errors:

N³: If the OS is Windows generalized, and it is captured as specialized, you will get a provisioning timeout error because the original VM is not usable as it is marked as generalized.

N⁴: If the OS is Windows specialized, and it is captured as generalized, you will get a provisioning failure error because the new VM is running with the original computer name, username, and password. Also, the original VM is not usable because it is marked as specialized.

Resolution

To resolve both these errors, delete the current image from the portal, and [recapture it from the current VHDs](#) with the same setting as that for the OS (generalized/specialized).

Issue: Custom/gallery/marketplace image; allocation failure

This error arises in situations when the new VM request is pinned to a cluster that either cannot support the VM size being requested, or does not have available free space to accommodate the request.

Cause 1: The cluster cannot support the requested VM size.

Resolution 1:

- Retry the request using a smaller VM size.
- If the size of the requested VM cannot be changed:
 - Stop all the VMs in the availability set. Click **Resource groups** > *your resource group* > **Resources** > *your availability set* > **Virtual Machines** > *your virtual machine* > **Stop**.
 - After all the VMs stop, create the new VM in the desired size.
 - Start the new VM first, and then select each of the stopped VMs and click **Start**.

Cause 2: The cluster does not have free resources.

Resolution 2:

- Retry the request at a later time.
- If the new VM can be part of a different availability set
 - Create a new VM in a different availability set (in the same region).
 - Add the new VM to the same virtual network.

Next steps

If you encounter issues when you start a stopped Windows VM or resize an existing Windows VM in Azure, see [Troubleshoot Resource Manager deployment issues with restarting or resizing an existing Windows Virtual Machine in Azure](#).

RequestDisallowedByPolicy error with Azure resource policy

6/13/2018 • 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

In PowerShell, provide that policy identifier as the `-Id` parameter to retrieve details about the policy that blocked your deployment.

```
(Get-AzureRmPolicyDefinition -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

5/21/2018 • 2 minutes to read • [Edit Online](#)

This article describes quota errors you may encounter when deploying resources.

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-AzureRmVMUsage** command to find virtual machine quotas.

```
Get-AzureRmVMUsage -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	

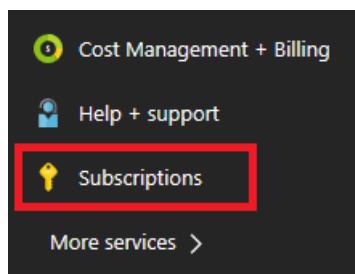
Solution

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.

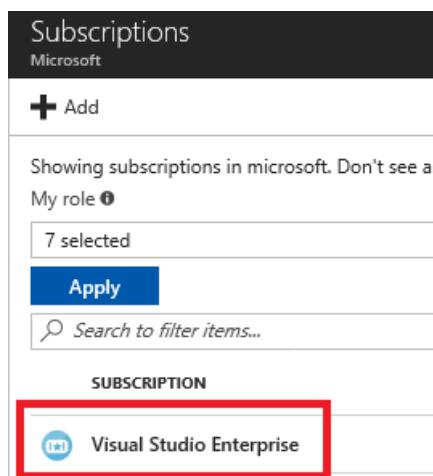
NOTE

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.

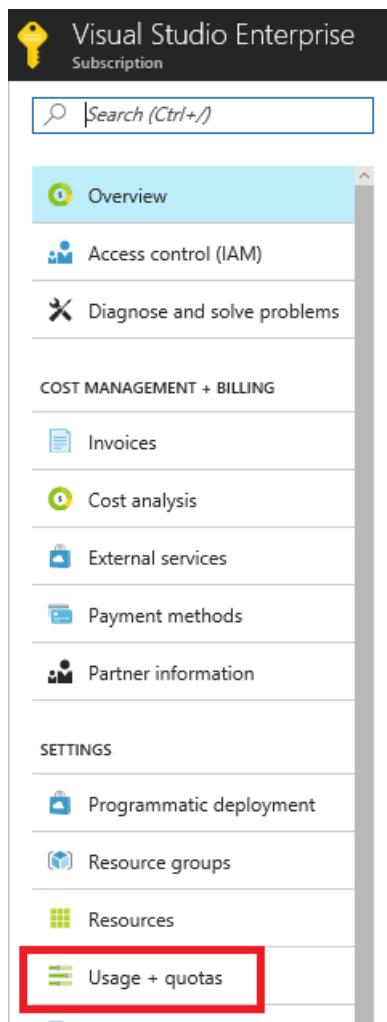
1. Select **Subscriptions**.



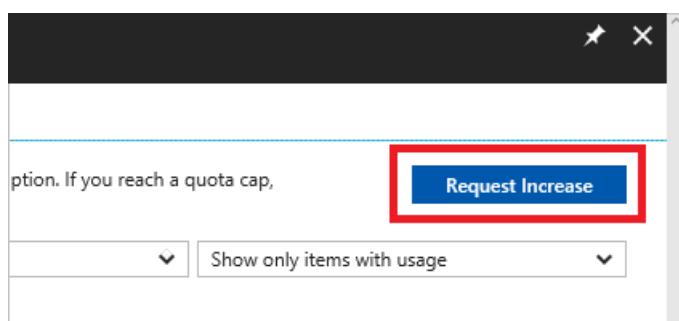
2. Select the subscription that needs an increased quota.



3. Select **Usage + quotas**



4. In the upper right corner, select **Request increase**.



5. Fill in the forms for the type of quota you need to increase.

New support request X

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

5/21/2018 • 2 minutes to read • [Edit Online](#)

This article describes how to resolve the **SkuNotAvailable** error.

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 have selected (such as VM size) is not available for the location you have selected.

Solution 1 - PowerShell

To determine which SKUs are available in a region, use the [Get-AzureRmComputeResourceSku](#) command. Filter the results by location. You must have the latest version of PowerShell for this command.

```
Get-AzureRmComputeResourceSku | where {$_.Locations -icontains "southcentralus"}
```

The results include a list of SKUs for the location and any restrictions for that SKU.

ResourceType	Name	Locations	Restriction	Capability	Value
availabilitySets	Classic	southcentralus		MaximumPlatformFaultDomainCount	3
availabilitySets	Aligned	southcentralus		MaximumPlatformFaultDomainCount	3
virtualMachines	Standard_A0	southcentralus			
virtualMachines	Standard_A1	southcentralus			
virtualMachines	Standard_A2	southcentralus			

Solution 2 - Azure CLI

To determine which SKUs are available in a region, use the `az vm list-skus` command. You can then use `grep` or a similar utility to filter the output.

ResourceType	Locations	Name	Capabilities	Tier
Size	Restrictions			
availabilitySets	eastus	Classic	MaximumPlatformFaultDomainCount=3	
availabilitySets	eastus	Aligned	MaximumPlatformFaultDomainCount=3	
availabilitySets	eastus2	Classic	MaximumPlatformFaultDomainCount=3	
availabilitySets	eastus2	Aligned	MaximumPlatformFaultDomainCount=3	
availabilitySets	westus	Classic	MaximumPlatformFaultDomainCount=3	
availabilitySets	westus	Aligned	MaximumPlatformFaultDomainCount=3	
availabilitySets	centralus	Classic	MaximumPlatformFaultDomainCount=3	
availabilitySets	centralus	Aligned	MaximumPlatformFaultDomainCount=3	

Solution 3 - Azure portal

To determine which SKUs are available in a region, use the [portal](#). Log 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 do not need to complete the deployment.

Choose a size

Browse the available sizes and their features

Prices presented below are estimates in your local currency that include only Azure infrastructure costs and any discounts for the subscription and location. The prices don't include any applicable software costs. Recommended sizes are determined by the publisher of the selected image based on hardware and software requirements.

★ Recommended | View all

DS1_V2 Standard	DS2_V2 Standard	DS3_V2 Standard
1 Core	2 Cores	4 Cores
3.5 GB	7 GB	14 GB
2 Data disks 3200 Max IOPS 7 GB Local SSD Load balancing Auto scale Premium disk supp...	4 Data disks 6400 Max IOPS 14 GB Local SSD Load balancing Auto scale Premium disk supp...	8 Data disks 12800 Max IOPS 28 GB Local SSD Load balancing Auto scale Premium disk supp...
54.31 USD/MONTH (ESTIMATED)	108.62 USD/MONTH (ESTIMATED)	217.99 USD/MONTH (ESTIMATED)

Solution 4 - REST

To determine which SKUs are available in a region, use the REST API for virtual machines. Send the following request:

```
GET
https://management.azure.com/subscriptions/{subscription-id}/providers/Microsoft.Compute/skus?api-version=2016-03-30
```

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": []  
    },  
    ...  
  ]  
}
```

If you are 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.

Troubleshoot deploying Windows virtual machine issues in Azure

5/11/2018 • 4 minutes to read • [Edit Online](#)

To troubleshoot virtual machine (VM) deployment issues in Azure, review the [top issues](#) for common failures and resolutions.

If you need more help at any point in this article, you can contact the Azure experts on [the MSDN Azure and Stack Overflow forums](#). Alternatively, you can file an Azure support incident. Go to the [Azure support site](#) and select **Get Support**.

Top issues

The following top issues may help resolve your issue. To start troubleshooting, review these steps:

- [The cluster cannot support the requested VM size](#)
- [The cluster does not have free resources](#)

The cluster cannot support the requested VM size

- Retry the request using a smaller VM size.
- If the size of the requested VM cannot be changed:
 - Stop all the VMs in the availability set. Click **Resource groups** > your resource group > **Resources** > your availability set > **Virtual Machines** > your virtual machine > **Stop**.
 - After all the VMs stop, create the VM in the desired size.
 - Start the new VM first, and then select each of the stopped VMs and click Start.

The cluster does not have free resources

- Retry the request later.
- If the new VM can be part of a different availability set
 - Create a VM in a different availability set (in the same region).
 - Add the new VM to the same virtual network.

How can I use and deploy a windows client image into Azure?

You can use Windows 7, Windows 8, or Windows 10 in Azure for dev/test scenarios if you have an appropriate Visual Studio (formerly MSDN) subscription. This [article](#) outlines the eligibility requirements for running Windows client in Azure and uses of the Azure Gallery images.

How can I deploy a virtual machine using the Hybrid Use Benefit (HUB)?

There are a couple of different ways to deploy Windows virtual machines with the Azure Hybrid Use Benefit.

For an Enterprise Agreement subscription:

- Deploy VMs from specific Marketplace images that are pre-configured with Azure Hybrid Use Benefit.

For Enterprise agreement:

- Upload a custom VM and deploy using a Resource Manager template or Azure PowerShell.

For more information, see the following resources:

- [Azure Hybrid Use Benefit overview](#)
- [Downloadable FAQ](#)
- [Azure Hybrid Use Benefit for Windows Server and Windows Client](#).
- [How can I use the Hybrid Use Benefit in Azure](#)

How do I activate my monthly credit for Visual studio Enterprise (BizSpark)

To activate your monthly credit, see this [article](#).

How to add Enterprise Dev/Test to my Enterprise Agreement (EA) to get access to Window client images?

The ability to create subscriptions based on the Enterprise Dev/Test offer is restricted to Account Owners who have been given permission to do so by an Enterprise Administrator. The Account Owner creates subscriptions via the Azure Account Portal, and then should add active Visual Studio subscribers as co-administrators. So that they can manage and use the resources needed for development and testing. For more information, see [Enterprise Dev/Test](#).

My drivers are missing for my Windows N-Series VM

Drivers for Windows-based VMs are located [here](#).

I can't find a GPU instance within my N-Series VM

To take advantage of the GPU capabilities of Azure N-series VMs running Windows Server 2016 or Windows Server 2012 R2, you must install NVIDIA graphics drivers on each VM after deployment. Driver setup information is available for [Windows VMs](#) and [Linux VMs](#).

Is N-Series VMs available in my region?

You can check the availability from the [Products available by region table](#), and pricing [here](#).

What client images can I use and deploy in Azure, and how to I get them?

You can use Windows 7, Windows 8, or Windows 10 in Azure for dev/test scenarios provided you have an appropriate Visual Studio (formerly MSDN) subscription.

- Windows 10 images are available from the Azure Gallery within [eligible dev/test offers](#).
- Visual Studio subscribers within any type of offer can also [adequately prepare and create](#) a 64-bit Windows 7, Windows 8, or Windows 10 image and then [upload to Azure](#). The use remains limited to dev/test by active Visual Studio subscribers.

This [article](#) outlines the eligibility requirements for running Windows client in Azure and use of the Azure Gallery images.

I am not able to see VM Size family that I want when resizing my VM.

When a VM is running, it is deployed to a physical server. The physical servers in Azure regions are grouped in clusters of common physical hardware. Resizing a VM that requires the VM to be moved to different hardware clusters is different depending on which deployment model was used to deploy the VM.

- VMs deployed in Classic deployment model, the cloud service deployment must be removed and redeployed to change the VMs to a size in another size family.
- VMs deployed in Resource Manager deployment model, you must stop all VMs in the availability set before changing the size of any VM in the availability set.

The listed VM size is not supported while deploying in Availability Set.

Choose a size that is supported on the availability set's cluster. It is recommended when creating an availability set to choose the largest VM size you think you need, and have that be your first deployment to the Availability set.

Can I add an existing Classic VM to an availability set?

Yes. You can add an existing classic VM to a new or existing Availability Set. For more information see [Add an existing virtual machine to an availability set](#).

Next steps

If you need more help at any point in this article, you can contact the Azure experts on [the MSDN Azure and Stack Overflow forums](#).

Alternatively, you can file an Azure support incident. Go to the [Azure support site](#) and select **Get Support**.

Azure Resource Manager template functions

5/21/2018 • 2 minutes to read • [Edit Online](#)

This article describes all the functions you can use in an Azure Resource Manager template.

You add functions in your templates by enclosing them within brackets: `[` and `]`, respectively. The expression is evaluated during deployment. While written as a string literal, the result of evaluating the expression can be of a different JSON type, such as an array, object, or integer. Just like in JavaScript, function calls are formatted as `functionName(arg1,arg2,arg3)`. You reference properties by using the dot and `[index]` operators.

A template expression cannot exceed 24,576 characters.

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 `toUpperCase` or `toLowerCase`), the function preserves the case. Certain resource types may have case requirements irrespective of how functions are evaluated.

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:

- [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](#)
- [guid](#)
- [indexOf](#)
- [last](#)
- [lastIndexOf](#)
- [length](#)
- [padLeft](#)
- [replace](#)
- [skip](#)
- [split](#)
- [startsWith](#)
- [string](#)
- [substring](#)
- [take](#)
- [toLower](#)
- [toUpper](#)
- [trim](#)
- [uniqueString](#)
- [uri](#)
- [uriComponent](#)
- [uriComponentToString](#)

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](#)

Array and object functions for Azure Resource Manager templates

7/9/2018 • 16 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](#).

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-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/array.json
```

```
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-AzureRmResourceGroupDeployment -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.

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
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-AzureRmResourceGroupDeployment -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:

```
{
    "$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-AzureRmResourceGroupDeployment -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-AzureRmResourceGroupDeployment -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-AzureRmResourceGroupDeployment -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-AzureRmResourceGroupDeployment -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.

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",
    "resources": [
    ],
    "outputs": {
        "jsonOutput": {
            "type": "object",
            "value": "[json('{"a": "b"}')]"
        },
        "nullOutput": {
            "type": "bool",
            "value": "[empty(json('null'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
jsonOutput	Object	{"a": "b"}
nullOutput	Boolean	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/json.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -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
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-AzureRmResourceGroupDeployment -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-AzureRmResourceGroupDeployment -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-AzureRmResourceGroupDeployment -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-AzureRmResourceGroupDeployment -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.

PARAMETER	REQUIRED	TYPE	DESCRIPTION
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-AzureRmResourceGroupDeployment -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-AzureRmResourceGroupDeployment -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-AzureRmResourceGroupDeployment -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-AzureRmResourceGroupDeployment -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

5/21/2018 • 5 minutes to read • [Edit Online](#)

Resource Manager provides several functions for making comparisons in your templates.

- [equals](#)
- [greater](#)
- [greaterOrEquals](#)
- [less](#)
- [lessOrEquals](#)

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-AzureRmResourceGroupDeployment -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-AzureRmResourceGroupDeployment -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-AzureRmResourceGroupDeployment -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-AzureRmResourceGroupDeployment -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.

```
{
    "$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-AzRmResourceGroupDeployment -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-AzureRmResourceGroupDeployment -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

5/21/2018 • 4 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](#).

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": ""  
    }  
}
```

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')]"  
}
```

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-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/deployment.json
```

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-AzureRmResourceGroupDeployment -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 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](#).

Logical functions for Azure Resource Manager templates

5/21/2018 • 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 both 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.

Return value

Returns **True** if both 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

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/andornot.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri  
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-  
manager/functions/andornot.json
```

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

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/bool.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/bool.json
```

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 is true.
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

You can use this function to conditionally set a resource property. The following example is not a full template, but it shows the relevant portions for conditionally setting the availability set.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        ...
        "availabilitySet": {
            "type": "string",
            "allowedValues": [
                "yes",
                "no"
            ]
        }
    },
    "variables": {
        ...
        "availabilitySetName": "availabilitySet1",
        "availabilitySet": {
            "id": "[resourceId('Microsoft.Compute/availabilitySets', variables('availabilitySetName'))]"
        }
    },
    "resources": [
        {
            "condition": "[equals(parameters('availabilitySet'), 'yes')]",
            "type": "Microsoft.Compute/availabilitySets",
            "name": "[variables('availabilitySetName')]",
            ...
        },
        {
            "apiVersion": "2016-03-30",
            "type": "Microsoft.Compute/virtualMachines",
            "properties": {
                "availabilitySet": "[if>equals(parameters('availabilitySet'), 'yes'),
variables('availabilitySet'), json('null'))]",
                ...
            }
        },
        ...
    ],
    ...
}
```

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']"
      }
    }
}
```

The output from the preceding example is:

NAME	TYPE	VALUE
yesOutput	String	yes
noOutput	String	no

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/if.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-manager/functions/if.json
```

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

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/andornot.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/andornot.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-AzRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-manager/functions/not-
equals.json
```

Or

```
or(arg1, arg2)
```

Checks whether either 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.

Return value

Returns **True** if either 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

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/andornot.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri  
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-  
manager/functions/andornot.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](#).

Numeric functions for Azure Resource Manager templates

5/21/2018 • 7 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](#)

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-AzureRmResourceGroupDeployment -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": "websitetscopy",
      "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-AzureRmResourceGroupDeployment -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-AzureRmResourceGroupDeployment -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-AzureRmResourceGroupDeployment -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-AzureRmResourceGroupDeployment -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-AzureRmResourceGroupDeployment -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.

```
{
    "$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-AzureRmResourceGroupDeployment -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-AzureRmResourceGroupDeployment -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

6/6/2018 • 10 minutes to read • [Edit Online](#)

Resource Manager provides the following functions for getting resource values:

- [listKeys](#)
- [listSecrets](#)
- [list*](#)
- [providers](#)
- [reference](#)
- [resourceGroup](#)
- [resourceId](#)
- [subscription](#)

To get values from parameters, variables, or the current deployment, see [Deployment value functions](#).

listKeys, listSecrets and list*

`listKeys(resourceName or resourceIdentifier, apiVersion)`

`listSecrets(resourceName or resourceIdentifier, apiVersion)`

`list{Value}(resourceName or resourceIdentifier, apiVersion)`

Returns the values for any resource type that supports the list operation. The most common usages are `listKeys` and `listSecrets`.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
resourceName or resourceId	Yes	string	Unique identifier for the resource.
apiVersion	Yes	string	API version of resource runtime state. Typically, in the format, yyyy-mm-dd .

Return value

The returned object from `listKeys` has 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

Any operation that starts with **list** can be used as a function in your template. The available operations include not only `listKeys`, but also operations like `list`, `listAdminKeys`, and `listStatus`. However, you cannot use **list** operations that require values in the request body. For example, the [List Account SAS](#) operation requires request body parameters like `signedExpiry`, so you cannot use it within a template.

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-AzureRmProviderOperation](#) PowerShell cmdlet. The following example gets all list operations for storage accounts:

```
Get-AzureRmProviderOperation -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')]"
```

Specify the resource by using either the resource name or the [resourceId function](#). When using this function in the same template that deploys the referenced resource, use the resource name.

Example

The following [example template](#) shows how to return the primary and secondary keys from a storage account in the outputs section.

```
{
  "$schema": "http://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": "[listKeys(parameters('storageAccountName'), '2016-12-01')]"
    }
  }
}
```

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/listkeys.json --parameters storageAccountName=<your-storage-account>
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/listkeys.json -storageAccountName <your-storage-account>
```

providers

```
providers(providerNamespace, [resourceType])
```

Returns information about a resource provider and its supported resource types. If you do not 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 is not 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",  
        ...  
    ]  
}
```

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/providers.json --parameters providerNamespace=Microsoft.Web resourceType=sites
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri  
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-  
manager/functions/providers.json -providerNamespace Microsoft.Web -resourceType sites
```

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 is not 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 do not 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 does not 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 derives its value from a runtime state, and therefore cannot be used in the variables section. It can be used in outputs section of a template or [linked template](#). It cannot be used 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.

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 do not need to also use the dependsOn property. The function is not evaluated until the referenced resource has completed deployment.

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 are not 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": "http://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"  
}  
}
```

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/referencewithstorage.json --parameters storageAccountName=<your-storage-account>
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri  
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-  
manager/functions/referencewithstorage.json -storageAccountName <your-storage-account>
```

The following [example template](#) references a storage account that is not deployed in this template. The storage account already exists within the same resource group.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "storageAccountName": {
            "type": "string"
        }
    },
    "resources": [],
    "outputs": {
        "ExistingStorage": {
            "value": "[reference(concat('Microsoft.Storage/storageAccounts/', parameters('storageAccountName')), '2016-01-01'))",
            "type": "object"
        }
    }
}
```

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/reference.json --parameters storageAccountName=<your-storage-account>
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/reference.json -storageAccountName <your-storage-account>
```

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

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]",
    ...
  }
]
```

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"
  }
}
```

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/resourcegroup.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/resourcegroup.json
```

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.
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}/{resourceType}/{resourceName}
```

Remarks

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'))]"
```

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": "http://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": {
        "vnetID": "[resourceId(parameters('virtualNetworkResourceGroup'), 'Microsoft.Network/virtualNetworks',
parameters('virtualNetworkName'))]",
        "subnet1Ref": "[concat(variables('vnetID'), '/subnets/', 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/11111111-1111-1111-111111111111/resourceGroups/otherResourceGroup/providers/Microsoft.Storage/storageAccounts/examplestorage
nestedResourceOutput	String	/subscriptions/{current-sub-id}/resourceGroups/examplegroup/providers/Microsoft.SQL/servers/serverName/databases/databaseName

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/resourceid.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri  
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-  
manager/functions/resourceid.json
```

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"  
        }  
    }  
}
```

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/subscription.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri  
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-  
manager/functions/subscription.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](#).

String functions for Azure Resource Manager templates

5/21/2018 • 28 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](#)
- [guid](#)
- [indexOf](#)
- [last](#)
- [lastIndexOf](#)
- [length](#)
- [padLeft](#)
- [replace](#)
- [skip](#)
- [split](#)
- [startsWith](#)
- [string](#)
- [substring](#)
- [take](#)
- [toLowerCase](#)
- [toUpperCase](#)
- [trim](#)
- [uniqueString](#)
- [uri](#)
- [uriComponent](#)
- [uriComponentToString](#)

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	b25lLCB0d28sIHRo cmVl
toStringOutput	String	one, two, three
toJsonOutput	Object	{"one": "a", "two": "b"}

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/base64.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri  
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-  
manager/functions/base64.json
```

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	b25lLCB0d28sIHRobcmVl
toStringOutput	String	one, two, three
toJsonOutput	Object	{"one": "a", "two": "b"}

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/base64.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/base64.json
```

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
toJsonOutput	Object	{"one": "a", "two": "b"}

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/base64.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri  
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-  
manager/functions/base64.json
```

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-5yj4jf5mbg72

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-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/concat-string.json
```

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
```

contains

```
contains (container, itemToFind)
```

Checks whether an array contains a value, an object contains a key, or a string contains a substring.

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

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-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/contains.json
```

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:text/plain;base64,SGVsbgG8sIFdvcmxkIQ=="
        }
    },
    "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!

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/datauri.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/datauri.json
```

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!

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/datauri.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/datauri.json
```

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 is 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

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
```

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
endsCapTrue	Bool	True
endsFalse	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/startsendswith.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-manager/functions/startssendswith.json
```

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

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-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri  
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-  
manager/functions/first.json
```

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 is not a random string, but rather the result of a hash function. The returned value is 36 characters long. It is not globally unique.

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"  
        }  
    }  
}
```

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/guid.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri  
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-  
manager/functions/guid.json
```

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 is not 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

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/indexof.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri  
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-  
manager/functions/indexof.json
```

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

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-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-manager/functions/last.json
```

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 is not 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

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/indexof.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/indexof.json
```

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

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-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/length.json
```

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

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/padleft.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/padleft.json
```

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'), '-', '')]"
        },
        "secodeOutput": {
            "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
secodeOutput	String	123-123-xxxx

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/replace.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri  
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-  
manager/functions/replace.json
```

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 is 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

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-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/skip.json
```

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"]

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/split.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/split.json
```

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

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/startsendswith.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri  
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-  
manager/functions/startssendswith.json
```

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:

```
{
    "$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"]
intOutput	String	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/string.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-manager/functions/string.json
```

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.

Return value

The substring.

Remarks

The function fails when the substring extends beyond the end of the string. 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

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/substring.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/substring.json
```

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.

Parameter	Required	Type	Description
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.

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"]

NAME	TYPE	VALUE
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-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/take.json
```

toLowerCase

`toLowerCase(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": {
        "toLowerCaseOutput": {
            "value": "[toLowerCase(parameters('testString'))]",
            "type": "string"
        },
        "toUpperCaseOutput": {
            "type": "string",
            "value": "[toUpperCase(parameters('testString'))]"
        }
    }
}
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
toLowerCaseOutput	String	one two three
toUpperCaseOutput	String	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/tolower.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri  
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-  
manager/functions/tolower.json
```

toUpperCase

```
toUpperCase(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": "[toLowerCase(parameters('testString'))]",
            "type": "string"
        },
        "toUpperOutput": {
            "type": "string",
            "value": "[toUpperCase(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

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/tolower.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/tolower.json
```

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

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/trim.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri  
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-  
manager/functions/trim.json
```

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 is not a random string, but rather the result of a hash function. The returned value is 13 characters long. It is not 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.

```
tcvhiju5h2o5o
```

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 is not unique if constructed the same way.

```
"resources": [{
    "name": "[concat('storage', uniqueString(resourceGroup().id))]",
    "type": "Microsoft.Storage/storageAccounts",
    ...
}
```

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"
        }
    }
}
```

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/uniquestring.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/uniquestring.json
```

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
toStringOutput	String	http://contoso.com/resources/nested/azuredeploy.json

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/uri.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri  
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-  
manager/functions/uri.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
toStringOutput	String	http://contoso.com/resources/nested/azuredeploy.json

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/uri.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/uri.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

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/uri.json
```

To deploy this example template with PowerShell, use:

```
New-AzureRmResourceGroupDeployment -ResourceGroupName functionexamplegroup -TemplateUri
https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/azure-resource-
manager/functions/uri.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](#).

Manage personal data associated with Azure Resource Manager

5/21/2018 • 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](#).

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-AzureRmResourceGroupDeployment](#)
- [az group deployment list](#)

To delete **deployments** from the history, use:

- [Delete](#)
- [Remove-AzureRmResourceGroupDeployment](#)
- [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-AzureRmResourceGroup](#)
- [az group list](#)

To delete **resource groups**, use:

- [Delete](#)
- [Remove-AzureRmResourceGroup](#)
- [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-AzureRmTag](#)
- [az tag list](#)

To delete **tags**, use:

- [Delete](#)
- [Remove-AzureRmTag](#)
- [az tag delete](#)

Next steps

- For an overview of Azure Resource Manager, see the [What is Resource Manager?](#)

Throttling Resource Manager requests

5/21/2018 • 3 minutes to read • [Edit Online](#)

For each subscription and tenant, Resource Manager limits read requests to 15,000 per hour and write requests to 1,200 per hour. 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 have reached the limit.

When you reach the limit, you receive the HTTP status code **429 Too many requests**.

The number of requests is scoped to either your subscription or your tenant. If you have multiple, concurrent applications making requests in your subscription, the requests from those applications are added together to determine the number of remaining requests.

Subscription scoped requests are ones the involve passing your subscription ID, such as retrieving the resource groups in your subscription. Tenant scoped requests don't include your subscription ID, such as retrieving valid Azure locations.

Remaining requests

You can determine the number of remaining requests by examining response headers. Each request includes values for the number of remaining read and 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.
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).

RESPONSE HEADER	DESCRIPTION
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"]
```

Or, if want to see the remaining requests for debugging, you can provide the **-Debug** parameter on your **PowerShell** cmdlet.

```
Get-AzureRmResourceGroup -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: 14999
```

To get write limits, use a write operation:

```
New-AzureRmResourceGroup -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': '14998'
```

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 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

5/21/2018 • 3 minutes to read • [Edit Online](#)

Some Azure REST operations run asynchronously because the operation cannot be completed quickly. This topic 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 are 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 is not 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, you 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](#).