Contents

- Bicep documentation
- Overview
  - What is Bicep?
  - Compare JSON and Bicep
  - Frequently asked questions
- Install
  - Install Bicep tools
  - Troubleshoot installation
- Quickstarts
  - Create Bicep templates - VS Code
  - Create Bicep templates - Visual Studio
  - Create template specs
  - Use loops
  - Use private module registry
  - CI/CD
    - Bicep with pipelines
    - Bicep with GitHub Actions
    - Bicep with MSBuild
- Tutorials
  - Learn modules for Bicep
  - Learn live - Bicep
- Samples - Azure Services
  - AI + Machine Learning
    - Cognitive Search
    - Cognitive Services
    - Data Science Virtual Machine
    - Machine Learning secure workspace
  - Analytics
    - Analysis Services

- Data Factory
- Event Hubs
- HDInsight - Hadoop
- HDInsight - HBase
- HDInsight - Interactive Query
- HDInsight - Kafka
- HDInsight - Spark
- Stream Analytics
- Synapse Analytics - Dedicated SQL pool

Compute

- Batch
- Functions
- Linux virtual machine
- Windows virtual machine scale sets
- Windows virtual machine

Containers

- AKS
- Container Instances
- Container Registry
- Service Fabric

Databases

- Cache for Redis
- Cosmos DB
- Database for MariaDB
- Database for MySQL
- Database for PostgreSQL
- Database for PostgreSQL - Flexible Server
- Database Migration Service
- SQL Database
- SQL Managed Instance
- SQL Server on Virtual Machines

DevOps

- App Configuration

- DevTest Labs
- Pipelines

Integration

- API for FHIR
- Event Grid
- Logic Apps
- Service Bus

Internet of Things

- IoT Edge
- IoT Hub
- IoT Hub Device Provisioning Service

Management and Governance

- Advisor
- Backup
- Cost Management + Billing
- Microsoft Purview account
- Monitor
- Policy
- Resource Graph
- Role-based access control - custom roles
- Role-based access control - role assignment
- Service Health
- Site Recovery

Media Services

- Media Services account

Networking

- Application Gateway
- Content Delivery Network
- DDoS Protection
- DNS
- ExpressRoute
- Firewall - Availability Zones

- Firewall - IP Groups
- Firewall - multiple addresses
- Firewall Manager - policy
- Firewall Manager - virtual hub
- Front Door
- Front Door - Standard/Premium
- Load Balancer - internal
- Load Balancer - public
- NAT gateway
- Private Endpoint
- Private Link service
- Traffic Manager
- Virtual Network
- Web Application Firewall
- Security
  - Attestation
  - Defender for Cloud
  - Key Vault - key
  - Key Vault - secret
- Storage
  - Data Share
  - Storage account
- Web
  - API Management
  - Notification Hubs
  - Redis Web App
  - SignalR Service
- Concepts
  - Bicep file
  - Data types
  - Parameters (param)
  - Variables (var)

Resources (resource)
Existing resources (existing)
Child resources (parent)
Extension resources (scope)
Dependencies (dependsOn)
Modules (module)
Outputs (output)
Loops (for)
Conditions (if)
Scopes (targetScope)
  Resource group
  Subscription
  Management group
  Tenant
Functions
  All functions
  Any function
  Array functions
  Date functions
  Deployment functions
  File functions
  Lambda functions
  Logical functions
  Numeric functions
  Object functions
  Resource functions
  Scope functions
  String functions
Operators
  All operators
  Accessor operators
  Comparison operators

- Logical operators
- Numeric operators
- How to
  - Author
    - Best practices
    - Visual Studio Code
    - Private module registry
    - Deployment script
    - Template specs
    - Patterns
      - Configuration set
      - Logical parameter
      - Name generation
      - Shared variable file
    - Common scenarios
      - Access control
      - Monitoring
      - Secrets
      - Virtual networks
  - Provide parameters
    - Parameter file
    - Sensitive values
  - Validate file
    - Linter
    - Linter rules
      - Admin user name not literal
      - Artifacts parameters
      - Max outputs
      - Max parameters
      - Max resources
      - Max variables
      - No hardcoded environment URLs

No hard-coded locations
No location expressions outside of parameter default values
No unnecessary dependsOn entries
No unused existing resources
No unused parameters
No unused variables
Outputs should not contain secrets
Prefer interpolation
Prefer unquoted property names
Protect commandToExecute secrets
Secure parameter default
Secure secrets in parameters
Simplify interpolation
Use explicit values for module location parameters
Use stable resource identifier
Use stable VM image
Deploy
What-if check
Deploy - VS Code
Deploy - CLI
Deploy - PowerShell
Deploy - Cloud Shell
Configure settings
Bicep config file
Linter settings
Module settings
Migrate to Bicep
Recommended workflow
Decompile
Contribute to Bicep
Reference
Resource reference

Bicep CLI

Azure PowerShell

Azure CLI

What is Bicep?

9/21/2022 • 4 minutes to read • Edit Online

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. In a Bicep file, you define the infrastructure you want to deploy to Azure, and then use that file throughout the development lifecycle to repeatedly deploy your infrastructure. Your resources are deployed in a consistent manner.

Bicep provides concise syntax, reliable type safety, and support for code reuse. Bicep offers a first-class authoring experience for your infrastructure-as-code solutions in Azure.

Benefits of Bicep

Bicep provides the following advantages:

- **Support for all resource types and API versions**: Bicep immediately supports all preview and GA versions for Azure services. As soon as a resource provider introduces new resources types and API versions, you can use them in your Bicep file. You don't have to wait for tools to be updated before using the new services.
- **Simple syntax**: When compared to the equivalent JSON template, Bicep files are more concise and easier to read. Bicep requires no previous knowledge of programming languages. Bicep syntax is declarative and specifies which resources and resource properties you want to deploy.

  The following examples show the difference between a Bicep file and the equivalent JSON template. Both examples deploy a storage account.

  - Bicep
  - JSON

```
param location string = resourceGroup().location
param storageAccountName string = 'toylaunch${uniqueString(resourceGroup().id)}'

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-06-01' = {
  name: storageAccountName
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
  properties: {
    accessTier: 'Hot'
  }
}
```

- **Authoring experience**: When you use the Bicep Extension for VS Code to create your Bicep files, you get a first-class authoring experience. The editor provides rich type-safety, intellisense, and syntax validation.

```
app > main.bicep
1
2
3
4
5
```

- **Repeatable results**: Repeatedly deploy your infrastructure throughout the development lifecycle and have confidence your resources are deployed in a consistent manner. Bicep files are idempotent, which means you can deploy the same file many times and get the same resource types in the same state. You can develop one file that represents the desired state, rather than developing lots of separate files to represent updates.
- **Orchestration**: You don't have to worry about the complexities of ordering operations. Resource Manager orchestrates the deployment of interdependent resources so they're created in the correct order. When possible, Resource Manager deploys resources in parallel so your deployments finish faster than serial deployments. You deploy the file through one command, rather than through multiple imperative commands.

- **Modularity**: You can break your Bicep code into manageable parts by using modules. The module deploys a set of related resources. Modules enable you to reuse code and simplify development. Add the module to a Bicep file anytime you need to deploy those resources.
- **Integration with Azure services**: Bicep is integrated with Azure services such as Azure Policy, template specs, and Blueprints.
- **Preview changes**: You can use the what-if operation to get a preview of changes before deploying the Bicep file. With what-if, you see which resources will be created, updated, or deleted, and any resource properties that will be changed. The what-if operation checks the current state of your environment and eliminates the need to manage state.
- **No state or state files to manage**: All state is stored in Azure. Users can collaborate and have confidence their updates are handled as expected.
- **No cost and open source**: Bicep is completely free. You don't have to pay for premium capabilities. It's

also supported by Microsoft support.

Get started

To start with Bicep:

1. **Install the tools**. See Set up Bicep development and deployment environments. Or, you can use the VS Code Devcontainer/Codespaces repo to get a pre-configured authoring environment.
2. **Complete the quickstart and the Learn modules for Bicep**.

To decompile an existing ARM template to Bicep, see Decompiling ARM template JSON to Bicep. You can use the Bicep Playground to view Bicep and equivalent JSON side by side.

To learn about the resources that are available in your Bicep file, see Bicep resource reference

Bicep examples can be found in the Bicep GitHub repo

About the language

Bicep isn't intended as a general programming language to write applications. A Bicep file declares Azure resources and resource properties, without writing a sequence of programming commands to create resources.

To track the status of the Bicep work, see the Bicep project repository.

To learn about Bicep, see the following video.

You can use Bicep instead of JSON to develop your Azure Resource Manager templates (ARM templates). The JSON syntax to create an ARM template can be verbose and require complicated expressions. Bicep syntax reduces that complexity and improves the development experience. Bicep is a transparent abstraction over ARM template JSON and doesn't lose any of the JSON template capabilities. During deployment, the Bicep CLI converts a Bicep file into ARM template JSON.

Resource types, API versions, and properties that are valid in an ARM template are valid in a Bicep file.

Bicep offers an easier and more concise syntax when compared to the equivalent JSON. You don't use bracketed expressions `[...]`. Instead, you directly call functions, and get values from parameters and variables. You give each deployed resource a symbolic name, which makes it easy to reference that resource in your template.

For a full comparison of the syntax, see Comparing JSON and Bicep for templates.

Bicep automatically manages dependencies between resources. You can avoid setting `dependsOn` when the symbolic name of a resource is used in another resource declaration.

The structure of the Bicep file is more flexible than the JSON template. You can declare parameters, variables, and outputs anywhere in the file. In JSON, you have to declare all parameters, variables, and outputs within the corresponding sections of the template.

Next steps

Get started with the Quickstart.

For answers to common questions, see Frequently asked questions for Bicep.

Comparing JSON and Bicep for templates

9/21/2022 • 3 minutes to read • Edit Online

This article compares Bicep syntax with JSON syntax for Azure Resource Manager templates (ARM templates). In most cases, Bicep provides syntax that is less verbose than the equivalent in JSON.

If you're familiar with using JSON to develop ARM templates, use the following examples to learn about the equivalent syntax for Bicep.

Compare complete files

The Bicep Playground lets you view Bicep and equivalent JSON side by side. You can compare the implementations of the same infrastructure.

For example, you can view the file to deploy a SQL server and database. The Bicep is about half the size of the ARM template.

Expressions

To author an expression:

```
func()
```

```
"[func()]"
```

Parameters

To declare a parameter with a default value:

```
param orgName string = 'Contoso'
```

```
"parameters": {
  "orgName": {
    "type": "string",
    "defaultValue": "Contoso"
  }
}
```

To get a parameter value, use the name you defined:

```
name: orgName
```

```
"name": "[parameters('orgName'))]"
```

Variables

To declare a variable:

```
var description = 'example value'
```

```
"variables": {
  "description": "example value"
},
```

To get a variable value, use the name you defined:

```
workloadSetting: description
```

```
"workloadSetting": "[variables('description'))]"
```

Strings

To concatenate strings:

```
name: '${namePrefix}-vm'
```

```
"name": "[concat(parameters('namePrefix'), '-vm')]"
```

Logical operators

To return the logical **AND**:

```
isMonday && isNovember
```

```
[and(parameter('isMonday'), parameter('isNovember'))]
```

To conditionally set a value:

```
isMonday ? 'valueIfTrue' : 'valueIfFalse'
```

```
[if(parameters('isMonday'), 'valueIfTrue', 'valueIfFalse')]
```

Deployment scope

To set the target scope of the deployment:

```
targetScope = 'subscription'
```

```
"$schema": "https://schema.management.azure.com/schemas/2018-05-01/subscriptionDeploymentTemplate.json#"
```

Resources

To declare a resource:

```
resource virtualMachine 'Microsoft.Compute/virtualMachines@2020-06-01' = {
  ...
}
```

```
"resources": [
  {
    "type": "Microsoft.Compute/virtualMachines",
    "apiVersion": "2020-06-01",
    ...
  }
]
```

To conditionally deploy a resource:

```
resource virtualMachine 'Microsoft.Compute/virtualMachines@2020-06-01' = if(deployVM) {
  ...
}
```

```
"resources": [
  {
    "condition": "[parameters('deployVM')]",
    "type": "Microsoft.Compute/virtualMachines",
    "apiVersion": "2020-06-01",
    ...
  }
]
```

To set a resource property:

```
sku: '2016-Datacenter'
```

```
"sku": "2016-Datacenter",
```

To get the resource ID of a resource in the template:

```
nic1.id
```

```
[resourceId('Microsoft.Network/networkInterfaces', variables('nic1Name'))]
```

Loops

To iterate over items in an array or count:

```
[for storageName in storageAccountNames: {
  ...
}]
```

```
"copy": {
  "name": "storagecopy",
  "count": "[length(parameters('storageAccountNames'))]"
},
...
```

Resource dependencies

For Bicep, you can set an explicit dependency but this approach isn't recommended. Instead, rely on implicit dependencies. An implicit dependency is created when one resource declaration references the identifier of another resource.

The following shows a network interface with an implicit dependency on a network security group. It references the network security group with `netSecurityGroup.id`.

```
resource netSecurityGroup 'Microsoft.Network/networkSecurityGroups@2020-06-01' = {
  ...
}

resource nic1 'Microsoft.Network/networkInterfaces@2020-06-01' = {
  name: nic1Name
  location: location
  properties: {
    ...
    networkSecurityGroup: {
      id: netSecurityGroup.id
    }
  }
}
```

If you must set an explicit dependence, use:

```
dependsOn: [ storageAccount ]
```

```
"dependsOn": ["[resourceId('Microsoft.Storage/storageAccounts', 'parameters('storageAccountName'))]"]
```

Reference resources

To get a property from a resource in the template:

```
storageAccount.properties.primaryEndpoints.blob
```

```
[reference(resourceId('Microsoft.Storage/storageAccounts',
variables('storageAccountName'))).primaryEndpoints.blob]
```

To get a property from an existing resource that isn't deployed in the template:

```
resource storageAccount 'Microsoft.Storage/storageAccounts@2019-06-01' existing = {
  name: storageAccountName
}

// use later in template as often as needed
storageAccount.properties.primaryEndpoints.blob
```

```
// required every time the property is needed
"[reference(resourceId('Microsoft.Storage/storageAccounts/', parameters('storageAccountName')), '2019-06-
01').primaryEndpoints.blob]"
```

In Bicep, use the nested accessor (`::`) to get a property on a resource nested within a parent resource:

```
VNet1::Subnet1.properties.addressPrefix
```

For JSON, use reference function:

```
[reference(resourceId('Microsoft.Network/virtualNetworks/subnets',
variables('subnetName'))).properties.addressPrefix]
```

Outputs

To output a property from a resource in the template:

```
output hostname string = publicIP.properties.dnsSettings.fqdn
```

```
"outputs": {
  "hostname": {
    "type": "string",
    "value": "[reference(resourceId('Microsoft.Network/publicIPAddresses',
variables('publicIPAddressName'))).dnsSettings.fqdn]"
  },
}
```

To conditionally output a value:

```
output hostname string = condition ? publicIP.properties.dnsSettings.fqdn : ''
```

```
"outputs": {
  "hostname": {
    "condition": "[variables('condition')]",
    "type": "string",
    "value": "[reference(resourceId('Microsoft.Network/publicIPAddresses',
variables('publicIPAddressName'))).dnsSettings.fqdn]"
  }
}
```

The Bicep ternary operator is the equivalent to the if function in an ARM template JSON, not the condition property. The ternary syntax has to evaluate to one value or the other. If the condition is false in the preceding samples, Bicep outputs a hostname with an empty string, but JSON outputs no values.

Code reuse

To separate a solution into multiple files:

- For Bicep, use modules.
- For ARM templates, use linked templates.

Next steps

- For information about the Bicep, see Bicep quickstart.
- To learn about converting templates between the languages, see Converting ARM templates between JSON and Bicep.

Install Bicep tools

9/21/2022 • 5 minutes to read • Edit Online

Let's make sure your environment is set up for working with Bicep files. To author and deploy Bicep files, we recommend any of the following options:

TASKS	OPTIONS	BICEP CLI INSTALLATION
Author	VS Code and Bicep extension	automatic
Deploy	Azure CLI	automatic
	Azure PowerShell	manual
	VS Code and Bicep extension	automatic
	Air-gapped cloud	download

VS Code and Bicep extension

To create Bicep files, you need a good Bicep editor. We recommend:

- **Visual Studio Code** - If you don't already have Visual Studio Code, install it.
- **Bicep extension for Visual Studio Code**. Visual Studio Code with the Bicep extension provides language support and resource autocompletion. The extension helps you create and validate Bicep files.

  To install the extension, search for *bicep* in the **Extensions** tab or in the Visual Studio marketplace.

  Select **Install**.

To verify you've installed the extension, open any file with the `.bicep` file extension. You should see the language mode in the lower right corner change to **Bicep**.

If you get an error during installation, see Troubleshoot Bicep installation.

You can deploy your Bicep files directly from the VS Code editor. For more information, see Deploy Bicep files from Visual Studio Code.

Azure CLI

When you use Azure CLI with Bicep, you have everything you need to deploy and decompile Bicep files. Azure CLI automatically installs the Bicep CLI when a command is executed that needs it.

You must have Azure CLI version **2.20.0 or later** installed. To install or update Azure CLI, see:

- Install Azure CLI on Windows
- Install Azure CLI on Linux
- Install Azure CLI on macOS

To verify your current version, run:

```
az --version
```

To validate your Bicep CLI installation, use:

```
az bicep version
```

To upgrade to the latest version, use:

```
az bicep upgrade
```

For more commands, see Bicep CLI.

> **IMPORTANT**
> Azure CLI installs a self-contained instance of the Bicep CLI. This instance doesn't conflict with any versions you may have manually installed. Azure CLI doesn't add Bicep CLI to your PATH.

You're done with setting up your Bicep environment. The rest of this article describes installation steps that you don't need when using Azure CLI.

Azure PowerShell

You must have Azure PowerShell version **5.6.0 or later** installed. To update or install, see Install Azure PowerShell.

Azure PowerShell doesn't automatically install the Bicep CLI. Instead, you must manually install the Bicep CLI.

> **IMPORTANT**
> The self-contained instance of the Bicep CLI installed by Azure CLI isn't available to PowerShell commands. Azure PowerShell deployments fail if you haven't manually installed the Bicep CLI.

When you manually install the Bicep CLI, run the Bicep commands with the `bicep` syntax, instead of the `az bicep` syntax for Azure CLI.

To check your Bicep CLI version, run:

```
bicep --version
```

Install manually

The following methods install the Bicep CLI and add it to your PATH. You must manually install for any use other than Azure CLI.

When installing manually, select a location that is different than the one managed by Azure CLI. All of the following examples use a location named **bicep** or **.bicep**. This location won't conflict with the location managed by Azure CLI, which uses **.azure**.

- Linux
- macOS
- Windows

Linux

```
# Fetch the latest Bicep CLI binary
curl -Lo bicep https://github.com/Azure/bicep/releases/latest/download/bicep-linux-x64
# Mark it as executable
chmod +x ./bicep
# Add bicep to your PATH (requires admin)
sudo mv ./bicep /usr/local/bin/bicep
# Verify you can now access the 'bicep' command
bicep --help
# Done!
```

> **NOTE**
> For lightweight Linux distributions like Alpine, use **bicep-linux-musl-x64** instead of **bicep-linux-x64** in the preceding script.

macOS

Via homebrew

```
# Add the tap for bicep
brew tap azure/bicep

# Install the tool
brew install bicep
```

Via BASH

```
# Fetch the latest Bicep CLI binary
curl -Lo bicep https://github.com/Azure/bicep/releases/latest/download/bicep-osx-x64
# Mark it as executable
chmod +x ./bicep
# Add Gatekeeper exception (requires admin)
sudo spctl --add ./bicep
# Add bicep to your PATH (requires admin)
sudo mv ./bicep /usr/local/bin/bicep
# Verify you can now access the 'bicep' command
bicep --help
# Done!
```

Windows

Windows Installer

Download and run the latest Windows installer. The installer doesn't require administrative privileges. After the installation, Bicep CLI is added to your user PATH. Close and reopen any open command shell windows for the PATH change to take effect.

Chocolatey

```
choco install bicep
```

Winget

```
winget install -e --id Microsoft.Bicep
```

Manual with PowerShell

```
# Create the install folder
$installPath = "$env:USERPROFILE\.bicep"
$installDir = New-Item -ItemType Directory -Path $installPath -Force
$installDir.Attributes += 'Hidden'
# Fetch the latest Bicep CLI binary
(New-Object Net.WebClient).DownloadFile("https://github.com/Azure/bicep/releases/latest/download/bicep-win-
x64.exe", "$installPath\bicep.exe")
# Add bicep to your PATH
$currentPath = (Get-Item -path "HKCU:\Environment" ).GetValue('Path', '', 'DoNotExpandEnvironmentNames')
if (-not $currentPath.Contains("%USERPROFILE%\.bicep")) { setx PATH ($currentPath + ";%USERPROFILE%\.bicep")
}
if (-not $env:path.Contains($installPath)) { $env:path += ";$installPath" }
# Verify you can now access the 'bicep' command.
bicep --help
# Done!
```

Install on air-gapped cloud

The `bicep install` and `bicep upgrade` commands don't work in an air-gapped environment. To install Bicep CLI in an air-gapped environment, you need to download the Bicep CLI executable manually and save it to **.azure/bin**. This location is where the instance managed by Azure CLI is installed.

- **Linux**
  1. Download **bicep-linux-x64** from the Bicep release page in a non-air-gapped environment.
  2. Copy the executable to the **$HOME/.azure/bin** directory on an air-gapped machine. Rename file to **bicep**.
- **macOS**
  1. Download **bicep-osx-x64** from the Bicep release page in a non-air-gapped environment.
  2. Copy the executable to the **$HOME/.azure/bin** directory on an air-gapped machine. Rename file to **bicep**.
- **Windows**
  1. Download **bicep-win-x64.exe** from the Bicep release page in a non-air-gapped environment.
  2. Copy the executable to the **%UserProfile%/.azure/bin** directory on an air-gapped machine. Rename file to **bicep.exe**.

Install the nightly builds

If you'd like to try the latest pre-release bits of Bicep before they're released, see Install nightly builds.

> **WARNING**
>
> These pre-release builds are much more likely to have known or unknown bugs.

Next steps

For more information about using Visual Studio Code and the Bicep extension, see Quickstart: Create Bicep files with Visual Studio Code.

If you have problems with your Bicep installation, see Troubleshoot Bicep installation.

To deploy Bicep files from an Azure Pipeline, see Integrate Bicep with Azure Pipelines. To deploy Bicep files through GitHub Actions, see Deploy Bicep files by using GitHub Actions.

Troubleshoot Bicep installation

9/21/2022 • 2 minutes to read • Edit Online

This article describes how to resolve potential errors in your Bicep installation.

.NET runtime error

When installing the Bicep extension for Visual Studio Code, you may run into the following error messages:

```
Failed to install .NET runtime v5.0
```

```
Failed to download .NET 5.0.x ....... Error!
```

To solve the problem, you can manually install .NET from the .NET website, and then configure Visual Studio Code to reuse an existing installation of .NET with the following settings:

Windows

```
"dotnetAcquisitionExtension.existingDotnetPath": [
  {
    "extensionId": "ms-azuretools.vscode-bicep",
    "path": "C:\\Program Files\\dotnet\\dotnet.exe"
  }
]
```

macOS

If you need an **x64** installation, use:

```
"dotnetAcquisitionExtension.existingDotnetPath": [
  {
    "extensionId": "ms-azuretools.vscode-bicep",
    "path": "/usr/local/share/dotnet/x64/dotnet"
  }
]
```

For other **macOS** installations, use:

```
"dotnetAcquisitionExtension.existingDotnetPath": [
  {
    "extensionId": "ms-azuretools.vscode-bicep",
    "path": "/usr/local/share/dotnet/dotnet"
  }
]
```

See User and Workspace Settings for configuring Visual Studio Code settings.

Visual Studio Code error

If you see the following error message popup in VSCode:

```
The Bicep server crashed 5 times in the last 3 minutes. The server will not be restarted.
```

From VSCode, open the **Output** view in the pane at the bottom of the screen, and then select **Bicep**:

If you see the following output in the pane, and you are using Bicep CLI **version 0.4.1124** or later, check whether you have added the `dotnetAcquisitionExtension.existingDotnetPath` configuration option to VSCode. See .NET runtime error. If this configuration option is present, remove it and restart VSCode.

```
It was not possible to find any compatible framework version.
```

Otherwise, raise an issue in the Bicep repo, and include the output messages.

Multiple versions of Bicep CLI installed

If you manually install the Bicep CLI to more than one location, you may notice unexpected behavior such as the Bicep CLI not updating when you run the upgrade command. Or, you may notice that running `az bicep version` returns one version, but `bicep --version` returns a different version.

To resolve this issue, you can either update all locations, or select one location to maintain and delete the other locations.

First, open your command prompt (not PowerShell), and run `where bicep`. This command returns the locations of your Bicep installations. If you're using the instance of Bicep CLI that is managed by Azure CLI, you won't see this installation because it's not added to the PATH. If `where bicep` returns only one location, it may be that the conflicting versions you're seeing is between the manual installation and the Azure CLI installation.

To **keep all installation locations**, use the same method you used earlier to manually install the Bicep CLI for all locations you want to maintain. If you're using Azure CLI, run `az bicep upgrade` to update that version.

To **keep only one installation location**, use the following steps:

1. Delete the files for the installations you don't want to keep.
2. Remove those locations from your **PATH** environment variable.

If you have both a **manual installation and the instance managed by Azure CLI**, you can combine your usage to one instance.

1. Delete the manual installation location.
2. Add the location of the Bicep CLI installed by Azure CLI to the **PATH** variable. For Windows, the location maintained by Azure CLI is `%USERPROFILE%\.Azure\bin`.

After adding the Azure CLI instance to the PATH, you can use that version with either `az bicep` or `bicep`.

Next steps

For more information about using Visual Studio Code and the Bicep extension, see Quickstart: Create Bicep files with Visual Studio Code.

Quickstart: Create Bicep files with Visual Studio Code

9/21/2022 • 5 minutes to read • Edit Online

This quickstart guides you through the steps to create a Bicep file with Visual Studio Code. You'll create a storage account and a virtual network. You'll also learn how the Bicep extension simplifies development by providing type safety, syntax validation, and autocompletion.

Prerequisites

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

To set up your environment for Bicep development, see Install Bicep tools. After completing those steps, you'll have Visual Studio Code and the Bicep extension. You also have either the latest Azure CLI or the latest Azure PowerShell module.

Add resource snippet

Launch Visual Studio Code and create a new file named *main.bicep*.

VS Code with the Bicep extension simplifies development by providing pre-defined snippets. In this quickstart, you'll add a snippet that creates a virtual network.

In *main.bicep*, type **vnet**. Select **res-vnet** from the list, and then Tab or Enter.

> **TIP**
> If you don't see those intellisense options in VS Code, make sure you've installed the Bicep extension as specified in Prerequisites. If you have installed the extension, give the Bicep language service some time to start after opening your Bicep file. It usually starts quickly, but you will not have intellisense options until it starts. A notification in the lower right corner indicates that the service is starting. When that notification disappears, the service is running.

Your Bicep file now contains the following code:

```
resource virtualNetwork 'Microsoft.Network/virtualNetworks@2019-11-01' = {
  name: 'name'
  location: resourceGroup().location
  properties: {
    addressSpace: {
      addressPrefixes: [
        '10.0.0.0/16'
      ]
    }
    subnets: [
      {
        name: 'Subnet-1'
        properties: {
          addressPrefix: '10.0.0.0/24'
        }
      }
      {
        name: 'Subnet-2'
        properties: {
          addressPrefix: '10.0.1.0/24'
        }
      }
    ]
  }
}
```

This snippet contains all of the values you need to define a virtual network. However, you can modify this code to meet your requirements. For example, `name` isn't a great name for the virtual network. Change the `name` property to `examplevnet`.

```
name: 'examplevnet'
```

You could deploy this Bicep file, but we'll add a parameter and storage account before deploying.

Add parameter

Now, we'll add a parameter for the storage account name. At the top of file, add:

```
param storageName
```

When you add a space after **storageName**, notice that intellisense offers the data types that are available for the parameter. Select **string**.

You have the following parameter:

```
param storageName string
```

This parameter works fine, but storage accounts have limits on the length of the name. The name must have at least 3 characters and no more than 24 characters. You can specify those requirements by adding decorators to the parameter.

Add a line above the parameter, and type **@**. You see the available decorators. Notice there are decorators for both **minLength** and **maxLength**.

Add both decorators and specify the character limits, as shown below:

```
@minLength(3)
@maxLength(24)
param storageName string
```

You can also add a description for the parameter. Include information that helps people deploying the Bicep file understand the value to provide.

```
@minLength(3)
@maxLength(24)
@description('Provide a name for the storage account. Use only lower case letters and numbers. The name must be unique across Azure.')
param storageName string
```

Your parameter is ready to use.

Add resource

Instead of using a snippet to define the storage account, we'll use intellisense to set the values. Intellisense makes this step much easier than having to manually type the values.

To define a resource, use the `resource` keyword. Below your virtual network, type **resource exampleStorage**:

```
resource exampleStorage
```

exampleStorage is a symbolic name for the resource you're deploying. You can use this name to reference the resource in other parts of your Bicep file.

When you add a space after the symbolic name, a list of resource types is displayed. Continue typing **storage** until you can select it from the available options.

```
21          {
22            name: 'Subnet-2'
23            properties: {
24              addressPrefix: '10.0.1.0/24'
25            }
26          }
27        ]
28      }
29    }
30
31    resource exampleStorage sto
32
```

After selecting **Microsoft.Storage/storageAccounts**, you're presented with the available API versions. Select **2021-02-01**.

```
resource exampleStorage 'Microsoft.Storage/storageAccounts@'
```

After the single quote for the resource type, add `=` and a space. You're presented with options for adding properties to the resource. Select **required-properties**.

```
resource exampleStorage 'Microsoft.Storage/storageAccounts@2021-02-01' =
```

This option adds all of the properties for the resource type that are required for deployment. After selecting this

option, your storage account has the following properties:

```
resource exampleStorage 'Microsoft.Storage/storageAccounts@2021-02-01' = {
  name:
  location:
  sku: {
    name:
  }
  kind:

}
```

You're almost done. Just provide values for those properties.

Again, intellisense helps you. Set `name` to `storageName`, which is the parameter that contains a name for the storage account. For `location`, set it to `'eastus'`. When adding SKU name and kind, intellisense presents the valid options.

When you've finished, you have:

```
@minLength(3)
@maxLength(24)
param storageName string

resource virtualNetwork 'Microsoft.Network/virtualNetworks@2019-11-01' = {
  name: 'examplevnet'
  location: resourceGroup().location
  properties: {
    addressSpace: {
      addressPrefixes: [
        '10.0.0.0/16'
      ]
    }
    subnets: [
      {
        name: 'Subnet-1'
        properties: {
          addressPrefix: '10.0.0.0/24'
        }
      }
      {
        name: 'Subnet-2'
        properties: {
          addressPrefix: '10.0.1.0/24'
        }
      }
    ]
  }
}

resource exampleStorage 'Microsoft.Storage/storageAccounts@2021-02-01' = {
  name: storageName
  location: 'eastus'
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
}
```

For more information about the Bicep syntax, see Bicep structure.

Visualize resources

You can view a representation of the resources in your file.

From the upper right corner, select the visualizer button to open the Bicep Visualizer.

The visualizer shows the resources defined in the Bicep file with the resource dependency information. The two resources defined in this quickstart don't have dependency relationship, so you don't see a connector between the two resources.

Deploy the Bicep file

1. Right-click the Bicep file inside the VSCode, and then select **Deploy Bicep file**.

2. From the **Select Resource Group** listbox on the top, select **Create new Resource Group**.

3. Enter **exampleRG** as the resource group name, and then press **[ENTER]**.

4. Select a location for the resource group, and then press **[ENTER]**.

5. From **Select a parameter file**, select **None**.

6. Enter a unique storage account name, and then press **[ENTER]**. If you get an error message indicating the

storage account is already taken, the storage name you provided is in use. Provide a name that is more likely to be unique.

7. From **Create parameters file from values used in this deployment?**, select **No**.

It takes a few moments to create the resources. For more information, see Deploy Bicep files with visual Studio Code.

You can also deploy the Bicep file by using Azure CLI or Azure PowerShell:

- CLI
- PowerShell

```
az group create --name exampleRG --location eastus

az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
storageName=uniquename
```

When the deployment finishes, you should see a message indicating the deployment succeeded.

Clean up resources

When the Azure resources are no longer needed, use the Azure CLI or Azure PowerShell module to delete the quickstart resource group.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

Learn modules for Bicep

Quickstart: Create Bicep files with Visual Studio

9/21/2022 • 4 minutes to read • Edit Online

This quickstart guides you through the steps to create a Bicep file with Visual Studio. You'll create a storage account and a virtual network. You'll also learn how the Bicep extension simplifies development by providing type safety, syntax validation, and autocompletion.

Prerequisites

- Azure Subscription. If you don't have an Azure subscription, create a free account before you begin.
- Visual Studio version 17.3.0 preview 3 or newer. See Visual Studio Preview.
- Visual Studio Bicep extension. See Visual Studio Marketplace.
- Bicep file deployment requires either the latest Azure CLI or the latest Azure PowerShell module.

Add resource snippet

Launch Visual Studio and create a new file named **main.bicep**.

Visual Studio with the Bicep extension simplifies development by providing pre-defined snippets. In this quickstart, you'll add a snippet that creates a virtual network.

In *main.bicep*, type **vnet**. Select **res-vnet** from the list, and then press **[TAB]** or **[ENTER]**.

> **TIP**
> If you don't see those intellisense options in Visual Studio, make sure you've installed the Bicep extension as specified in Prerequisites. If you have installed the extension, give the Bicep language service some time to start after opening your Bicep file. It usually starts quickly, but you will not have intellisense options until it starts.

Your Bicep file now contains the following code:

```
resource virtualNetwork 'Microsoft.Network/virtualNetworks@2019-11-01' = {
  name: 'name'
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        '10.0.0.0/16'
      ]
    }
    subnets: [
      {
        name: 'Subnet-1'
        properties: {
          addressPrefix: '10.0.0.0/24'
        }
      }
      {
        name: 'Subnet-2'
        properties: {
          addressPrefix: '10.0.1.0/24'
        }
      }
    ]
  }
}
```

This snippet contains all of the values you need to define a virtual network. However, you can modify this code to meet your requirements. For example, `name` isn't a great name for the virtual network. Change the `name` property to `exampleVnet`.

```
name: 'exampleVnet'
```

Notice **location** has a red curly underline. This indicates a problem. Hover your cursor over **location**. The error message is - *The name "location" doesn't exist in the current context.* We'll create a location parameter in the next section.

Add parameters

Now, we'll add two parameters for the storage account name and the location. At the top of file, add:

```
param storageName
```

When you add a space after **storageName**, notice that intellisense offers the data types that are available for the parameter. Select **string**.

You have the following parameter:

```
param storageName string
```

This parameter works fine, but storage accounts have limits on the length of the name. The name must have at least 3 characters and no more than 24 characters. You can specify those requirements by adding decorators to the parameter.

Add a line above the parameter, and type **@**. You see the available decorators. Notice there are decorators for both **minLength** and **maxLength**.

Add both decorators and specify the character limits, as shown below:

```
@minLength(3)
@maxLength(24)
param storageName string
```

You can also add a description for the parameter. Include information that helps people deploying the Bicep file understand the value to provide.

```
@minLength(3)
@maxLength(24)
@description('Provide a name for the storage account. Use only lower case letters and numbers. The name must be unique across Azure.')
param storageName string
```

The storage account name parameter is ready to use.

Add another location parameter:

```
param location string = resourceGroup().location
```

Add resource

Instead of using a snippet to define the storage account, we'll use intellisense to set the values. Intellisense makes this step much easier than having to manually type the values.

To define a resource, use the `resource` keyword. Below your virtual network, type **resource exampleStorage**:

```
resource exampleStorage
```

exampleStorage is a symbolic name for the resource you're deploying. You can use this name to reference the resource in other parts of your Bicep file.

When you add a space after the symbolic name, a list of resource types is displayed. Continue typing **storage** until you can select it from the available options.

After selecting **Microsoft.Storage/storageAccounts**, you're presented with the available API versions. Select **2021-09-01** or the latest API version. We recommend using the latest API version.

After the single quote for the resource type, add `=` and a space. You're presented with options for adding properties to the resource. Select **required-properties**.

This option adds all of the properties for the resource type that are required for deployment. After selecting this option, your storage account has the following properties:

```
resource exampleStorage 'Microsoft.Storage/storageAccounts@2021-09-01' = {
 name: 1
 location: 2
 sku: {
  name: 3
 }
 kind: 4
}
```

There are four placeholders in the code. Use [TAB] to go through them and enter the values. Again, intellisense helps you. Set `name` to **storageName**, which is the parameter that contains a name for the storage account. For

`location`, set it to `location`. When adding SKU name and kind, intellisense presents the valid options.

When you've finished, you have:

```
@minLength(3)
@maxLength(24)
@description('Provide a name for the storage account. Use only lower case letters and numbers. The name must be unique across Azure.')
param storageName string
param location string = resourceGroup().location

resource virtualNetwork 'Microsoft.Network/virtualNetworks@2019-11-01' = {
  name: storageName
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        '10.0.0.0/16'
      ]
    }
    subnets: [
      {
        name: 'Subnet-1'
        properties: {
          addressPrefix: '10.0.0.0/24'
        }
      }
      {
        name: 'Subnet-2'
        properties: {
          addressPrefix: '10.0.1.0/24'
        }
      }
    ]
  }
}

resource exampleStorage 'Microsoft.Storage/storageAccounts@2021-09-01' = {
  name: storageName
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
}
```

For more information about the Bicep syntax, see Bicep structure.

Deploy the Bicep file

Bicep file deployment can't be done from Visual Studio yet. You can deploy the Bicep file by using Azure CLI or Azure PowerShell:

- CLI
- PowerShell

```
az group create --name exampleRG --location eastus

az deployment group create --resource-group exampleRG --template-file main.bicep --parameters storageName=uniquename
```

When the deployment finishes, you should see a message indicating the deployment succeeded.

Clean up resources

When the Azure resources are no longer needed, use the Azure CLI or Azure PowerShell module to delete the quickstart resource group.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

Learn modules for Bicep

Quickstart: Create and deploy a template spec with Bicep

9/21/2022 • 9 minutes to read • Edit Online

This quickstart describes how to create and deploy a template spec with a Bicep file. A template spec is deployed to a resource group so that people in your organization can deploy resources in Microsoft Azure. Template specs let you share deployment templates without needing to give users access to change the Bicep file. This template spec example uses a Bicep file to deploy a storage account.

When you create a template spec, the Bicep file is transpiled into JavaScript Object Notation (JSON). The template spec uses JSON to deploy Azure resources. Currently, you can't use the Microsoft Azure portal to import a Bicep file and create a template spec resource.

Prerequisites

- An Azure account with an active subscription. Create an account for free.
- Azure PowerShell version 6.3.0 or later or Azure CLI version 2.27.0 or later.
- Visual Studio Code with the Bicep extension.

Create Bicep file

You create a template spec from a local Bicep file. Copy the following sample and save it to your computer as *main.bicep*. The examples use the path *C:\templates\main.bicep*. You can use a different path, but you'll need to change the commands.

The following Bicep file is used in the **PowerShell** and **CLI** tabs. The **Bicep file** tab uses a different template that combines Bicep and JSON to create and deploy a template spec.

```
@allowed([
  'Premium_LRS'
  'Premium_ZRS'
  'Standard_GRS'
  'Standard_GZRS'
  'Standard_LRS'
  'Standard_RAGRS'
  'Standard_RAGZRS'
  'Standard_ZRS'
])
@description('Storage account type.')
param storageAccountType string = 'Standard_LRS'

@description('Location for all resources.')
param location string = resourceGroup().location

var storageAccountName = 'storage${uniqueString(resourceGroup().id)}'

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-08-01' = {
  name: storageAccountName
  location: location
  sku: {
    name: storageAccountType
  }
  kind: 'StorageV2'
  properties: {}
}

output storageAccountNameOutput string = storageAccount.name
```

Create template spec

The template spec is a resource type named Microsoft.Resources/templateSpecs. To create a template spec, use Azure CLI, Azure PowerShell, or a Bicep file.

This example uses the resource group name `templateSpecRG`. You can use a different name, but you'll need to change the commands.

- PowerShell
- CLI
- Bicep file

1. Create a new resource group to contain the template spec.

```
New-AzResourceGroup `
  -Name templateSpecRG `
  -Location westus2
```

2. Create the template spec in that resource group. Give the new template spec the name *storageSpec*.

```
New-AzTemplateSpec `
  -Name storageSpec `
  -Version "1.0" `
  -ResourceGroupName templateSpecRG `
  -Location westus2 `
  -TemplateFile "C:\templates\main.bicep"
```

Deploy template spec

Use the template spec to deploy a storage account. This example uses the resource group name `storageRG`. You can use a different name, but you'll need to change the commands.

- PowerShell
- CLI
- Bicep file

1. Create a resource group to contain the new storage account.

```
New-AzResourceGroup `
  -Name storageRG `
  -Location westus2
```

2. Get the resource ID of the template spec.

```
$id = (Get-AzTemplateSpec -ResourceGroupName templateSpecRG -Name storageSpec -Version
"1.0").Versions.Id
```

3. Deploy the template spec.

```
New-AzResourceGroupDeployment `
  -TemplateSpecId $id `
  -ResourceGroupName storageRG
```

4. You provide parameters exactly as you would for a Bicep file deployment. Redeploy the template spec with a parameter for the storage account type.

```
New-AzResourceGroupDeployment `
  -TemplateSpecId $id `
  -ResourceGroupName storageRG `
  -storageAccountType Standard_GRS
```

Grant access

If you want to let other users in your organization deploy your template spec, you need to grant them read access. You can assign the Reader role to an Azure AD group for the resource group that contains template specs you want to share. For more information, see Tutorial: Grant a group access to Azure resources using Azure PowerShell.

Update Bicep file

After the template spec was created, you decided to update the Bicep file. To continue with the examples in the **PowerShell** or **CLI** tabs, copy the sample and replace your *main.bicep* file.

The parameter `storageNamePrefix` specifies a prefix value for the storage account name. The `storageAccountName` variable concatenates the prefix with a unique string.

```
@allowed([
  'Premium_LRS'
  'Premium_ZRS'
  'Standard_GRS'
  'Standard_GZRS'
  'Standard_LRS'
  'Standard_RAGRS'
  'Standard_RAGZRS'
  'Standard_ZRS'
])
@description('Storage account type.')
param storageAccountType string = 'Standard_LRS'

@description('Location for all resources.')
param location string = resourceGroup().location

@maxLength(11)
@description('The storage account name prefix.')
param storageNamePrefix string = 'storage'

var storageAccountName = '${toLower(storageNamePrefix)}${uniqueString(resourceGroup().id)}'

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-08-01' = {
  name: storageAccountName
  location: location
  sku: {
    name: storageAccountType
  }
  kind: 'StorageV2'
  properties: {}
}

output storageAccountNameOutput string = storageAccount.name
```

Update template spec version

Rather than create a new template spec for the revised template, add a new version named `2.0` to the existing template spec. Users can choose to deploy either version.

- PowerShell
- CLI
- Bicep file

1. Create a new version of the template spec.

```
New-AzTemplateSpec `
  -Name storageSpec `
  -Version "2.0" `
  -ResourceGroupName templateSpecRG `
  -Location westus2 `
  -TemplateFile "C:\templates\main.bicep"
```

2. To deploy the new version, get the resource ID for the `2.0` version.

```
$id = (Get-AzTemplateSpec -ResourceGroupName templateSpecRG -Name storageSpec -Version
"2.0").Versions.Id
```

3. Deploy the new version and use the `storageNamePrefix` to specify a prefix for the storage account name.

```
New-AzResourceGroupDeployment `
  -TemplateSpecId $id `
  -ResourceGroupName storageRG `
  -storageNamePrefix "demo"
```

Clean up resources

To clean up the resources you deployed in this quickstart, delete both resource groups. The resource group, template specs, and storage accounts will be deleted.

Use Azure PowerShell or Azure CLI to delete the resource groups.

```
Remove-AzResourceGroup -Name "templateSpecRG"

Remove-AzResourceGroup -Name "storageRG"
```

```
az group delete --name templateSpecRG

az group delete --name storageRG
```

Next steps

Azure Resource Manager template specs in Bicep

Quickstart: Create multiple resource instances in Bicep

9/21/2022 • 4 minutes to read • Edit Online

Learn how to use different `for` syntaxes to create multiple resource instances in Bicep. Even though this article only shows creating multiple resource instances, the same methods can be used to define copies of module, variable, property, or output. To learn more, see Bicep loops.

This article contains the following topics:

- use integer index
- use array elements
- use array and index
- use dictionary object
- loop with condition

Prerequisites

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

To set up your environment for Bicep development, see Install Bicep tools. After completing those steps, you'll have Visual Studio Code and the Bicep extension. You also have either the latest Azure CLI or the latest Azure PowerShell module.

Create a single instance

In this section, you define a Bicep file for creating a storage account, and then deploy the Bicep file. The subsequent sections provide the Bicep samples for different `for` syntaxes. You can use the same deployment method to deploy and experiment those samples. If your deployment fails, it is likely one of the two causes:

- The storage account name is too long. Storage account names must be between 3 and 24 characters in length and may contain numbers and lowercase letters only.
- The storage account name is not unique. Your storage account name must be unique within Azure.

The following Bicep file defines one storage account:

```
param rgLocation string = resourceGroup().location

resource createStorage 'Microsoft.Storage/storageAccounts@2021-06-01' = {
  name: 'storage${uniqueString(resourceGroup().id)}'
  location: rgLocation
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
}
```

Save the Bicep file locally, and then use Azure CLI or Azure PowerShell to deploy the Bicep file:

- CLI
- PowerShell

```
resourceGroupName = "{provide-a-resource-group-name}"
templateFile="{provide-the-path-to-the-bicep-file}"

az group create --name $resourceGroupName --location eastus

az deployment group create --resource-group $resourceGroupName --template-file $templateFile
```

Use integer index

A for loop with an index is used in the following sample to create two storage accounts:

```
param rgLocation string = resourceGroup().location
param storageCount int = 2

resource createStorages 'Microsoft.Storage/storageAccounts@2021-06-01' = [for i in range(0, storageCount): {
  name: '${i}storage${uniqueString(resourceGroup().id)}'
  location: rgLocation
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
}]

output names array = [for i in range(0,storageCount) : {
  name: createStorages[i].name
} ]
```

The index number is used as a part of the storage account name. After deploying the Bicep file, you get two storage accounts that are similar to:

Inside range(), the first number is the starting number, and the second number is the number of times the loop will run. So if you change it to **range(3,2)**, you will also get two storage accounts:

The output of the preceding sample shows how to reference the resources created in a loop. The output is similar to:

```
"outputs": {
  "names": {
    "type": "Array",
    "value": [
      {
        "name": "0storage52iyjssggmvue"
      },
      {
        "name": "1storage52iyjssggmvue"
      }
    ]
  }
},
```

Use array elements

You can loop through an array. The following sample shows an array of strings.

```
param rgLocation string = resourceGroup().location
param storageNames array = [
  'contoso'
  'fabrikam'
]

resource createStorages 'Microsoft.Storage/storageAccounts@2021-06-01' = [for name in storageNames: {
  name: '${name}str${uniqueString(resourceGroup().id)}'
  location: rgLocation
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
}]
```

The loop uses all the strings in the array as a part of the storage account names. In this case, it creates two storage accounts:

You can also loop through an array of objects. The loop not only customizes the storage account names, but also configures the SKUs.

```
param rgLocation string = resourceGroup().location
param storages array = [
  {
    name: 'contoso'
    skuName: 'Standard_LRS'
  }
  {
    name: 'fabrikam'
    skuName: 'Premium_LRS'
  }
]

resource createStorages 'Microsoft.Storage/storageAccounts@2021-06-01' = [for storage in storages: {
  name: '${storage.name}obj${uniqueString(resourceGroup().id)}'
  location: rgLocation
  sku: {
    name: storage.skuName
  }
  kind: 'StorageV2'
}]
```

The loop creates two storage accounts. The SKU of the storage account with the name starting with **fabrikam** is **Premium_LRS**.

Use array and index

In same cases, you might want to combine an array loop with an index loop. The following sample shows how to use the array and the index number for the naming convention.

```
param rgLocation string = resourceGroup().location
param storageNames array = [
  'contoso'
  'fabrikam'
]

resource createStorages 'Microsoft.Storage/storageAccounts@2021-06-01' = [for (name, i) in storageNames: {
  name: '${i}${name}${uniqueString(resourceGroup().id)}'
  location: rgLocation
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
}]
```

After deploying the preceding sample, you create two storage accounts that are similar to:

Use dictionary object

To iterate over elements in a dictionary object, use the items function, which converts the object to an array. Use the `value` property to get properties on the objects.

```
param rgLocation string = resourceGroup().location

param storageConfig object = {
  storage1: {
    name: 'contoso'
    skuName: 'Standard_LRS'
  }
  storage2: {
    name: 'fabrikam'
    skuName: 'Premium_LRS'
  }
}

resource createStorages 'Microsoft.Storage/storageAccounts@2021-06-01' = [for config in
items(storageConfig): {
  name: '${config.value.name}${uniqueString(resourceGroup().id)}'
  location: rgLocation
  sku: {
    name: config.value.skuName
  }
  kind: 'StorageV2'
}]
```

The loop creates two storage accounts. The SKU of the storage account with the name starting with **fabrikam** is **Premium_LRS**.

Loop with condition

For resources and modules, you can add an `if` expression with the loop syntax to conditionally deploy the collection.

```
param rgLocation string = resourceGroup().location
param storageCount int = 2
param createNewStorage bool = true

resource createStorages 'Microsoft.Storage/storageAccounts@2021-06-01' = [for i in range(0, storageCount):
if(createNewStorage) {
  name: '${i}storage${uniqueString(resourceGroup().id)}'
  location: rgLocation
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
}]
```

For more information, see conditional deployment in Bicep.

Clean up resources

When the Azure resources are no longer needed, use the Azure CLI or Azure PowerShell module to delete the quickstart resource group.

- CLI
- PowerShell

```
resourceGroupName = "{provide-the-resource-group-name}"

az group delete --name $resourceGroupName
```

Next steps

Learn modules for Bicep

Quickstart: Publish Bicep modules to private module registry

9/21/2022 • 3 minutes to read • Edit Online

Learn how to publish Bicep modules to private modules registry, and how to call the modules from your Bicep files. Private module registry allows you to share Bicep modules within your organization. To learn more, see Create private registry for Bicep modules. To contribute to the public module registry, see the contribution guide.

Prerequisites

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

To work with module registries, you must have Bicep CLI version `0.4.1008` or later. To use with Azure CLI, you must also have Azure CLI version `2.31.0` or later; to use with Azure PowerShell, you must also have Azure PowerShell version `7.0.0` or later.

A Bicep registry is hosted on Azure Container Registry (ACR). To create one, see Quickstart: Create a container registry by using a Bicep file.

To set up your environment for Bicep development, see Install Bicep tools. After completing those steps, you'll have Visual Studio Code and the Bicep extension.

Create Bicep modules

A module is a Bicep file that is deployed from another Bicep file. Any Bicep file can be used as a module. You can use the following Bicep file in this quickstart. It creates a storage account:

```
@minLength(3)
@maxLength(11)
param storagePrefix string

@allowed([
  'Standard_LRS'
  'Standard_GRS'
  'Standard_RAGRS'
  'Standard_ZRS'
  'Premium_LRS'
  'Premium_ZRS'
  'Standard_GZRS'
  'Standard_RAGZRS'
])
param storageSKU string = 'Standard_LRS'
param location string

var uniqueStorageName = '${storagePrefix}${uniqueString(resourceGroup().id)}'

resource stg 'Microsoft.Storage/storageAccounts@2021-06-01' = {
  name: uniqueStorageName
  location: location
  sku: {
    name: storageSKU
  }
  kind: 'StorageV2'
  properties: {
    supportsHttpsTrafficOnly: true
  }
}

output storageEndpoint object = stg.properties.primaryEndpoints
```

Save the Bicep file as **storage.bicep**.

Publish modules

If you don't have an Azure container registry (ACR), see Prerequisites to create one. The login server name of the ACR is needed. The format of the login server name is: `<registry-name>.azurecr.io`. To get the login server name:

- Azure CLI
- Azure PowerShell

```
az acr show --resource-group <resource-group-name> --name <registry-name> --query loginServer
```

Use the following syntax to publish a Bicep file as a module to a private module registry.

- Azure CLI
- Azure PowerShell

```
az bicep publish --file storage.bicep --target br:exampleregistry.azurecr.io/bicep/modules/storage:v1
```

In the preceding sample, **./storage.bicep** is the Bicep file to be published. Update the file path if needed. The module path has the following syntax:

```
br:<registry-name>.azurecr.io/<file-path>:<tag>
```

- **br** is the schema name for a Bicep registry.
- **file path** is called `repository` in Azure Container Registry. The **file path** can contain segments that are separated by the `/` character. This file path is created if it doesn't exist in the registry.
- **tag** is used for specifying a version for the module.

To verify the published modules, you can list the ACR repository:

- Azure CLI
- Azure PowerShell

```
az acr repository list --name <registry-name> --output table
```

Call modules

To call a module, create a new Bicep file in Visual Studio Code. In the new Bicep file, enter the following line.

```
module stgModule 'br:<registry-name>.azurecr.io/bicep/modules/storage:v1'
```

Replace **<registry-name>** with your ACR registry name. It takes a short moment to restore the module to your local cache. After the module is restored, the red curly line underneath the module path will go away. At the end of the line, add = and a space, and then select **required-properties** as shown in the following screenshot. The module structure is automatically populated.

The following example is a completed Bicep file.

```
@minLength(3)
@maxLength(11)
param namePrefix string
param location string = resourceGroup().location

module stgModule 'br:ace1207.azurecr.io/bicep/modules/storage:v1' = {
  name: 'stgStorage'
  params: {
    location: location
    storagePrefix: namePrefix
  }
}
```

Save the Bicep file locally, and then use Azure CLI or Azure PowerShell to deploy the Bicep file:

- Azure CLI
- Azure PowerShell

```
resourceGroupName = "{provide-a-resource-group-name}"
templateFile="{provide-the-path-to-the-bicep-file}"

az group create --name $resourceGroupName --location eastus

az deployment group create --resource-group $resourceGroupName --template-file $templateFile
```

From the Azure portal, verify the storage account has been created successfully.

Clean up resources

When the Azure resources are no longer needed, use the Azure CLI or Azure PowerShell module to delete the quickstart resource group.

- Azure CLI
- Azure PowerShell

```
resourceGroupName = "{provide-the-resource-group-name}"

az group delete --name $resourceGroupName
```

Next steps

Learn modules for Bicep

Quickstart: Integrate Bicep with Azure Pipelines

9/21/2022 • 2 minutes to read • Edit Online

This quickstart shows you how to integrate Bicep files with Azure Pipelines for continuous integration and continuous deployment (CI/CD).

It provides a short introduction to the pipeline task you need for deploying a Bicep file. If you want more detailed steps on setting up the pipeline and project, see Deploy Azure resources by using Bicep and Azure Pipelines.

Prerequisites

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

You need an Azure DevOps organization. If you don't have one, create one for free. If your team already has an Azure DevOps organization, make sure you're an administrator of the Azure DevOps project that you want to use.

You need to have configured a service connection to your Azure subscription. The tasks in the pipeline execute under the identity of the service principal. For steps to create the connection, see Create a DevOps project.

You need a Bicep file that defines the infrastructure for your project. This file is in a repository.

Create pipeline

1. From your Azure DevOps organization, select **Pipelines** and **New pipeline**.

2. Specify where your code is stored.

3. Select the repository that has the code for your project.

4. Select **Starter pipeline** for the type of pipeline to create.

Deploy Bicep files

You can use Azure Resource Group Deployment task or Azure CLI task to deploy a Bicep file.

Use Azure Resource Group Deployment task

Replace your starter pipeline with the following YAML. It creates a resource group and deploys a Bicep file by using an Azure Resource Group Deployment task:

```
trigger:
- master

name: Deploy Bicep files

variables:
  vmImageName: 'ubuntu-latest'

  azureServiceConnection: '<your-connection-name>'
  resourceGroupName: 'exampleRG'
  location: '<your-resource-group-location>'
  templateFile: './main.bicep'
pool:
  vmImage: $(vmImageName)

steps:
- task: AzureResourceManagerTemplateDeployment@3
  inputs:
    deploymentScope: 'Resource Group'
    azureResourceManagerConnection: '$(azureServiceConnection)'
    action: 'Create Or Update Resource Group'
    resourceGroupName: '$(resourceGroupName)'
    location: '$(location)'
    templateLocation: 'Linked artifact'
    csmFile: '$(templateFile)'
    overrideParameters: '-storageAccountType Standard_LRS'
    deploymentMode: 'Incremental'
    deploymentName: 'DeployPipelineTemplate'
```

For the descriptions of the task inputs, see Azure Resource Group Deployment task.

Select **Save**. The build pipeline automatically runs. Go back to the summary for your build pipeline, and watch the status.

Use Azure CLI task

Replace your starter pipeline with the following YAML. It creates a resource group and deploys a Bicep file by using an Azure CLI task:

```
trigger:
- master

name: Deploy Bicep files

variables:
  vmImageName: 'ubuntu-latest'

  azureServiceConnection: '<your-connection-name>'
  resourceGroupName: 'exampleRG'
  location: '<your-resource-group-location>'
  templateFile: 'main.bicep'
pool:
  vmImage: $(vmImageName)

steps:
- task: AzureCLI@2
  inputs:
    azureSubscription: $(azureServiceConnection)
    scriptType: bash
    scriptLocation: inlineScript
    inlineScript: |
      az --version
      az group create --name $(resourceGroupName) --location $(location)
      az deployment group create --resource-group $(resourceGroupName) --template-file $(templateFile)
```

For the descriptions of the task inputs, see Azure CLI task.

Select **Save**. The build pipeline automatically runs. Go back to the summary for your build pipeline, and watch the status.

Clean up resources

When the Azure resources are no longer needed, use the Azure CLI or Azure PowerShell to delete the quickstart resource group.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

Deploy Bicep files by using GitHub Actions

Quickstart: Deploy Bicep files by using GitHub Actions

9/21/2022 • 6 minutes to read • Edit Online

GitHub Actions is a suite of features in GitHub to automate your software development workflows.

In this quickstart, you use the GitHub Actions for Azure Resource Manager deployment to automate deploying a Bicep file to Azure.

It provides a short introduction to GitHub actions and Bicep files. If you want more detailed steps on setting up the GitHub actions and project, see Deploy Azure resources by using Bicep and GitHub Actions.

Prerequisites

- An Azure account with an active subscription. Create an account for free.
- A GitHub account. If you don't have one, sign up for free.
- A GitHub repository to store your Bicep files and your workflow files. To create one, see Creating a new repository.

Create resource group

Create a resource group. Later in this quickstart, you'll deploy your Bicep file to this resource group.

```
az group create -n exampleRG -l westus
```

Generate deployment credentials

- Service principal
- Open ID Connect

Your GitHub Actions run under an identity. Use the az ad sp create-for-rbac command to create a service principal for the identity.

Replace the placeholder `myApp` with the name of your application. Replace `{subscription-id}` with your subscription ID.

```
az ad sp create-for-rbac --name myApp --role contributor --scopes /subscriptions/{subscription-id}/resourceGroups/exampleRG --sdk-auth
```

> **IMPORTANT**
> The scope in the previous example is limited to the resource group. We recommend that you grant minimum required access.

The output is a JSON object with the role assignment credentials that provide access to your App Service app similar to below. Copy this JSON object for later. You'll only need the sections with the `clientId`, `clientSecret`, `subscriptionId`, and `tenantId` values.

```
{
  "clientId": "<GUID>",
  "clientSecret": "<GUID>",
  "subscriptionId": "<GUID>",
  "tenantId": "<GUID>",
  (...)
}
```

Configure the GitHub secrets

- Service principal
- Open ID Connect

Create secrets for your Azure credentials, resource group, and subscriptions.

1. In GitHub, navigate to your repository.
2. Select **Settings** > **Secrets** > **New secret**.
3. Paste the entire JSON output from the Azure CLI command into the secret's value field. Name the secret `AZURE_CREDENTIALS`.
4. Create another secret named `AZURE_RG`. Add the name of your resource group to the secret's value field (`exampleRG`).
5. Create another secret named `AZURE_SUBSCRIPTION`. Add your subscription ID to the secret's value field (example: `90fd3f9d-4c61-432d-99ba-1273f236afa2`).

Add a Bicep file

Add a Bicep file to your GitHub repository. The following Bicep file creates a storage account:

```
@minLength(3)
@maxLength(11)
param storagePrefix string

@allowed([
  'Standard_LRS'
  'Standard_GRS'
  'Standard_RAGRS'
  'Standard_ZRS'
  'Premium_LRS'
  'Premium_ZRS'
  'Standard_GZRS'
  'Standard_RAGZRS'
])
param storageSKU string = 'Standard_LRS'

param location string = resourceGroup().location

var uniqueStorageName = '${storagePrefix}${uniqueString(resourceGroup().id)}'

resource stg 'Microsoft.Storage/storageAccounts@2021-04-01' = {
  name: uniqueStorageName
  location: location
  sku: {
    name: storageSKU
  }
  kind: 'StorageV2'
  properties: {
    supportsHttpsTrafficOnly: true
  }
}

output storageEndpoint object = stg.properties.primaryEndpoints
```

The Bicep file requires one parameter called **storagePrefix** with 3 to 11 characters.

You can put the file anywhere in the repository. The workflow sample in the next section assumes the Bicep file is named **main.bicep**, and it's stored at the root of your repository.

Create workflow

A workflow defines the steps to execute when triggered. It's a YAML (.yml) file in the **.github/workflows/** path of your repository. The workflow file extension can be either **.yml** or **.yaml**.

To create a workflow, take the following steps:

1. From your GitHub repository, select **Actions** from the top menu.
2. Select **New workflow**.
3. Select **set up a workflow yourself**.
4. Rename the workflow file if you prefer a different name other than **main.yml**. For example: **deployBicepFile.yml**.
5. Replace the content of the yml file with the following code:

   - Service principal
   - OpenID Connect

```
on: [push]
name: Azure ARM
jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:

      # Checkout code
    - uses: actions/checkout@main

      # Log into Azure
    - uses: azure/login@v1
      with:
        creds: ${{ secrets.AZURE_CREDENTIALS }}

      # Deploy Bicep file
    - name: deploy
      uses: azure/arm-deploy@v1
      with:
        subscriptionId: ${{ secrets.AZURE_SUBSCRIPTION }}
        resourceGroupName: ${{ secrets.AZURE_RG }}
        template: ./main.bicep
        parameters: 'storagePrefix=mystore storageSKU=Standard_LRS'
        failOnStdErr: false
```

Replace `mystore` with your own storage account name prefix.

> **NOTE**
>
> You can specify a JSON format parameters file instead in the ARM Deploy action (example: `.azuredeploy.parameters.json`).

The first section of the workflow file includes:

- **name**: The name of the workflow.
- **on**: The name of the GitHub events that triggers the workflow. The workflow is triggered when there's a push event on the main branch.

6. Select **Start commit**.
7. Select **Commit directly to the main branch**.
8. Select **Commit new file** (or **Commit changes**).

Updating either the workflow file or Bicep file triggers the workflow. The workflow starts right after you commit the changes.

Check workflow status

1. Select the **Actions** tab. You'll see a **Create deployStorageAccount.yml** workflow listed. It takes 1-2 minutes to run the workflow.
2. Select the workflow to open it.
3. Select **Run ARM deploy** from the menu to verify the deployment.

Clean up resources

When your resource group and repository are no longer needed, clean up the resources you deployed by deleting the resource group and your GitHub repository.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

Bicep file structure and syntax

Quickstart: Use MSBuild to convert Bicep to JSON

9/21/2022 • 8 minutes to read • Edit Online

This article describes how to use MSBuild to convert a Bicep file to Azure Resource Manager template (ARM template) JSON. The examples use MSBuild from the command line with C# project files that convert Bicep to JSON. The project files are examples that can be used in an MSBuild continuous integration (CI) pipeline.

Prerequisites

You'll need the latest versions of the following software:

- Visual Studio. The free community version will install .NET 6.0, .NET Core 3.1, .NET SDK, MSBuild, .NET Framework 4.8, NuGet package manager, and C# compiler. From the installer, select **Workloads** > **.NET desktop development**.
- Visual Studio Code with the extensions for Bicep and Azure Resource Manager (ARM) Tools.
- PowerShell or a command-line shell for your operating system.

MSBuild tasks and CLI packages

If your existing continuous integration (CI) pipeline relies on MSBuild, you can use MSBuild tasks and CLI packages to convert Bicep files into ARM template JSON.

The functionality relies on the following NuGet packages. The latest NuGet package versions match the latest Bicep version.

PACKAGE NAME	DESCRIPTION
Azure.Bicep.MSBuild	Cross-platform MSBuild task that invokes the Bicep CLI and compiles Bicep files into ARM template JSON.
Azure.Bicep.CommandLine.win-x64	Bicep CLI for Windows.
Azure.Bicep.CommandLine.linux-x64	Bicep CLI for Linux.
Azure.Bicep.CommandLine.osx-x64	Bicep CLI for macOS.

Azure.Bicep.MSBuild package

When referenced in a project file's `PackageReference` the `Azure.Bicep.MSBuild` package imports the `Bicep` task that's used to invoke the Bicep CLI. The package converts its output into MSBuild errors and the `BicepCompile` target that's used to simplify the `Bicep` task's usage. By default the `BicepCompile` runs after the `Build` target and compiles all `@(Bicep)` items and places the output in `$(OutputPath)` with the same file name and the *.json* extension.

The following example compiles *one.bicep* and *two.bicep* files in the same directory as the project file and places the compiled *one.json* and *two.json* in the `$(OutputPath)` directory.

```
<ItemGroup>
  <Bicep Include="one.bicep" />
  <Bicep Include="two.bicep" />
</ItemGroup>
```

You can override the output path per file using the `OutputFile` metadata on `Bicep` items. The following example will recursively find all *main.bicep* files and place the compiled *.json* files in `$(OutputPath)` under a subdirectory with the same name in `$(OutputPath)`:

```
<ItemGroup>
  <Bicep Include="**\main.bicep" OutputFile="$(OutputPath)\%(RecursiveDir)\%(FileName).json" />
</ItemGroup>
```

More customizations can be performed by setting one of the following properties in your project:

PROPERTY NAME	DEFAULT VALUE	DESCRIPTION
`BicepCompileAfterTargets`	`Build`	Used as `AfterTargets` value for the `BicepCompile` target. Change the value to override the scheduling of the `BicepCompile` target in your project.
`BicepCompileDependsOn`	None	Used as `DependsOnTargets` value for the `BicepCompile` target. This property can be set to targets that you want `BicepCompile` target to depend on.
`BicepCompileBeforeTargets`	None	Used as `BeforeTargets` value for the `BicepCompile` target.
`BicepOutputPath`	`$(OutputPath)`	Set this property to override the default output path for the compiled ARM template. `OutputFile` metadata on `Bicep` items takes precedence over this value.

The `Azure.Bicep.MSBuild` requires the `BicepPath` property to be set either in order to function. You may set it by referencing the appropriate `Azure.Bicep.CommandLine.*` package for your operating system or manually by installing the Bicep CLI and setting the `BicepPath` environment variable or MSBuild property.

Azure.Bicep.CommandLine packages

The `Azure.Bicep.CommandLine.*` packages are available for Windows, Linux, and macOS. When referenced in a project file via a `PackageReference`, the `Azure.Bicep.CommandLine.*` packages set the `BicepPath` property to the full path of the Bicep executable for the platform. The reference to this package may be omitted if Bicep CLI is installed through other means and the `BicepPath` environment variable or MSBuild property are set accordingly.

SDK-based examples

The following examples contain a default Console App SDK-based C# project file that was modified to convert Bicep files into ARM templates. Replace `__LATEST_VERSION__` with the latest version of the Bicep NuGet packages.

The .NET Core 3.1 and .NET 6 examples are similar. But .NET 6 uses a different format for the *Program.cs* file. For more information, see .NET 6 C# console app template generates top-level statements.

.NET 6

In this example, the `RootNamespace` property contains a placeholder value. When you create a project file, the value matches your project's name.

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net6.0</TargetFramework>
    <RootNamespace>net6-sdk-project-name</RootNamespace>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Azure.Bicep.CommandLine.win-x64" Version="__LATEST_VERSION__" />
    <PackageReference Include="Azure.Bicep.MSBuild" Version="__LATEST_VERSION__" />
  </ItemGroup>

  <ItemGroup>
    <Bicep Include="**\main.bicep" OutputFile="$(OutputPath)\%(RecursiveDir)\%(FileName).json" />
  </ItemGroup>
</Project>
```

.NET Core 3.1

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp3.1</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Azure.Bicep.CommandLine.win-x64" Version="__LATEST_VERSION__" />
    <PackageReference Include="Azure.Bicep.MSBuild" Version="__LATEST_VERSION__" />
  </ItemGroup>

  <ItemGroup>
    <Bicep Include="**\main.bicep" OutputFile="$(OutputPath)\%(RecursiveDir)\%(FileName).json" />
  </ItemGroup>
</Project>
```

NoTargets SDK

The following example contains a project that converts Bicep files into ARM templates using Microsoft.Build.NoTargets. This SDK allows creation of standalone projects that compile only Bicep files. Replace `__LATEST_VERSION__` with the latest version of the Bicep NuGet packages.

For Microsoft.Build.NoTargets, specify a version like `Microsoft.Build.NoTargets/3.5.6`.

```
<Project Sdk="Microsoft.Build.NoTargets/__LATEST_VERSION__">
  <PropertyGroup>
    <TargetFramework>net48</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Azure.Bicep.CommandLine.win-x64" Version="__LATEST_VERSION__" />
    <PackageReference Include="Azure.Bicep.MSBuild" Version="__LATEST_VERSION__" />
  </ItemGroup>

  <ItemGroup>
    <Bicep Include="main.bicep"/>
  </ItemGroup>
</Project>
```

Classic framework

The following example converts Bicep to JSON inside a classic project file that's not SDK-based. Only use the

classic example if the previous examples don't work for you. Replace `__LATEST_VERSION__` with the latest version of the Bicep NuGet packages.

In this example, the `ProjectGuid`, `RootNamespace` and `AssemblyName` properties contain placeholder values. When you create a project file, a unique GUID is created and the name values match your project's name.

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="15.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <Import Project="$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Microsoft.Common.props"
Condition="Exists('$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Microsoft.Common.props')" />
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
    <ProjectGuid>{11111111-1111-1111-1111-111111111111}</ProjectGuid>
    <OutputType>Exe</OutputType>
    <RootNamespace>ClassicFramework</RootNamespace>
    <AssemblyName>ClassicFramework</AssemblyName>
    <TargetFrameworkVersion>v4.8</TargetFrameworkVersion>
    <FileAlignment>512</FileAlignment>
    <AutoGenerateBindingRedirects>true</AutoGenerateBindingRedirects>
    <Deterministic>true</Deterministic>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
    <PlatformTarget>AnyCPU</PlatformTarget>
    <DebugSymbols>true</DebugSymbols>
    <DebugType>full</DebugType>
    <Optimize>false</Optimize>
    <OutputPath>bin\Debug\</OutputPath>
    <DefineConstants>DEBUG;TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
    <PlatformTarget>AnyCPU</PlatformTarget>
    <DebugType>pdbonly</DebugType>
    <Optimize>true</Optimize>
    <OutputPath>bin\Release\</OutputPath>
    <DefineConstants>TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <ItemGroup>
    <Reference Include="System" />
    <Reference Include="System.Core" />
    <Reference Include="System.Xml.Linq" />
    <Reference Include="System.Data.DataSetExtensions" />
    <Reference Include="Microsoft.CSharp" />
    <Reference Include="System.Data" />
    <Reference Include="System.Net.Http" />
    <Reference Include="System.Xml" />
  </ItemGroup>
  <ItemGroup>
    <Compile Include="Program.cs" />
    <Compile Include="Properties\AssemblyInfo.cs" />
  </ItemGroup>
  <ItemGroup>
    <None Include="App.config" />
    <Bicep Include="main.bicep" />
  </ItemGroup>
  <ItemGroup>
    <PackageReference Include="Azure.Bicep.CommandLine.win-x64">
      <Version>__LATEST_VERSION__</Version>
    </PackageReference>
    <PackageReference Include="Azure.Bicep.MSBuild">
      <Version>__LATEST_VERSION__</Version>
    </PackageReference>
  </ItemGroup>
  <Import Project="$(MSBuildToolsPath)\Microsoft.CSharp.targets" />
</Project>
```

Convert Bicep to JSON

The following examples show how MSBuild converts a Bicep file to JSON. Follow the instructions to create one of the project files for .NET, .NET Core 3.1, or Classic framework. Then continue to create the Bicep file and run MSBuild.

- .NET
- .NET Core 3.1
- Classic framework

Build a project in .NET with the dotnet CLI.

1. Open Visual Studio code and select **Terminal** > **New Terminal** to start a PowerShell session.
2. Create a directory named *bicep-msbuild-demo* and go to the directory. This example uses *C:\bicep-msbuild-demo*.

   ```
   New-Item -Name .\bicep-msbuild-demo -ItemType Directory
   Set-Location -Path .\bicep-msbuild-demo
   ```

3. Run the `dotnet` command to create a new console with the .NET 6 framework.

   ```
   dotnet new console --framework net6.0
   ```

   The project file uses the same name as your directory, *bicep-msbuild-demo.csproj*. For more information about how to create a console application from Visual Studio Code, see the tutorial.

4. Replace the contents of *bicep-msbuild-demo.csproj* with the .NET 6 or NoTargets SDK examples.
5. Replace `__LATEST_VERSION__` with the latest version of the Bicep NuGet packages.
6. Save the file.

Create Bicep file

You'll need a Bicep file that will be converted to JSON.

1. Use Visual Studio Code and create a new file.
2. Copy the following sample and save it as *main.bicep* in the *C:\bicep-msbuild-demo* directory.

```
@allowed([
  'Premium_LRS'
  'Premium_ZRS'
  'Standard_GRS'
  'Standard_GZRS'
  'Standard_LRS'
  'Standard_RAGRS'
  'Standard_RAGZRS'
  'Standard_ZRS'
])
@description('Storage account type.')
param storageAccountType string = 'Standard_LRS'

@description('Location for all resources.')
param location string = resourceGroup().location

var storageAccountName = 'storage${uniqueString(resourceGroup().id)}'

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-09-01' = {
  name: storageAccountName
  location: location
  sku: {
    name: storageAccountType
  }
  kind: 'StorageV2'
}

output storageAccountNameOutput string = storageAccount.name
```

Run MSBuild

Run MSBuild to convert the Bicep file to JSON.

1. Open a Visual Studio Code terminal session.

2. In the PowerShell session, go to the *C:\bicep-msbuild-demo* directory.

3. Run MSBuild.

   ```
   MSBuild.exe -restore .\bicep-msbuild-demo.csproj
   ```

   The `restore` parameter creates dependencies needed to compile the Bicep file during the initial build. The parameter is optional after the initial build.

4. Go to the output directory and open the *main.json* file that should look like the sample.

   MSBuild creates an output directory based on the SDK or framework version:

   - .NET 6: *\bin\Debug\net6.0*
   - .NET Core 3.1: *\bin\Debug\netcoreapp3.1*
   - NoTargets SDK: *\bin\Debug\net48*
   - Classic framework: *\bin\Debug*

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "metadata": {
    "_generator": {
      "name": "bicep",
      "version": "0.8.9.13224",
      "templateHash": "12345678901234567890"
    }
  },
  "parameters": {
    "storageAccountType": {
      "type": "string",
      "defaultValue": "Standard_LRS",
      "metadata": {
        "description": "Storage account type."
      },
      "allowedValues": [
        "Premium_LRS",
        "Premium_ZRS",
        "Standard_GRS",
        "Standard_GZRS",
        "Standard_LRS",
        "Standard_RAGRS",
        "Standard_RAGZRS",
        "Standard_ZRS"
      ]
    },
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]",
      "metadata": {
        "description": "Location for all resources."
      }
    }
  },
  "variables": {
    "storageAccountName": "[format('storage{0}', uniqueString(resourceGroup().id))]"
  },
  "resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "apiVersion": "2021-09-01",
      "name": "[variables('storageAccountName')]",
      "location": "[parameters('location')]",
      "sku": {
        "name": "[parameters('storageAccountType')]"
      },
      "kind": "StorageV2"
    }
  ],
  "outputs": {
    "storageAccountNameOutput": {
      "type": "string",
      "value": "[variables('storageAccountName')]"
    }
  }
}
```

If you make changes or want to rerun the build, delete the output directory so new files can be created.

Clean up resources

When you're finished with the files, delete the directory. For this example, delete *C:\bicep-msbuild-demo*.

```
Remove-Item -Path "C:\bicep-msbuild-demo" -Recurse
```

Next steps

- For more information about MSBuild, see MSBuild reference and .NET project files.
- To learn more about MSBuild properties, items, targets, and tasks, see MSBuild concepts.
- For more information about the .NET CLI, see .NET CLI overview.

Learn modules for Bicep

9/21/2022 • 2 minutes to read • Edit Online

Ready to see how Bicep can help simplify and accelerate your deployments to Azure? Check out the many hands-on courses.

> **TIP**
> Want to learn Bicep live from subject matter experts? Follow on-demand Learn Live sessions with our experts.

Get started

If you're new to Bicep, a great way to get started is by reviewing the following Learn module. You'll learn how Bicep makes it easier to define how your Azure resources should be configured and deployed in a way that's automated and repeatable. You'll deploy several Azure resources so you can see for yourself how Bicep works. We provide free access to Azure resources to help you practice the concepts.

Build your first Bicep template

Learn more

To learn even more about Bicep's features, take these learning paths:

Part 1: Fundamentals of Bicep

Part 2: Intermediate Bicep

Part 3: Advanced Bicep

Use Bicep in a deployment pipeline

After that, you might be interested in adding your Bicep code to a deployment pipeline. Take one of these two learning paths based on the tool you want to use:

Option 1: Deploy Azure resources by using Bicep and Azure Pipelines

Option 2: Deploy Azure resources by using Bicep and GitHub Actions

Next steps

- For a short introduction to Bicep, see Bicep quickstart.
- For suggestions about how to improve your Bicep files, see Best practices for Bicep.

Quickstart: Deploy Cognitive Search using Bicep

9/21/2022 • 3 minutes to read • Edit Online

This article walks you through the process for using a Bicep file to deploy an Azure Cognitive Search resource in the Azure portal.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Only those properties included in the template are used in the deployment. If more customization is required, such as setting up network security, you can update the service as a post-deployment task. To customize an existing service with the fewest steps, use Azure CLI or Azure PowerShell. If you're evaluating preview features, use the Management REST API.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Service name must only contain lowercase letters, digits or dashes, cannot use dash as the first two or last one characters, cannot contain consecutive dashes, and is limited between 2 and 60 characters in length.')
@minLength(2)
@maxLength(60)
param name string

@allowed([
  'free'
  'basic'
  'standard'
  'standard2'
  'standard3'
  'storage_optimized_l1'
  'storage_optimized_l2'
])
@description('The pricing tier of the search service you want to create (for example, basic or standard).')
param sku string = 'standard'

@description('Replicas distribute search workloads across the service. You need at least two replicas to support high availability of query workloads (not applicable to the free tier).')
@minValue(1)
@maxValue(12)
param replicaCount int = 1

@description('Partitions allow for scaling of document count as well as faster indexing by sharding your index over multiple search units.')
@allowed([
  1
  2
  3
  4
  6
  12
])
param partitionCount int = 1

@description('Applicable only for SKUs set to standard3. You can set this property to enable a single, high density partition that allows up to 1000 indexes, which is much higher than the maximum indexes allowed for any other SKU.')
@allowed([
  'default'
  'highDensity'
])
param hostingMode string = 'default'

@description('Location for all resources.')
param location string = resourceGroup().location

resource search 'Microsoft.Search/searchServices@2020-08-01' = {
  name: name
  location: location
  sku: {
    name: sku
  }
  properties: {
    replicaCount: replicaCount
    partitionCount: partitionCount
    hostingMode: hostingMode
  }
}
```

The Azure resource defined in this Bicep file:

- Microsoft.Search/searchServices: create an Azure Cognitive Search service

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
serviceName=<service-name>
```

> **NOTE**
>
> Replace **<service-name>** with the name of the Search service. The service name must only contain lowercase letters, digits, or dashes. You can't use a dash as the first two characters or the last character. The name has a minimum length of 2 characters and a maximum length of 60 characters.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

Other Cognitive Search quickstarts and tutorials build upon this quickstart. If you plan to continue on to work with subsequent quickstarts and tutorials, you may wish to leave this resource in place. When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you created a Cognitive Search service using a Bicep file, and then validated the deployment. To learn more about Cognitive Search and Azure Resource Manager, continue on to the articles below.

- Read an overview of Azure Cognitive Search.
- Create an index for your search service.
- Create a demo app using the portal wizard.
- Create a skillset to extract information from your data.

Quickstart: Create a Cognitive Services resource using Bicep

9/21/2022 • 3 minutes to read • Edit Online

Follow this quickstart to create Cognitive Services resource using Bicep.

Azure Cognitive Services are cloud-based artificial intelligence (AI) services that help developers build cognitive intelligence into applications without having direct AI or data science skills or knowledge. They are available through REST APIs and client library SDKs in popular development languages. Azure Cognitive Services enables developers to easily add cognitive features into their applications with cognitive solutions that can see, hear, speak, and analyze.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Things to consider

Using Bicep to create a Cognitive Service resource lets you create a multi-service resource. This enables you to:

- Access multiple Azure Cognitive Services with a single key and endpoint.
- Consolidate billing from the services you use.
- You must create your first Face, Language service, or Computer Vision resources from the Azure portal to review and acknowledge the terms and conditions. You can do so here: Face, Language service, Computer Vision. After that, you can create subsequent resources using any deployment tool (SDK, CLI, or ARM template, etc) under the same Azure subscription.

Prerequisites

- If you don't have an Azure subscription, create one for free.

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('That name is the name of our application. It has to be unique.Type a name followed by your
resource group name. (<name>-<resourceGroupName>)')
param cognitiveServiceName string = 'CognitiveService-${uniqueString(resourceGroup().id)}'

@description('Location for all resources.')
param location string = resourceGroup().location

@allowed([
  'S0'
])
param sku string = 'S0'

resource cognitiveService 'Microsoft.CognitiveServices/accounts@2021-10-01' = {
  name: cognitiveServiceName
  location: location
  sku: {
    name: sku
  }
  kind: 'CognitiveServices'
  properties: {
    apiProperties: {
      statisticsEnabled: false
    }
  }
}
```

One Azure resource is defined in the Bicep file: Microsoft.CognitiveServices/accounts specifies that it is a Cognitive Services resource. The `kind` field in the Bicep file defines the type of resource.

See the list of SKUs and pricing information below.

Multi-service

SERVICE	KIND
Multiple services. For more information, see the pricing page.	`CognitiveServices`

Vision

SERVICE	KIND
Computer Vision	`ComputerVision`
Custom Vision - Prediction	`CustomVision.Prediction`
Custom Vision - Training	`CustomVision.Training`
Face	`Face`
Form Recognizer	`FormRecognizer`

Speech

SERVICE	KIND
Speech Services	`SpeechServices`

Language

SERVICE	KIND
LUIS	`LUIS`
QnA Maker	`QnAMaker`
Language service	`TextAnalytics`
Text Translation	`TextTranslation`

Decision

SERVICE	KIND
Anomaly Detector	`AnomalyDetector`
Content Moderator	`ContentModerator`
Personalizer	`Personalizer`

Pricing tiers and billing

Pricing tiers (and the amount you get billed) are based on the number of transactions you send using your authentication information. Each pricing tier specifies the:

- maximum number of allowed transactions per second (TPS).
- service features enabled within the pricing tier.
- cost for a predefined number of transactions. Going above this number will cause an extra charge as specified in the pricing details for your service.

> **NOTE**
> Many of the Cognitive Services have a free tier you can use to try the service. To use the free tier, use `F0` as the SKU for your resource.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

   ```
   az group create --name exampleRG --location eastus
   az deployment group create --resource-group exampleRG --template-file main.bicep
   ```

   When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI

- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

If you need to recover a deleted resource, see Recover deleted Cognitive Services resources.

See also

- See **Authenticate requests to Azure Cognitive Services** on how to securely work with Cognitive Services.
- See **What are Azure Cognitive Services?** to get a list of different categories within Cognitive Services.
- See **Natural language support** to see the list of natural languages that Cognitive Services supports.
- See **Use Cognitive Services as containers** to understand how to use Cognitive Services on-prem.
- See **Plan and manage costs for Cognitive Services** to estimate cost of using Cognitive Services.

Quickstart: Create an Ubuntu Data Science Virtual Machine using Bicep

9/21/2022 • 3 minutes to read • Edit Online

This quickstart will show you how to create an Ubuntu 18.04 Data Science Virtual Machine using Bicep. Data Science Virtual Machines are cloud-based virtual machines preloaded with a suite of data science and machine learning frameworks and tools. When deployed on GPU-powered compute resources, all tools and libraries are configured to use the GPU.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Username for Administrator Account')
param adminUsername string

@description('The name of you Virtual Machine.')
param vmName string = 'vmName'

@description('Location for all resources.')
param location string = resourceGroup().location

@description('Choose between CPU or GPU processing')
@allowed([
  'CPU-4GB'
  'CPU-7GB'
  'CPU-8GB'
  'CPU-14GB'
  'CPU-16GB'
  'GPU-56GB'
])
param cpu_gpu string = 'CPU-4GB'

@description('Name of the VNET')
param virtualNetworkName string = 'vNet'

@description('Name of the subnet in the virtual network')
param subnetName string = 'subnet'

@description('Name of the Network Security Group')
param networkSecurityGroupName string = 'SecGroupNet'

@description('Type of authentication to use on the Virtual Machine. SSH key is recommended.')
@allowed([
  'sshPublicKey'
  'password'
])
param authenticationType string = 'sshPublicKey'
```

```
@description('SSH Key or password for the Virtual Machine. SSH key is recommended.')
@secure()
param adminPasswordOrKey string

var networkInterfaceName = '${vmName}NetInt'
var virtualMachineName = vmName
var publicIpAddressName = '${vmName}PublicIP'
var subnetRef = resourceId('Microsoft.Network/virtualNetworks/subnets', virtualNetworkName, subnetName)
var nsgId = networkSecurityGroup.id
var osDiskType = 'StandardSSD_LRS'
var storageAccountName = 'storage${uniqueString(resourceGroup().id)}'
var storageAccountType = 'Standard_LRS'
var storageAccountKind = 'Storage'
var vmSize = {
  'CPU-4GB': 'Standard_B2s'
  'CPU-7GB': 'Standard_D2s_v3'
  'CPU-8GB': 'Standard_D2s_v3'
  'CPU-14GB': 'Standard_D4s_v3'
  'CPU-16GB': 'Standard_D4s_v3'
  'GPU-56GB': 'Standard_NC6_Promo'
}
var linuxConfiguration = {
  disablePasswordAuthentication: true
  ssh: {
    publicKeys: [
      {
        path: '/home/${adminUsername}/.ssh/authorized_keys'
        keyData: adminPasswordOrKey
      }
    ]
  }
}

resource networkInterface 'Microsoft.Network/networkInterfaces@2021-05-01' = {
  name: networkInterfaceName
  location: location
  properties: {
    ipConfigurations: [
      {
        name: 'ipconfig1'
        properties: {
          subnet: {
            id: subnetRef
          }
          privateIPAllocationMethod: 'Dynamic'
          publicIPAddress: {
            id: publicIpAddress.id
          }
        }
      }
    ]
    networkSecurityGroup: {
      id: nsgId
    }
  }
  dependsOn: [
    virtualNetwork
  ]
}

resource networkSecurityGroup 'Microsoft.Network/networkSecurityGroups@2021-05-01' = {
  name: networkSecurityGroupName
  location: location
  properties: {
    securityRules: [
      {
        name: 'JupyterHub'
        properties: {
          priority: 1010
```

```
          protocol: 'Tcp'
          access: 'Allow'
          direction: 'Inbound'
          sourceAddressPrefix: '*'
          sourcePortRange: '*'
          destinationAddressPrefix: '*'
          destinationPortRange: '8000'
        }
      }
      {
        name: 'RStudioServer'
        properties: {
          priority: 1020
          protocol: 'Tcp'
          access: 'Allow'
          direction: 'Inbound'
          sourceAddressPrefix: '*'
          sourcePortRange: '*'
          destinationAddressPrefix: '*'
          destinationPortRange: '8787'
        }
      }
      {
        name: 'SSH'
        properties: {
          priority: 1030
          protocol: 'Tcp'
          access: 'Allow'
          direction: 'Inbound'
          sourceAddressPrefix: '*'
          sourcePortRange: '*'
          destinationAddressPrefix: '*'
          destinationPortRange: '22'
        }
      }
    ]
  }
}

resource virtualNetwork 'Microsoft.Network/virtualNetworks@2021-05-01' = {
  name: virtualNetworkName
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        '10.0.0.0/24'
      ]
    }
    subnets: [
      {
        name: subnetName
        properties: {
          addressPrefix: '10.0.0.0/24'
          privateEndpointNetworkPolicies: 'Enabled'
          privateLinkServiceNetworkPolicies: 'Enabled'
        }
      }
    ]
  }
}

resource publicIpAddress 'Microsoft.Network/publicIPAddresses@2021-05-01' = {
  name: publicIpAddressName
  location: location
  sku: {
    name: 'Basic'
    tier: 'Regional'
  }
  properties: {
```

```
    publicIPAllocationMethod: 'Dynamic'
  }
}

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-08-01' = {
  name: storageAccountName
  location: location
  sku: {
    name: storageAccountType
  }
  kind: storageAccountKind
}

resource virtualMachine 'Microsoft.Compute/virtualMachines@2021-11-01' = {
  name: '${virtualMachineName}-${cpu_gpu}'
  location: location
  properties: {
    hardwareProfile: {
      vmSize: vmSize[cpu_gpu]
    }
    storageProfile: {
      osDisk: {
        createOption: 'FromImage'
        managedDisk: {
          storageAccountType: osDiskType
        }
      }
      imageReference: {
        publisher: 'microsoft-dsvm'
        offer: 'ubuntu-1804'
        sku: '1804-gen2'
        version: 'latest'
      }
    }
    networkProfile: {
      networkInterfaces: [
        {
          id: networkInterface.id
        }
      ]
    }
    osProfile: {
      computerName: virtualMachineName
      adminUsername: adminUsername
      adminPassword: adminPasswordOrKey
      linuxConfiguration: ((authenticationType == 'password') ? json('null') : linuxConfiguration)
    }
  }
}

output adminUsername string = adminUsername
```

The following resources are defined in the Bicep file:

- Microsoft.Network/networkInterfaces
- Microsoft.Network/networkSecurityGroups
- Microsoft.Network/virtualNetworks
- Microsoft.Network/publicIPAddresses
- Microsoft.Storage/storageAccounts
- Microsoft.Compute/virtualMachines: Create a cloud-based virtual machine. In this template, the virtual machine is configured as a Data Science Virtual Machine running Ubuntu 18.04.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - Azure CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
adminUsername=<admin-user> vmName=<vm-name>
```

> **NOTE**
> Replace **<admin-user>** with the username for the administrator account. Replace **<vm-name>** with the name of your virtual machine.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- Azure CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- Azure CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you created a Data Science Virtual Machine using Bicep.

Sample programs & ML walkthroughs

How to create a secure workspace by using template

9/21/2022 • 6 minutes to read • Edit Online

Templates provide a convenient way to create reproducible service deployments. The template defines what will be created, with some information provided by you when you use the template. For example, specifying a unique name for the Azure Machine Learning workspace.

In this tutorial, you learn how to use a Microsoft Bicep and Hashicorp Terraform template to create the following Azure resources:

- Azure Virtual Network. The following resources are secured behind this VNet:
  - Azure Machine Learning workspace
    - Azure Machine Learning compute instance
    - Azure Machine Learning compute cluster
  - Azure Storage Account
  - Azure Key Vault
  - Azure Application Insights
  - Azure Container Registry
  - Azure Bastion host
  - Azure Machine Learning Virtual Machine (Data Science Virtual Machine)
  - The **Bicep** template also creates an Azure Kubernetes Service cluster, and a separate resource group for it.

Prerequisites

Before using the steps in this article, you must have an Azure subscription. If you don't have an Azure subscription, create a free account.

You must also have either a Bash or Azure PowerShell command line.

> **TIP**
> When reading this article, use the tabs in each section to select whether to view information on using Bicep or Terraform templates.

- Bicep
- Terraform

1. To install the command-line tools, see Set up Bicep development and deployment environments.
2. The Bicep template used in this article is located at https://github.com/Azure/azure-quickstart-templates/blob/master/quickstarts/microsoft.machinelearningservices/machine-learning-end-to-end-secure. Use the following commands to clone the GitHub repo to your development environment:

> **TIP**
> If you do not have the `git` command on your development environment, you can install it from https://git-scm.com/.

```
git clone https://github.com/Azure/azure-quickstart-templates
cd azure-quickstart-templates/quickstarts/microsoft.machinelearningservices/machine-learning-end-to-end-secure
```

Understanding the template

- Bicep
- Terraform

The Bicep template is made up of the main.bicep and the `.bicep` files in the modules subdirectory. The following table describes what each file is responsible for:

FILE	DESCRIPTION
main.bicep	Parameters and variables. Passing parameters & variables to other modules in the `modules` subdirectory.
vnet.bicep	Defines the Azure Virtual Network and subnets.
nsg.bicep	Defines the network security group rules for the VNet.
bastion.bicep	Defines the Azure Bastion host and subnet. Azure Bastion allows you to easily access a VM inside the VNet using your web browser.
dsvmjumpbox.bicep	Defines the Data Science Virtual Machine (DSVM). Azure Bastion is used to access this VM through your web browser.
storage.bicep	Defines the Azure Storage account used by the workspace for default storage.
keyvault.bicep	Defines the Azure Key Vault used by the workspace.
containerregistry.bicep	Defines the Azure Container Registry used by the workspace.
applicationinsights.bicep	Defines the Azure Application Insights instance used by the workspace.
machinelearningnetworking.bicep	Defines te private endpoints and DNS zones for the Azure Machine Learning workspace.
Machinelearning.bicep	Defines the Azure Machine Learning workspace.
machinelearningcompute.bicep	Defines an Azure Machine Learning compute cluster and compute instance.
privateaks.bicep	Defines an Azure Kubernetes Services cluster instance.

> **IMPORTANT**
> The DSVM and Azure Bastion is used as an easy way to connect to the secured workspace for this tutorial. In a production environment, we recommend using an Azure VPN gateway or Azure ExpressRoute to access the resources inside the VNet directly from your on-premises network.

Configure the template

- Bicep
- Terraform

To run the Bicep template, use the following commands from the `machine-learning-end-to-end-secure` where the `main.bicep` file is:

1. To create a new Azure Resource Group, use the following command. Replace `exampleRG` with your resource group name, and `eastus` with the Azure region you want to use:

   - Azure CLI
   - Azure PowerShell

   ```
   az group create --name exampleRG --location eastus
   ```

2. To run the template, use the following command:

   - Azure CLI
   - Azure PowerShell

   ```
   az deployment group create \
       --resource-group exampleRG \
       --template-file main.bicep \
       --parameters \
       prefix=myprefix \
       dsvmJumpboxUsername=azureadmin \
       dsvmJumpboxPassword=securepassword
   ```

Connect to the workspace

After the template completes, use the following steps to connect to the DSVM:

1. From the Azure portal, select the Azure Resource Group you used with the template. Then, select the Data Science Virtual Machine that was created by the template. If you have trouble finding it, use the filters section to filter the **Type** to **virtual machine**.

2. From the **Overview** section of the Virtual Machine, select **Connect**, and then select **Bastion** from the dropdown.

3. When prompted, provide the **username** and **password** you specified when configuring the template and then select **Connect**.

> **IMPORTANT**
>
> The first time you connect to the DSVM desktop, a PowerShell window opens and begins running a script. Allow this to complete before continuing with the next step.

4. From the DSVM desktop, start **Microsoft Edge** and enter `https://ml.azure.com` as the address. Sign in to your Azure subscription, and then select the workspace created by the template. The studio for your workspace is displayed.

Next steps

> **IMPORTANT**
>
> The Data Science Virtual Machine (DSVM) and any compute instance resources bill you for every hour that they are running. To avoid excess charges, you should stop these resources when they are not in use. For more information, see the following articles:
>
> - Create/manage VMs (Linux).
> - Create/manage VMs (Windows).
> - Create/manage compute instance.

To continue learning how to use the secured workspace from the DSVM, see Tutorial: Get started with a Python

script in Azure Machine Learning.

To learn more about common secure workspace configurations and input/output requirements, see Azure Machine Learning secure workspace traffic flow.

Quickstart: Create a server - Bicep

9/21/2022 • 2 minutes to read • Edit Online

This quickstart describes how to create an Analysis Services server resource in your Azure subscription by using Bicep.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

- **Azure subscription**: Visit Azure Free Trial to create an account.
- **Azure Active Directory**: Your subscription must be associated with an Azure Active Directory tenant. And, you need to be signed in to Azure with an account in that Azure Active Directory. To learn more, see Authentication and user permissions.

Review the Bicep file

The Bicep file used in this quickstart is from Azure quickstart templates.

```
@description('The name of the Azure Analysis Services server to create. Server name must begin with a letter, be lowercase alphanumeric, and between 3 and 63 characters in length. Server name must be unique per region.')
param serverName string

@description('Location of the Azure Analysis Services server. For supported regions, see https://docs.microsoft.com/en-us/azure/analysis-services/analysis-services-overview#availability-by-region')
param location string = resourceGroup().location

@description('The sku name of the Azure Analysis Services server to create. Choose from: B1, B2, D1, S0, S1, S2, S3, S4, S8, S9. Some skus are region specific. See https://docs.microsoft.com/en-us/azure/analysis-services/analysis-services-overview#availability-by-region')
param skuName string = 'S0'

@description('The total number of query replica scale-out instances. Scale-out of more than one instance is supported on selected regions only. See https://docs.microsoft.com/en-us/azure/analysis-services/analysis-services-overview#availability-by-region')
param capacity int = 1

@description('The inbound firewall rules to define on the server. If not specified, firewall is disabled.')
param firewallSettings object = {
  firewallRules: [
    {
      firewallRuleName: 'AllowFromAll'
      rangeStart: '0.0.0.0'
      rangeEnd: '255.255.255.255'
    }
  ]
  enablePowerBIService: true
}

@description('The SAS URI to a private Azure Blob Storage container with read, write and list permissions. Required only if you intend to use the backup/restore functionality. See https://docs.microsoft.com/en-us/azure/analysis-services/analysis-services-backup')
param backupBlobContainerUri string = ''

resource server 'Microsoft.AnalysisServices/servers@2017-08-01' = {
  name: serverName
  location: location
  sku: {
    name: skuName
    capacity: capacity
  }
  properties: {
    ipV4FirewallSettings: firewallSettings
    backupBlobContainerUri: backupBlobContainerUri
  }
}
```

A single Microsoft.AnalysisServices/servers resource with a firewall rule is defined in the Bicep file.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus

az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
serverName=<analysis-service-name>
```

> **NOTE**
>
> Replace **<analysis-service-name>** with a unique analysis service name.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Validate the deployment

Use the Azure portal or Azure PowerShell to verify the resource group and server resource was created.

```
Get-AzAnalysisServicesServer -Name <analysis-service-name>
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and the server resource.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you used a Bicep file to create a new resource group and an Azure Analysis Services server resource. After you've created a server resource by using the template, consider the following:

Quickstart: Configure server firewall - Portal

Quickstart: Create an Azure Data Factory using Bicep

9/21/2022 • 5 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This quickstart describes how to use Bicep to create an Azure data factory. The pipeline you create in this data factory **copies** data from one folder to another folder in an Azure blob storage. For a tutorial on how to **transform** data using Azure Data Factory, see Tutorial: Transform data using Spark.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

> **NOTE**
> This article does not provide a detailed introduction of the Data Factory service. For an introduction to the Azure Data Factory service, see Introduction to Azure Data Factory.

Prerequisites

Azure subscription

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Data Factory Name')
param dataFactoryName string = 'datafactory${uniqueString(resourceGroup().id)}'

@description('Location of the data factory.')
param location string = resourceGroup().location

@description('Name of the Azure storage account that contains the input/output data.')
param storageAccountName string = 'storage${uniqueString(resourceGroup().id)}'

@description('Name of the blob container in the Azure Storage account.')
param blobContainerName string = 'blob${uniqueString(resourceGroup().id)}'

var dataFactoryLinkedServiceName = 'ArmtemplateStorageLinkedService'
var dataFactoryDataSetInName = 'ArmtemplateTestDatasetIn'
var dataFactoryDataSetOutName = 'ArmtemplateTestDatasetOut'
var pipelineName = 'ArmtemplateSampleCopyPipeline'

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-08-01' = {
  name: storageAccountName
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
}

resource blobContainer 'Microsoft.Storage/storageAccounts/blobServices/containers@2021-08-01' = {
```

```
resource blobContainer 'Microsoft.Storage/storageAccounts/blobServices/containers@2021-08-01' = {
  name: '${storageAccount.name}/default/${blobContainerName}'
}

resource dataFactory 'Microsoft.DataFactory/factories@2018-06-01' = {
  name: dataFactoryName
  location: location
  identity: {
    type: 'SystemAssigned'
  }
}

resource dataFactoryLinkedService 'Microsoft.DataFactory/factories/linkedservices@2018-06-01' = {
  parent: dataFactory
  name: dataFactoryLinkedServiceName
  properties: {
    type: 'AzureBlobStorage'
    typeProperties: {
      connectionString:
'DefaultEndpointsProtocol=https;AccountName=${storageAccount.name};AccountKey=${storageAccount.listKeys().ke
ys[0].value}'
    }
  }
}

resource dataFactoryDataSetIn 'Microsoft.DataFactory/factories/datasets@2018-06-01' = {
  parent: dataFactory
  name: dataFactoryDataSetInName
  properties: {
    linkedServiceName: {
      referenceName: dataFactoryLinkedService.name
      type: 'LinkedServiceReference'
    }
    type: 'Binary'
    typeProperties: {
      location: {
        type: 'AzureBlobStorageLocation'
        container: blobContainerName
        folderPath: 'input'
        fileName: 'emp.txt'
      }
    }
  }
}

resource dataFactoryDataSetOut 'Microsoft.DataFactory/factories/datasets@2018-06-01' = {
  parent: dataFactory
  name: dataFactoryDataSetOutName
  properties: {
    linkedServiceName: {
      referenceName: dataFactoryLinkedService.name
      type: 'LinkedServiceReference'
    }
    type: 'Binary'
    typeProperties: {
      location: {
        type: 'AzureBlobStorageLocation'
        container: blobContainerName
        folderPath: 'output'
      }
    }
  }
}

resource dataFactoryPipeline 'Microsoft.DataFactory/factories/pipelines@2018-06-01' = {
  parent: dataFactory
  name: pipelineName
  properties: {
    activities: [
      {
```

```
      {
        name: 'MyCopyActivity'
        type: 'Copy'
        typeProperties: {
          source: {
            type: 'BinarySource'
            storeSettings: {
              type: 'AzureBlobStorageReadSettings'
              recursive: true
            }
          }
          sink: {
            type: 'BinarySink'
            storeSettings: {
              type: 'AzureBlobStorageWriteSettings'
            }
          }
          enableStaging: false
        }
        inputs: [
          {
            referenceName: dataFactoryDataSetIn.name
            type: 'DatasetReference'
          }
        ]
        outputs: [
          {
            referenceName: dataFactoryDataSetOut.name
            type: 'DatasetReference'
          }
        ]
      }
    ]
  }
}
```

There are several Azure resources defined in the Bicep file:

- Microsoft.Storage/storageAccounts: Defines a storage account.
- Microsoft.DataFactory/factories: Create an Azure Data Factory.
- Microsoft.DataFactory/factories/linkedServices: Create an Azure Data Factory linked service.
- Microsoft.DataFactory/factories/datasets: Create an Azure Data Factory dataset.
- Microsoft.DataFactory/factories/pipelines: Create an Azure Data Factory pipeline.

Create a file

Open a text editor such as **Notepad**, and create a file named **emp.txt** with the following content:

```
John, Doe
Jane, Doe
```

Save the file locally. You'll use it later in the quickstart.

Deploy the Bicep file

1. Save the Bicep file from Azure Quickstart Templates as **main.bicep** to your local computer.

2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep
```

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure CLI or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

You can also use the Azure portal to review the deployed resources.

1. Sign in to the Azure portal.
2. Navigate to your resource group.
3. You'll see your resources listed. Select each resource to see an overview.

Upload a file

Use the Azure portal to upload the **emp.txt** file.

1. Navigate to your resource group and select the storage account created. Then, select the **Containers** tab on the left panel.

2. On the **Containers** page, select the blob container created. The name is in the format - blob<uniqueid>.

3. Select **Upload**, and then select the **Files** box icon in the right pane. Navigate to and select the **emp.txt** file that you created earlier.
4. Expand the **Advanced** heading.
5. In the **Upload to folder** box, enter *input*.
6. Select the **Upload** button. You should see the **emp.txt** file and the status of the upload in the list.
7. Select the **Close** icon (an **X**) to close the **Upload blob** page.

Keep the container page open because you can use it to verify the output at the end of this quickstart.

Start trigger

1. Navigate to the resource group page, and select the data factory you created.
2. Select **Open** on the **Open Azure Data Factory Studio** tile.

3. Select the **Author** tab .
4. Select the pipeline created: **ArmtemplateSampleCopyPipeline**.

5. Select **Add Trigger** > **Trigger Now**.

6. In the right pane under **Pipeline run**, select **OK**.

Monitor the pipeline

1. Select the **Monitor** tab.

2. You see the activity runs associated with the pipeline run. In this quickstart, the pipeline only has one activity of type **Copy**. You should see a run for that activity.

Activity name	Activity type	Run start	Duration	Status	Error	Log	Integration runtime	User properties	Run ID
MyCopyActivity	Copy data	Apr 25, 2022, 9:45:42 am	00:00:06	Succeeded			AutoResolveIntegrationRuntime (East US)		b5803bb6-d107-48ea-a4b5-d37a4458b190

Verify the output file

The pipeline automatically creates an output folder in the blob container. It then copies the **emp.txt** file from the input folder to the output folder.

1. On the **Containers** page in the Azure portal, select **Refresh** to see the output folder.

2. Select **output** in the folder list.

3. Confirm that the **emp.txt** is copied to the output folder.

Clean up resources

When no longer needed, use the Azure CLI or Azure PowerShell to delete the resource group and all of its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

You can also use the Azure portal to delete the resource group.

1. In the Azure portal, navigate to your resource group.
2. Select **Delete resource group**.
3. A tab will appear. Enter the resource group name and select **Delete**.

Next steps

In this quickstart, you created an Azure Data Factory using Bicep and validated the deployment. To learn more about Azure Data Factory and Bicep, continue on to the articles below.

- Azure Data Factory documentation
- Learn more about Bicep

Quickstart: Create an event hub by using Bicep

9/21/2022 • 2 minutes to read • Edit Online

Azure Event Hubs is a Big Data streaming platform and event ingestion service, capable of receiving and processing millions of events per second. Event Hubs can process and store events, data, or telemetry produced by distributed software and devices. Data sent to an event hub can be transformed and stored using any real-time analytics provider or batching/storage adapters. For detailed overview of Event Hubs, see Event Hubs overview and Event Hubs features. In this quickstart, you create an event hub by using Bicep. You deploy a Bicep file to create a namespace of type Event Hubs, with one event hub.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Specifies a project name that is used to generate the Event Hub name and the Namespace name.')
param projectName string

@description('Specifies the Azure location for all resources.')
param location string = resourceGroup().location

@description('Specifies the messaging tier for Event Hub Namespace.')
@allowed([
  'Basic'
  'Standard'
])
param eventHubSku string = 'Standard'

var eventHubNamespaceName = '${projectName}ns'
var eventHubName = projectName

resource eventHubNamespace 'Microsoft.EventHub/namespaces@2021-11-01' = {
  name: eventHubNamespaceName
  location: location
  sku: {
    name: eventHubSku
    tier: eventHubSku
    capacity: 1
  }
  properties: {
    isAutoInflateEnabled: false
    maximumThroughputUnits: 0
  }
}

resource eventHub 'Microsoft.EventHub/namespaces/eventhubs@2021-11-01' = {
  parent: eventHubNamespace
  name: eventHubName
  properties: {
    messageRetentionInDays: 7
    partitionCount: 1
  }
}
```

The resources defined in the Bicep file include:

- **Microsoft.EventHub/namespaces**
- **Microsoft.EventHub/namespaces/eventhubs**

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
projectName=<project-name>
```

> **NOTE**
> Replace **<project-name>** with a project name. It will be used to generate the Event Hubs name and the Namespace name.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Validate the deployment

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the VM and all of the resources in the resource group.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this article, you created an Event Hubs namespace and an event hub in the namespace using Bicep. For step-by-step instructions to send events to (or) receive events from an event hub, see the **Send and receive events** tutorials:

- .NET Core
- Java
- Python
- JavaScript
- Go
- C (send only)
- Apache Storm (receive only)

Quickstart: Create Apache Hadoop cluster in Azure HDInsight using Bicep

9/21/2022 • 4 minutes to read • Edit Online

In this quickstart, you use Bicep to create an Apache Hadoop cluster in Azure HDInsight. Hadoop was the original open-source framework for distributed processing and analysis of big data sets on clusters. The Hadoop ecosystem includes related software and utilities, including Apache Hive, Apache HBase, Spark, Kafka, and many others.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Currently HDInsight comes with seven different cluster types. Each cluster type supports a different set of components. All cluster types support Hive. For a list of supported components in HDInsight, see What's new in the Hadoop cluster versions provided by HDInsight?

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('The name of the HDInsight cluster to create.')
param clusterName string

@description('The type of the HDInsight cluster to create.')
@allowed([
  'hadoop'
  'intractivehive'
  'hbase'
  'storm'
  'spark'
])
param clusterType string

@description('These credentials can be used to submit jobs to the cluster and to log into cluster
dashboards.')
param clusterLoginUserName string

@description('The password must be at least 10 characters in length and must contain at least one digit, one
upper case letter, one lower case letter, and one non-alphanumeric character except (single-quote, double-
quote, backslash, right-bracket, full-stop). Also, the password must not contain 3 consecutive characters
from the cluster username or SSH username.')
@minLength(10)
@secure()
param clusterLoginPassword string

@description('These credentials can be used to remotely access the cluster. The username cannot be admin.')
param sshUserName string

@description('SSH password must be 6-72 characters long and must contain at least one digit, one upper case
letter, and one lower case letter.  It must not contain any 3 consecutive characters from the cluster login
name')
@minLength(6)
```

```
@minLength(6)
@maxLength(72)
@secure()
param sshPassword string

@description('Location for all resources.')
param location string = resourceGroup().location

@description('This is the headnode Azure Virtual Machine size, and will affect the cost. If you don\'t know,
just leave the default value.')
@allowed([
  'Standard_A4_v2'
  'Standard_A8_v2'
  'Standard_E2_v3'
  'Standard_E4_v3'
  'Standard_E8_v3'
  'Standard_E16_v3'
  'Standard_E20_v3'
  'Standard_E32_v3'
  'Standard_E48_v3'
])
param HeadNodeVirtualMachineSize string = 'Standard_E4_v3'

@description('This is the workdernode Azure Virtual Machine size, and will affect the cost. If you don\'t
know, just leave the default value.')
@allowed([
  'Standard_A4_v2'
  'Standard_A8_v2'
  'Standard_E2_v3'
  'Standard_E4_v3'
  'Standard_E8_v3'
  'Standard_E16_v3'
  'Standard_E20_v3'
  'Standard_E32_v3'
  'Standard_E48_v3'
])
param WorkerNodeVirtualMachineSize string = 'Standard_E4_v3'

var defaultStorageAccount = {
  name: uniqueString(resourceGroup().id)
  type: 'Standard_LRS'
}

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-08-01' = {
  name: defaultStorageAccount.name
  location: location
  sku: {
    name: defaultStorageAccount.type
  }
  kind: 'StorageV2'
  properties: {}
}

resource cluster 'Microsoft.HDInsight/clusters@2021-06-01' = {
  name: clusterName
  location: location
  properties: {
    clusterVersion: '4.0'
    osType: 'Linux'
    clusterDefinition: {
      kind: clusterType
      configurations: {
        gateway: {
          'restAuthCredential.isEnabled': true
          'restAuthCredential.username': clusterLoginUserName
          'restAuthCredential.password': clusterLoginPassword
        }
      }
    }
```

```
    storageProfile: {
      storageaccounts: [
        {
          name: replace(replace(concat(storageAccount.properties.primaryEndpoints.blob), 'https:', ''), '/',
''))
          isDefault: true
          container: clusterName
          key: listKeys(storageAccount.id, '2021-08-01').keys[0].value
        }
      ]
    }
    computeProfile: {
      roles: [
        {
          name: 'headnode'
          targetInstanceCount: 2
          hardwareProfile: {
            vmSize: HeadNodeVirtualMachineSize
          }
          osProfile: {
            linuxOperatingSystemProfile: {
              username: sshUserName
              password: sshPassword
            }
          }
        }
        {
          name: 'workernode'
          targetInstanceCount: 2
          hardwareProfile: {
            vmSize: WorkerNodeVirtualMachineSize
          }
          osProfile: {
            linuxOperatingSystemProfile: {
              username: sshUserName
              password: sshPassword
            }
          }
        }
      ]
    }
  }
}

output storage object = storageAccount.properties
output cluster object = cluster.properties
```

Two Azure resources are defined in the Bicep file:

- Microsoft.Storage/storageAccounts: create an Azure Storage Account.
- Microsoft.HDInsight/cluster: create an HDInsight cluster.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.
   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
clusterName=<cluster-name> clusterType=<cluster-type> clusterLoginUserName=<cluster-username>
sshUserName=<ssh-username>
```

You need to provide values for the parameters:

- Replace **<cluster-name>** with the name of the HDInsight cluster to create.
- Replace **<cluster-type>** with the type of the HDInsight cluster to create. Allowed strings include: `hadoop`, `interactivehive`, `hbase`, `storm`, and `spark`.
- Replace **<cluster-username>** with the credentials used to submit jobs to the cluster and to log in to cluster dashboards.
- Replace **<ssh-username>** with the credentials used to remotely access the cluster. The username cannot be admin.

You'll also be prompted to enter the following:

- **clusterLoginPassword**, which must be at least 10 characters long and contain one digit, one uppercase letter, one lowercase letter, and one non-alphanumeric character except single-quote, double-quote, backslash, right-bracket, full-stop. It also must not contain three consecutive characters from the cluster username or SSH username.
- **sshPassword**, which must be 6-72 characters long and must contain at least one digit, one uppercase letter, and one lowercase letter. It must not contain any three consecutive characters from the cluster login name.

> **NOTE**
> When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you learned how to create an Apache Hadoop cluster in HDInsight using Bicep. In the next

article, you learn how to perform an extract, transform, and load (ETL) operation using Hadoop on HDInsight.

Extract, transform, and load data using Interactive Query on HDInsight

Quickstart: Create Apache HBase cluster in Azure HDInsight using Bicep

9/21/2022 • 4 minutes to read • Edit Online

In this quickstart, you use Bicep to create an Apache HBase cluster in Azure HDInsight. HBase is an open-source, NoSQL database that is built on Apache Hadoop and modeled after Google BigTable.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('The name of the HDInsight cluster to create.')
param clusterName string

@description('These credentials can be used to submit jobs to the cluster and to log into cluster
dashboards.')
param clusterLoginUserName string

@description('The password must be at least 10 characters in length and must contain at least one digit, one
upper case letter, one lower case letter, and one non-alphanumeric character except (single-quote, double-
quote, backslash, right-bracket, full-stop). Also, the password must not contain 3 consecutive characters
from the cluster username or SSH username.')
@minLength(10)
@secure()
param clusterLoginPassword string

@description('These credentials can be used to remotely access the cluster.')
param sshUserName string

@description('SSH password must be 6-72 characters long and must contain at least one digit, one upper case
letter, and one lower case letter.  It must not contain any 3 consecutive characters from the cluster login
name')
@minLength(6)
@maxLength(72)
@secure()
param sshPassword string

@description('Location for all resources.')
param location string = resourceGroup().location

@description('This is the headnode Azure Virtual Machine size, and will affect the cost. If you don\'t know,
just leave the default value.')
@allowed([
  'Standard_A4_v2'
  'Standard_A8_v2'
  'Standard_E2_v3'
  'Standard_E4_v3'
  'Standard_E8_v3'
  'Standard_E16_v3'
  'Standard_E20_v3'
```

```
  'Standard_E32_v3'
  'Standard_E48_v3'
])
param HeadNodeVirtualMachineSize string = 'Standard_E4_v3'

@description('This is the worerdnode Azure Virtual Machine size, and will affect the cost. If you don\'t
know, just leave the default value.')
@allowed([
  'Standard_A4_v2'
  'Standard_A8_v2'
  'Standard_E2_v3'
  'Standard_E4_v3'
  'Standard_E8_v3'
  'Standard_E16_v3'
  'Standard_E20_v3'
  'Standard_E32_v3'
  'Standard_E48_v3'
])
param WorkerNodeVirtualMachineSize string = 'Standard_E4_v3'

@description('This is the Zookeepernode Azure Virtual Machine size, and will affect the cost. If you don\'t
know, just leave the default value.')
@allowed([
  'Standard_A4_v2'
  'Standard_A8_v2'
  'Standard_E2_v3'
  'Standard_E4_v3'
  'Standard_E8_v3'
  'Standard_E16_v3'
  'Standard_E20_v3'
  'Standard_E32_v3'
  'Standard_E48_v3'
])
param ZookeeperNodeVirtualMachineSize string = 'Standard_E4_v3'

var defaultStorageAccount = {
  name: uniqueString(resourceGroup().id)
  type: 'Standard_LRS'
}

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-08-01' = {
  name: defaultStorageAccount.name
  location: location
  sku: {
    name: defaultStorageAccount.type
  }
  kind: 'Storage'
  properties: {}
}

resource cluster 'Microsoft.HDInsight/clusters@2021-06-01' = {
  name: clusterName
  location: location
  properties: {
    clusterVersion: '4.0'
    osType: 'Linux'
    clusterDefinition: {
      kind: 'hbase'
      configurations: {
        gateway: {
          'restAuthCredential.isEnabled': true
          'restAuthCredential.username': clusterLoginUserName
          'restAuthCredential.password': clusterLoginPassword
        }
      }
    }
    storageProfile: {
      storageaccounts: [
        {
```

```
          {
            name: replace(replace(reference(storageAccount.id, '2021-08-01').primaryEndpoints.blob,
'https://', ''), '/', '')
            isDefault: true
            container: clusterName
            key: listKeys(storageAccount.id, '2021-08-01').keys[0].value
          }
        ]
      }
      computeProfile: {
        roles: [
          {
            name: 'headnode'
            targetInstanceCount: 2
            hardwareProfile: {
              vmSize: HeadNodeVirtualMachineSize
            }
            osProfile: {
              linuxOperatingSystemProfile: {
                username: sshUserName
                password: sshPassword
              }
            }
          }
          {
            name: 'workernode'
            targetInstanceCount: 2
            hardwareProfile: {
              vmSize: WorkerNodeVirtualMachineSize
            }
            osProfile: {
              linuxOperatingSystemProfile: {
                username: sshUserName
                password: sshPassword
              }
            }
          }
          {
            name: 'zookeepernode'
            targetInstanceCount: 3
            hardwareProfile: {
              vmSize: ZookeeperNodeVirtualMachineSize
            }
            osProfile: {
              linuxOperatingSystemProfile: {
                username: sshUserName
                password: sshPassword
              }
            }
          }
        ]
      }
    }
}

output cluster object = cluster.properties
```

Two Azure resources are defined in the Bicep file:

- Microsoft.Storage/storageAccounts: create an Azure Storage Account.
- Microsoft.HDInsight/cluster: create an HDInsight cluster.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

- CLI
- PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
clusterName=<cluster-name> clusterLoginUserName=<cluster-username> sshUserName=<ssh-username>
```

You need to provide values for the parameters:

- Replace **<cluster-name>** with the name of the HDInsight cluster to create.
- Replace **<cluster-username>** with the credentials used to submit jobs to the cluster and to log in to cluster dashboards.
- Replace **<ssh-username>** with the credentials used to remotely access the cluster.

You'll be prompted to enter the following:

- **clusterLoginPassword**, which must be at least 10 characters long and must contain at least one digit, one uppercase letter, one lowercase letter, and one non-alphanumeric character except single-quote, double-quote, backslash, right-bracket, full-stop. It also must not contain three consecutive characters from the cluster username or SSH username.
- **sshPassword**, which must be 6-72 characters long and must contain at least one digit, one uppercase letter, and one lowercase letter. It must not contain any three consecutive characters from the cluster login name.

> **NOTE**
> When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you learned how to create an Apache HBase cluster in HDInsight using Bicep. In the next

article, you learn how to query HBase in HDInsight with HBase Shell.

Query Apache HBase in Azure HDInsight with HBase Shell

Quickstart: Create Interactive Query cluster in Azure HDInsight using Bicep

9/21/2022 • 4 minutes to read • Edit Online

In this quickstart, you use a Bicep to create an Interactive Query cluster in Azure HDInsight. Interactive Query (also called Apache Hive LLAP, or Low Latency Analytical Processing) is an Azure HDInsight cluster type.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('The name of the HDInsight cluster to create.')
param clusterName string

@description('These credentials can be used to submit jobs to the cluster and to log into cluster
dashboards.')
param clusterLoginUserName string

@description('The password must be at least 10 characters in length and must contain at least one digit, one
upper case letter, one lower case letter, and one non-alphanumeric character except (single-quote, double-
quote, backslash, right-bracket, full-stop). Also, the password must not contain 3 consecutive characters
from the cluster username or SSH username.')
@minLength(10)
@secure()
param clusterLoginPassword string

@description('These credentials can be used to remotely access the cluster.')
param sshUserName string

@description('SSH password must be 6-72 characters long and must contain at least one digit, one upper case
letter, and one lower case letter.  It must not contain any 3 consecutive characters from the cluster login
name')
@minLength(6)
@maxLength(72)
@secure()
param sshPassword string

@description('Location for all resources.')
param location string = resourceGroup().location

@description('This is the headnode Azure Virtual Machine size, and will affect the cost. If you don\'t know,
just leave the default value.')
@allowed([
  'Standard_A4_v2'
  'Standard_A8_v2'
  'Standard_E2_v3'
  'Standard_E4_v3'
  'Standard_E8_v3'
  'Standard_E16_v3'
  'Standard_E20_v3'
```

```
  'Standard_E32_v3'
  'Standard_E48_v3'
])
param HeadNodeVirtualMachineSize string = 'Standard_E8_v3'

@description('This is the worerdnode Azure Virtual Machine size, and will affect the cost. If you don\'t
know, just leave the default value.')
@allowed([
  'Standard_A4_v2'
  'Standard_A8_v2'
  'Standard_E2_v3'
  'Standard_E4_v3'
  'Standard_E8_v3'
  'Standard_E16_v3'
  'Standard_E20_v3'
  'Standard_E32_v3'
  'Standard_E48_v3'
])
param WorkerNodeVirtualMachineSize string = 'Standard_E16_v3'

@description('This is the worerdnode Azure Virtual Machine size, and will affect the cost. If you don\'t
know, just leave the default value.')
@allowed([
  'Standard_A4_v2'
  'Standard_A8_v2'
  'Standard_E2_v3'
  'Standard_E4_v3'
  'Standard_E8_v3'
  'Standard_E16_v3'
  'Standard_E20_v3'
  'Standard_E32_v3'
  'Standard_E48_v3'
])
param ZookeeperNodeVirtualMachineSize string = 'Standard_E4_v3'

var defaultStorageAccount = {
  name: uniqueString(resourceGroup().id)
  type: 'Standard_LRS'
}

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-08-01' = {
  name: defaultStorageAccount.name
  location: location
  sku: {
    name: defaultStorageAccount.type
  }
  kind: 'StorageV2'
  properties: {}
}

resource cluster 'Microsoft.HDInsight/clusters@2021-06-01' = {
  name: clusterName
  location: location
  properties: {
    clusterVersion: '4.0'
    osType: 'Linux'
    tier: 'Standard'
    clusterDefinition: {
      kind: 'interactivehive'
      configurations: {
        gateway: {
          'restAuthCredential.isEnabled': true
          'restAuthCredential.username': clusterLoginUserName
          'restAuthCredential.password': clusterLoginPassword
        }
      }
    }
    storageProfile: {
      storageaccounts: [
```

```
      storageAccounts: [
        {
          name: replace(replace(concat(reference(storageAccount.id, '2021-08-01').primaryEndpoints.blob),
'https:', ''), '/', '')
          isDefault: true
          container: clusterName
          key: listKeys(storageAccount.id, '2021-08-01').keys[0].value
        }
      ]
    }
    computeProfile: {
      roles: [
        {
          name: 'headnode'
          minInstanceCount: 1
          targetInstanceCount: 2
          hardwareProfile: {
            vmSize: HeadNodeVirtualMachineSize
          }
          osProfile: {
            linuxOperatingSystemProfile: {
              username: sshUserName
              password: sshPassword
            }
          }
        }
        {
          name: 'workernode'
          minInstanceCount: 1
          targetInstanceCount: 2
          hardwareProfile: {
            vmSize: WorkerNodeVirtualMachineSize
          }
          osProfile: {
            linuxOperatingSystemProfile: {
              username: sshUserName
              password: sshPassword
            }
          }
        }
        {
          name: 'zookeepernode'
          minInstanceCount: 1
          targetInstanceCount: 3
          hardwareProfile: {
            vmSize: ZookeeperNodeVirtualMachineSize
          }
          osProfile: {
            linuxOperatingSystemProfile: {
              username: sshUserName
              password: sshPassword
            }
          }
        }
      ]
    }
  }
}

output storage object = storageAccount.properties
output cluster object = cluster.properties
```

Two Azure resources are defined in the Bicep file:

- Microsoft.Storage/storageAccounts: create an Azure Storage Account.
- Microsoft.HDInsight/cluster: create an HDInsight cluster.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
clusterName=<cluster-name> clusterLoginUserName=<cluster-username> sshUserName=<ssh-username>
```

   You need to provide values for the parameters:

   - Replace **<cluster-name>** with the name of the HDInsight cluster to create.
   - Replace **<cluster-username>** with the credentials used to submit jobs to the cluster and to log in to cluster dashboards.
   - Replace **<ssh-username>** with the credentials used to remotely access the cluster. The username cannot be admin.

   You'll also be prompted to enter the following:

   - **clusterLoginPassword**, which must be at least 10 characters long and contain one digit, one uppercase letter, one lowercase letter, and one non-alphanumeric character except single-quote, double-quote, backslash, right-bracket, full-stop. It also must not contain three consecutive characters from the cluster username or SSH username.
   - **sshPassword**, which must be 6-72 characters long and must contain at least one digit, one uppercase letter, and one lowercase letter. It must not contain any three consecutive characters from the cluster login name.

   > **NOTE**
   > When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you learned how to create an Interactive Query cluster in HDInsight using Bicep. In the next article, you learn how to use Apache Zeppelin to run Apache Hive queries.

Execute Apache Hive queries in Azure HDInsight with Apache Zeppelin

Quickstart: Create Apache Kafka cluster in Azure HDInsight using Bicep

9/21/2022 • 9 minutes to read • Edit Online

In this quickstart, you use a Bicep to create an Apache Kafka cluster in Azure HDInsight. Kafka is an open-source, distributed streaming platform. It's often used as a message broker, as it provides functionality similar to a publish-subscribe message queue.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

The Kafka API can only be accessed by resources inside the same virtual network. In this quickstart, you access the cluster directly using SSH. To connect other services, networks, or virtual machines to Kafka, you must first create a virtual network and then create the resources within the network. For more information, see the Connect to Apache Kafka using a virtual network document.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('The name of the Kafka cluster to create. This must be a unique name.')
param clusterName string

@description('These credentials can be used to submit jobs to the cluster and to log into cluster
dashboards.')
param clusterLoginUserName string

@description('The password must be at least 10 characters in length and must contain at least one digit, one
upper case letter, one lower case letter, and one non-alphanumeric character except (single-quote, double-
quote, backslash, right-bracket, full-stop). Also, the password must not contain 3 consecutive characters
from the cluster username or SSH username.')
@minLength(10)
@secure()
param clusterLoginPassword string

@description('These credentials can be used to remotely access the cluster.')
param sshUserName string

@description('SSH password must be 6-72 characters long and must contain at least one digit, one upper case
letter, and one lower case letter.  It must not contain any 3 consecutive characters from the cluster login
name')
@minLength(6)
@maxLength(72)
@secure()
param sshPassword string

@description('Location for all resources.')
param location string = resourceGroup().location

@description('This is the headnode Azure Virtual Machine size, and will affect the cost. If you don\'t know,
just leave the default value.')
@allowed([
```

```
@allowed([
  'Standard_A4_v2'
  'Standard_A8_v2'
  'Standard_E2_v3'
  'Standard_E4_v3'
  'Standard_E8_v3'
  'Standard_E16_v3'
  'Standard_E20_v3'
  'Standard_E32_v3'
  'Standard_E48_v3'
])
param HeadNodeVirtualMachineSize string = 'Standard_E4_v3'

@description('This is the worerdnode Azure Virtual Machine size, and will affect the cost. If you don\'t
know, just leave the default value.')
@allowed([
  'Standard_A4_v2'
  'Standard_A8_v2'
  'Standard_E2_v3'
  'Standard_E4_v3'
  'Standard_E8_v3'
  'Standard_E16_v3'
  'Standard_E20_v3'
  'Standard_E32_v3'
  'Standard_E48_v3'
])
param WorkerNodeVirtualMachineSize string = 'Standard_E4_v3'

@description('This is the Zookeepernode Azure Virtual Machine size, and will affect the cost. If you don\'t
know, just leave the default value.')
@allowed([
  'Standard_A4_v2'
  'Standard_A8_v2'
  'Standard_E2_v3'
  'Standard_E4_v3'
  'Standard_E8_v3'
  'Standard_E16_v3'
  'Standard_E20_v3'
  'Standard_E32_v3'
  'Standard_E48_v3'
])
param ZookeeperNodeVirtualMachineSize string = 'Standard_E4_v3'

var defaultStorageAccount = {
  name: uniqueString(resourceGroup().id)
  type: 'Standard_LRS'
}

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-08-01' = {
  name: defaultStorageAccount.name
  location: location
  sku: {
    name: defaultStorageAccount.type
  }
  kind: 'StorageV2'
  properties: {}
}

resource cluster 'Microsoft.HDInsight/clusters@2021-06-01' = {
  name: clusterName
  location: location
  properties: {
    clusterVersion: '4.0'
    osType: 'Linux'
    clusterDefinition: {
      kind: 'kafka'
      configurations: {
        gateway: {
          'restAuthCredential.isEnabled': true
```

```
          'restAuthCredential.username': clusterLoginUserName
          'restAuthCredential.password': clusterLoginPassword
        }
      }
    }
    storageProfile: {
      storageaccounts: [
        {
          name: replace(replace(concat(reference(storageAccount.id, '2021-08-01').primaryEndpoints.blob),
'https:', ''), '/', '')
          isDefault: true
          container: clusterName
          key: listKeys(storageAccount.id, '2021-08-01').keys[0].value
        }
      ]
    }
    computeProfile: {
      roles: [
        {
          name: 'headnode'
          targetInstanceCount: 2
          hardwareProfile: {
            vmSize: HeadNodeVirtualMachineSize
          }
          osProfile: {
            linuxOperatingSystemProfile: {
              username: sshUserName
              password: sshPassword
            }
          }
        }
        {
          name: 'workernode'
          targetInstanceCount: 4
          hardwareProfile: {
            vmSize: WorkerNodeVirtualMachineSize
          }
          dataDisksGroups: [
            {
              disksPerNode: 2
            }
          ]
          osProfile: {
            linuxOperatingSystemProfile: {
              username: sshUserName
              password: sshPassword
            }
          }
        }
        {
          name: 'zookeepernode'
          targetInstanceCount: 3
          hardwareProfile: {
            vmSize: ZookeeperNodeVirtualMachineSize
          }
          osProfile: {
            linuxOperatingSystemProfile: {
              username: sshUserName
              password: sshPassword
            }
          }
        }
      ]
    }
  }
}

output cluster object = cluster.properties
```

Two Azure resources are defined in the Bicep file:

- Microsoft.Storage/storageAccounts: create an Azure Storage Account.
- Microsoft.HDInsight/cluster: create an HDInsight cluster.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

   ```
   az group create --name exampleRG --location eastus
   az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
   clusterName=<cluster-name> clusterLoginUserName=<cluster-username> sshUserName=<ssh-username>
   ```

   You need to provide values for the parameters:

   - Replace **<cluster-name>** with the name of the HDInsight cluster to create. The cluster name needs to start with a letter and can contain only lowercase letters, numbers, and dashes.
   - Replace **<cluster-username>** with the credentials used to submit jobs to the cluster and to log in to cluster dashboards. Uppercase letters aren't allowed in the cluster username.
   - Replace **<ssh-username>** with the credentials used to remotely access the cluster.

   You'll be prompted to enter the following:

   - **clusterLoginPassword**, which must be at least 10 characters long and contain at least one digit, one uppercase letter, one lowercase letter, and one non-alphanumeric character except single-quote, double-quote, backslash, right-bracket, full-stop. It also must not contain three consecutive characters from the cluster username of SSH username.
   - **sshPassword**, which must be 6-72 characters long and must contain at least one digit, one uppercase letter, and one lowercase letter. It must not contain any three consecutive characters from the cluster login name.

   > **NOTE**
   > When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Get the Apache Zookeeper and Broker host information

When working with Kafka, you must know the *Apache Zookeeper* and *Broker* hosts. These hosts are used with the Kafka API and many of the utilities that ship with Kafka.

In this section, you get the host information from the Ambari REST API on the cluster.

1. Use ssh command to connect to your cluster. Edit the command below by replacing CLUSTERNAME with the name of your cluster, and then enter the command:

   ```
   ssh sshuser@CLUSTERNAME-ssh.azurehdinsight.net
   ```

2. From the SSH connection, use the following command to install the `jq` utility. This utility is used to parse JSON documents, and is useful in retrieving the host information:

   ```
   sudo apt -y install jq
   ```

3. To set an environment variable to the cluster name, use the following command:

   ```
   read -p "Enter the Kafka on HDInsight cluster name: " CLUSTERNAME
   ```

   When prompted, enter the name of the Kafka cluster.

4. To set an environment variable with Zookeeper host information, use the command below. The command retrieves all Zookeeper hosts, then returns only the first two entries. This is because you want some redundancy in case one host is unreachable.

   ```
   export KAFKAZKHOSTS=`curl -sS -u admin -G
   https://$CLUSTERNAME.azurehdinsight.net/api/v1/clusters/$CLUSTERNAME/services/ZOOKEEPER/components/ZO
   OKEEPER_SERVER | jq -r '["\(.host_components[].HostRoles.host_name):2181"] | join(",")' | cut -d',' -
   f1,2`
   ```

   When prompted, enter the password for the cluster login account (not the SSH account).

5. To verify that the environment variable is set correctly, use the following command:

   ```
   echo '$KAFKAZKHOSTS='$KAFKAZKHOSTS
   ```

   This command returns information similar to the following text:

   ```
   <zookeepername1>.eahjefxxp1netdbyklgqj5y1ud.ex.internal.cloudapp.net:2181,
   <zookeepername2>.eahjefxxp1netdbyklgqj5y1ud.ex.internal.cloudapp.net:2181
   ```

6. To set an environment variable with Kafka broker host information, use the following command:

   ```
   export KAFKABROKERS=`curl -sS -u admin -G
   https://$CLUSTERNAME.azurehdinsight.net/api/v1/clusters/$CLUSTERNAME/services/KAFKA/components/KAFKA_
   BROKER | jq -r '["\(.host_components[].HostRoles.host_name):9092"] | join(",")' | cut -d',' -f1,2`
   ```

   When prompted, enter the password for the cluster login account (not the SSH account).

7. To verify that the environment variable is set correctly, use the following command:

   ```
   echo '$KAFKABROKERS='$KAFKABROKERS
   ```

   This command returns information similar to the following text:

   ```
   <brokername1>.eahjefxxp1netdbyklgqj5y1ud.cx.internal.cloudapp.net:9092,
   <brokername2>.eahjefxxp1netdbyklgqj5y1ud.cx.internal.cloudapp.net:9092
   ```

Manage Apache Kafka topics

Kafka stores streams of data in *topics*. You can use the `kafka-topics.sh` utility to manage topics.

- **To create a topic**, use the following command in the SSH connection:

  ```
  /usr/hdp/current/kafka-broker/bin/kafka-topics.sh --create --replication-factor 3 --partitions 8 --
  topic test --zookeeper $KAFKAZKHOSTS
  ```

  This command connects to Zookeeper using the host information stored in `$KAFKAZKHOSTS`. It then creates a Kafka topic named **test**.

  - Data stored in this topic is partitioned across eight partitions.
  - Each partition is replicated across three worker nodes in the cluster.

    If you created the cluster in an Azure region that provides three fault domains, use a replication factor of 3. Otherwise, use a replication factor of 4.

    In regions with three fault domains, a replication factor of 3 allows replicas to be spread across the fault domains. In regions with two fault domains, a replication factor of four spreads the replicas evenly across the domains.

    For information on the number of fault domains in a region, see the Availability of Linux virtual machines document.

    Kafka isn't aware of Azure fault domains. When creating partition replicas for topics, it may not distribute replicas properly for high availability.

    To ensure high availability, use the Apache Kafka partition rebalance tool. This tool must be ran from an SSH connection to the head node of your Kafka cluster.

    For the highest availability of your Kafka data, you should rebalance the partition replicas for your topic when:

    - You create a new topic or partition
    - You scale up a cluster

- **To list topics**, use the following command:

  ```
  /usr/hdp/current/kafka-broker/bin/kafka-topics.sh --list --zookeeper $KAFKAZKHOSTS
  ```

  This command lists the topics available on the Kafka cluster.

- **To delete a topic**, use the following command:

  ```
  /usr/hdp/current/kafka-broker/bin/kafka-topics.sh --delete --topic topicname --zookeeper
  $KAFKAZKHOSTS
  ```

  This command deletes the topic named `topicname`.

  > **WARNING**
  > If you delete the `test` topic created earlier, then you must recreate it. It is used by steps later in this document.

For more information on the commands available with the `kafka-topics.sh` utility, use the following command:

```
/usr/hdp/current/kafka-broker/bin/kafka-topics.sh
```

Produce and consume records

Kafka stores *records* in topics. Records are produced by *producers*, and consumed by *consumers*. Producers and consumers communicate with the *Kafka broker* service. Each worker node in your HDInsight cluster is a Kafka broker host.

To store records into the test topic you created earlier, and then read them using a consumer, use the following steps:

1. To write records to the topic, use the `kafka-console-producer.sh` utility from the SSH connection:

   ```
   /usr/hdp/current/kafka-broker/bin/kafka-console-producer.sh --broker-list $KAFKABROKERS --topic test
   ```

   After this command, you arrive at an empty line.

2. Type a text message on the empty line and hit enter. Enter a few messages this way, and then use **Ctrl + C** to return to the normal prompt. Each line is sent as a separate record to the Kafka topic.

3. To read records from the topic, use the `kafka-console-consumer.sh` utility from the SSH connection:

   ```
   /usr/hdp/current/kafka-broker/bin/kafka-console-consumer.sh --bootstrap-server $KAFKABROKERS --topic
   test --from-beginning
   ```

   This command retrieves the records from the topic and displays them. Using `--from-beginning` tells the consumer to start from the beginning of the stream, so all records are retrieved.

   If you're using an older version of Kafka, replace `--bootstrap-server $KAFKABROKERS` with `--zookeeper $KAFKAZKHOSTS`.

4. Use **Ctrl + C** to stop the consumer.

You can also programmatically create producers and consumers. For an example of using this API, see the Apache Kafka Producer and Consumer API with HDInsight document.

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you learned how to create an Apache Kafka cluster in HDInsight using Bicep. In the next article, you learn how to create an application that uses the Apache Kafka Streams API and run it with Kafka on HDInsight.

Use Apache Kafka streams API in Azure HDInsight

Quickstart: Create Apache Spark cluster in Azure HDInsight using Bicep

9/21/2022 • 7 minutes to read • Edit Online

In this quickstart, you use Bicep to create an Apache Spark cluster in Azure HDInsight. You then create a Jupyter Notebook file, and use it to run Spark SQL queries against Apache Hive tables. Azure HDInsight is a managed, full-spectrum, open-source analytics service for enterprises. The Apache Spark framework for HDInsight enables fast data analytics and cluster computing using in-memory processing. Jupyter Notebook lets you interact with your data, combine code with markdown text, and do simple visualizations.

If you're using multiple clusters together, you'll want to create a virtual network, and if you're using a Spark cluster you'll also want to use the Hive Warehouse Connector. For more information, see Plan a virtual network for Azure HDInsight and Integrate Apache Spark and Apache Hive with the Hive Warehouse Connector.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('The name of the HDInsight cluster to create.')
param clusterName string

@description('These credentials can be used to submit jobs to the cluster and to log into cluster
dashboards. The username must consist of digits, upper or lowercase letters, and/or the following special
characters: (!#$%&\'()-^_`{}~).')
@minLength(2)
@maxLength(20)
param clusterLoginUserName string

@description('The password must be at least 10 characters in length and must contain at least one digit, one
upper case letter, one lower case letter, and one non-alphanumeric character except (single-quote, double-
quote, backslash, right-bracket, full-stop). Also, the password must not contain 3 consecutive characters
from the cluster username or SSH username.')
@minLength(10)
@secure()
param clusterLoginPassword string

@description('These credentials can be used to remotely access the cluster. The sshUserName can only consit
of digits, upper or lowercase letters, and/or the following special characters (%&\'^_`{}~). Also, it cannot
be the same as the cluster login username or a reserved word')
@minLength(2)
param sshUserName string

@description('SSH password must be 6-72 characters long and must contain at least one digit, one upper case
letter, and one lower case letter.  It must not contain any 3 consecutive characters from the cluster login
name')
@minLength(6)
@maxLength(72)
@secure()
```

```
param sshPassword string

@description('Location for all resources.')
param location string = resourceGroup().location

@description('This is the headnode Azure Virtual Machine size, and will affect the cost. If you don\'t know,
just leave the default value.')
@allowed([
  'Standard_A4_v2'
  'Standard_A8_v2'
  'Standard_E2_v3'
  'Standard_E4_v3'
  'Standard_E8_v3'
  'Standard_E16_v3'
  'Standard_E20_v3'
  'Standard_E32_v3'
  'Standard_E48_v3'
])
param headNodeVirtualMachineSize string = 'Standard_E8_v3'

@description('This is the workernode Azure Virtual Machine size, and will affect the cost. If you don\'t
know, just leave the default value.')
@allowed([
  'Standard_A4_v2'
  'Standard_A8_v2'
  'Standard_E2_v3'
  'Standard_E4_v3'
  'Standard_E8_v3'
  'Standard_E16_v3'
  'Standard_E20_v3'
  'Standard_E32_v3'
  'Standard_E48_v3'
])
param workerNodeVirtualMachineSize string = 'Standard_E8_v3'

resource defaultStorageAccount 'Microsoft.Storage/storageAccounts@2021-08-01' = {
  name: 'storage${uniqueString(resourceGroup().id)}'
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
}

resource cluster 'Microsoft.HDInsight/clusters@2021-06-01' = {
  name: clusterName
  location: location
  properties: {
    clusterVersion: '4.0'
    osType: 'Linux'
    tier: 'Standard'
    clusterDefinition: {
      kind: 'spark'
      configurations: {
        gateway: {
          'restAuthCredential.isEnabled': true
          'restAuthCredential.username': clusterLoginUserName
          'restAuthCredential.password': clusterLoginPassword
        }
      }
    }
    storageProfile: {
      storageaccounts: [
        {
          name: replace(replace(defaultStorageAccount.properties.primaryEndpoints.blob, 'https://', ''),
'/', '')
          isDefault: true
          container: clusterName
          key: defaultStorageAccount.listKeys('2021-08-01').keys[0].value
```

```
        }
      ]
    }
    computeProfile: {
      roles: [
        {
          name: 'headnode'
          targetInstanceCount: 2
          hardwareProfile: {
            vmSize: headNodeVirtualMachineSize
          }
          osProfile: {
            linuxOperatingSystemProfile: {
              username: sshUserName
              password: sshPassword
            }
          }
        }
        {
          name: 'workernode'
          targetInstanceCount: 2
          hardwareProfile: {
            vmSize: workerNodeVirtualMachineSize
          }
          osProfile: {
            linuxOperatingSystemProfile: {
              username: sshUserName
              password: sshPassword
            }
          }
        }
      ]
    }
  }
}

output storage object = defaultStorageAccount.properties
output cluster object = cluster.properties
```

Two Azure resources are defined in the Bicep file:

- Microsoft.Storage/storageAccounts: create an Azure Storage Account.
- Microsoft.HDInsight/cluster: create an HDInsight cluster.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

   ```
   az group create --name exampleRG --location eastus
   az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
   clusterName=<cluster-name> clusterLoginUserName=<cluster-username> sshUserName=<ssh-username>
   ```

   You need to provide values for the parameters:

   - Replace **<cluster-name>** with the name of the HDInsight cluster to create.
   - Replace **<cluster-username>** with the credentials used to submit jobs to the cluster and to log in to cluster dashboards. The username has a minimum length of two characters and a maximum length of

20 characters. It must consist of digits, upper or lowercase letters, and/or the following special characters: (!#$%&'()-^_`{}~).').

- Replace **<ssh-username>** with the credentials used to remotely access the cluster. The username has a minimum length of two characters. It must consist of digits, upper or lowercase letters, and/or the following special characters: (%&'^_`{}~). It cannot be the same as the cluster username.

You'll be prompted to enter the following:

- **clusterLoginPassword**, which must be at least 10 characters long and must contain at least one digit, one uppercase letter, one lowercase letter, and one non-alphanumeric character except single-quote, double-quote, backslash, right-bracket, full-stop. It also must not contain three consecutive characters from the cluster username or SSH username.
- **sshPassword**, which must be 6-72 characters long and must contain at least one digit, one uppercase letter, and one lowercase letter. It must not contain any three consecutive characters from the cluster login name.

> **NOTE**
>
> When the deployment finishes, you should see a message indicating the deployment succeeded.

If you run into an issue with creating HDInsight clusters, it could be that you don't have the right permissions to do so. For more information, see Access control requirements.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Create a Jupyter Notebook file

Jupyter Notebook is an interactive notebook environment that supports various programming languages. You can use a Jupyter Notebook file to interact with your data, combine code with markdown text, and perform simple visualizations.

1. Open the Azure portal.
2. Select **HDInsight clusters**, and then select the cluster you created.

3. From the portal, in **Cluster dashboards** section, select **Jupyter Notebook**. If prompted, enter the cluster login credentials for the cluster.

4. Select **New** > **PySpark** to create a notebook.

A new notebook is created and opened with the name Untitled(Untitled.pynb).

Run Apache Spark SQL statements

SQL (Structured Query Language) is the most common and widely used language for querying and transforming data. Spark SQL functions as an extension to Apache Spark for processing structured data, using the familiar SQL syntax.

1. Verify the kernel is ready. The kernel is ready when you see a hollow circle next to the kernel name in the notebook. Solid circle denotes that the kernel is busy.

When you start the notebook for the first time, the kernel performs some tasks in the background. Wait for the kernel to be ready.

2. Paste the following code in an empty cell, and then press **SHIFT + ENTER** to run the code. The command lists the Hive tables on the cluster:

```
%%sql
SHOW TABLES
```

When you use a Jupyter Notebook file with your HDInsight cluster, you get a preset `spark` session that you can use to run Hive queries using Spark SQL. `%%sql` tells Jupyter Notebook to use the preset `spark` session to run the Hive query. The query retrieves the top 10 rows from a Hive table (**hivesampletable**) that comes with all HDInsight clusters by default. The first time you submit the query, Jupyter will create a Spark application for the notebook. It takes about 30 seconds to complete. Once the Spark application is ready, the query is executed in about a second and produces the results. The output looks like:

Every time you run a query in Jupyter, your web browser window title shows a **(Busy)** status along with the notebook title. You also see a solid circle next to the **PySpark** text in the top-right corner.

3. Run another query to see the data in `hivesampletable`.

```
%%sql
SELECT * FROM hivesampletable LIMIT 10
```

The screen should refresh to show the query output.

	clientid	querytime	market	deviceplatform	devicemake	devicemodel	state	country	querydwelltime	sessionid	sessionpagevieworder
0	71448	2018-04-05 05:51:45	en-US	Android	Samsung	SCH-i500	California	United States	31.423273	1	41
1	71448	2018-04-05 05:51:33	en-US	Android	Samsung	SCH-i500	California	United States	11.878175	1	40
2	71448	2018-04-05 05:51:03	en-US	Android	Samsung	SCH-i500	California	United States	30.195784	1	39
3	71448	2018-04-05 05:51:03	en-US	Android	Samsung	SCH-i500	California	United States	0.129492	1	38
4	71448	2018-04-05 05:50:44	en-US	Android	Samsung	SCH-i500	California	United States	19.026435	1	37
5	71448	2018-04-05 05:50:27	en-US	Android	Samsung	SCH-i500	California	United States	16.411516	1	36
6	71448	2018-04-05 05:50:13	en-US	Android	Samsung	SCH-i500	California	United States	13.761518	1	35
7	71448	2018-04-05 05:50:04	en-US	Android	Samsung	SCH-i500	California	United States	9.091954	1	34
8	71448	2018-04-05 05:49:57	en-US	Android	Samsung	SCH-i500	California	United States	7.709461	1	33
9	71448	2018-04-05 05:54:21	en-US	Android	Samsung	SCH-i500	California	United States	0.899126	1	48

4. From the **File** menu on the notebook, select **Close and Halt**. Shutting down the notebook releases the cluster resources, including Spark application.

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you learned how to create an Apache Spark cluster in HDInsight and run a basic Spark SQL query. Advance to the next tutorial to learn how to use an HDInsight cluster to run interactive queries on sample data.

Run interactive queries on Apache Spark

Quickstart: Create an Azure Stream Analytics job using Bicep

9/21/2022 • 2 minutes to read • Edit Online

In this quickstart, you use Bicep to create an Azure Stream Analytics job. Once the job is created, you validate the deployment.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

To complete this article, you need to have an Azure subscription. Create one for free.

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Location for the resources.')
param location string = resourceGroup().location

@description('Stream Analytics Job Name, can contain alphanumeric characters and hypen and must be 3-63
characters long')
@minLength(3)
@maxLength(63)
param streamAnalyticsJobName string

@description('Number of Streaming Units')
@minValue(1)
@maxValue(48)
@allowed([
  1
  3
  6
  12
  18
  24
  30
  36
  42
  48
])
param numberOfStreamingUnits int

resource streamingJob 'Microsoft.StreamAnalytics/streamingjobs@2021-10-01-preview' = {
  name: streamAnalyticsJobName
  location: location
  properties: {
    sku: {
      name: 'Standard'
    }
    outputErrorPolicy: 'Stop'
    eventsOutOfOrderPolicy: 'Adjust'
    eventsOutOfOrderMaxDelayInSeconds: 0
    eventsLateArrivalMaxDelayInSeconds: 5
    dataLocale: 'en-US'
    transformation: {
      name: 'Transformation'
      properties: {
        streamingUnits: numberOfStreamingUnits
        query: 'SELECT\r\n    *\r\nINTO\r\n    [YourOutputAlias]\r\nFROM\r\n    [YourInputAlias]'
      }
    }
  }
}
```

The Azure resource defined in the Bicep file is Microsoft.StreamAnalytics/StreamingJobs: create an Azure Stream Analytics job.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.
   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
streamAnalyticsJobName =<job-name> numberOfStreamingUnits=<int>
```

You need to provide values for the following parameters:

- **streamAnalyticsJobName**: Replace **<job-name>** with the Stream Analytics job name. The name can contain alphanumeric characters and hyphens, and it must be at least 3-63 characters long.
- **numberOfStreamingUnits**: Replace **<int>** with the number of Streaming Units. Allowed values include: 1, 3, 6, 12, 18, 24, 30, 36, 42, and 48.

> **NOTE**
>
> When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

You can either use the Azure portal to check the Azure Stream Analytics job or use the following Azure CLI or Azure PowerShell script to list the resource.

Azure CLI

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

If you plan to continue on to subsequent tutorials, you may wish to leave these resources in place. When no longer needed, delete the resource group, which deletes the Azure Stream Analytics job. To delete the resource group by using Azure CLI or Azure PowerShell:

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you created an Azure Stream Analytics job using Bicep and validated the deployment. To learn how to create your own Bicep files using Visual Studio Code, continue on to the following article:

Quickstart: Create Bicep files with Visual Studio Code

Quickstart: Create an Azure Synapse Analytics dedicated SQL pool (formerly SQL DW) using Bicep

9/21/2022 • 2 minutes to read • Edit Online

This Bicep file will create a dedicated SQL pool (formerly SQL DW) with Transparent Data Encryption enabled. Dedicated SQL pool (formerly SQL DW) refers to the enterprise data warehousing features that are generally available in Azure Synapse.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('The SQL Logical Server name.')
param sqlServerName string = 'sql${uniqueString(resourceGroup().id)}'

@description('The administrator username of the SQL Server.')
param sqlAdministratorLogin string

@description('The administrator password of the SQL Server.')
@secure()
param sqlAdministratorPassword string

@description('The name of the Database.')
param databasesName string

@description('Enable/Disable Transparent Data Encryption')
@allowed([
  'Enabled'
  'Disabled'
])
param transparentDataEncryption string = 'Enabled'

@description('DW Performance Level expressed in DTU (i.e. 900 DTU = DW100c)')
@minValue(900)
@maxValue(54000)
param capacity int

@description('The SQL Database collation.')
param databaseCollation string = 'SQL_Latin1_General_CP1_CI_AS'

@description('Resource location')
param location string = resourceGroup().location

resource sqlServer 'Microsoft.Sql/servers@2021-11-01-preview' = {
  name: sqlServerName
  location: location
  properties: {
    administratorLogin: sqlAdministratorLogin
    administratorLoginPassword: sqlAdministratorPassword
    version: '12.0'
```

```
    version: '12.0'
    publicNetworkAccess: 'Enabled'
    restrictOutboundNetworkAccess: 'Disabled'
  }
}

resource sqlServerDatabase 'Microsoft.Sql/servers/databases@2021-11-01-preview' = {
  parent: sqlServer
  name: databasesName
  location: location
  sku: {
    name: 'DataWarehouse'
    tier: 'DataWarehouse'
    capacity: capacity
  }
  properties: {
    collation: databaseCollation
    catalogCollation: databaseCollation
    readScale: 'Disabled'
    requestedBackupStorageRedundancy: 'Geo'
    isLedgerOn: false
  }
}

resource encryption 'Microsoft.Sql/servers/databases/transparentDataEncryption@2021-11-01-preview' = {
  parent: sqlServerDatabase
  name: 'current'
  properties: {
    state: transparentDataEncryption
  }
}
```

The Bicep file defines one resource:

- Microsoft.Sql/servers

Deploy the Bicep file

1. Save the Bicep file as `main.bicep` to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
sqlAdministratorLogin=<admin-login> databasesName=<db-name> capacity=<int>
```

> **NOTE**
>
> Replace **<admin-login>** with the administrator login username for the SQL server. Replace **<db-name>** with the name of the database. Replace **<int>** with the DW performance level. The minimum value is 900 and the maximum value is 54000. You'll also be prompted to enter **sqlAdministratorPassword**.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you created a dedicated SQL pool (formerly SQL DW) using Bicep and validated the deployment. To learn more about Azure Synapse Analytics and Bicep, see the articles below.

- Read an Overview of Azure Synapse Analytics
- Learn more about Bicep
- Quickstart: Create Bicep files with Visual Studio Code

Quickstart: Create a Batch account by using a Bicep file

9/21/2022 • 2 minutes to read • Edit Online

Get started with Azure Batch by using a Bicep file to create a Batch account, including storage. You need a Batch account to create compute resources (pools of compute nodes) and Batch jobs. You can link an Azure Storage account with your Batch account, which is useful to deploy applications and store input and output data for most real-world workloads.

After completing this quickstart, you'll understand the key concepts of the Batch service and be ready to try Batch with more realistic workloads at larger scale.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

You must have an active Azure subscription.

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Batch Account Name')
param batchAccountName string = '${toLower(uniqueString(resourceGroup().id))}batch'

@description('Storage Account type')
@allowed([
  'Standard_LRS'
  'Standard_GRS'
  'Standard_ZRS'
  'Premium_LRS'
])
param storageAccountsku string = 'Standard_LRS'

@description('Location for all resources.')
param location string = resourceGroup().location

var storageAccountName = '${uniqueString(resourceGroup().id)}storage'

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-08-01' = {
  name: storageAccountName
  location: location
  sku: {
    name: storageAccountsku
  }
  kind: 'StorageV2'
  tags: {
    ObjectName: storageAccountName
  }
  properties: {}
}

resource batchAccount 'Microsoft.Batch/batchAccounts@2021-06-01' = {
  name: batchAccountName
  location: location
  tags: {
    ObjectName: batchAccountName
  }
  properties: {
    autoStorage: {
      storageAccountId: storageAccount.id
    }
  }
}

output storageAccountName string = storageAccountName
output batchAccountName string = batchAccountName
```

Two Azure resources are defined in the Bicep file:

- Microsoft.Storage/storageAccounts: Creates a storage account.
- Microsoft.Batch/batchAccounts: Creates a Batch account.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.
   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep
```

When the deployment finishes, you should see a message indicating the deployment succeeded.

Validate the deployment

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

If you plan to continue on with more of our tutorials, you may want to leave these resources in place. When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and all of its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you created a Batch account and a storage account using Bicep. To learn more about Azure Batch, continue to the Azure Batch tutorials.

Azure Batch tutorials

Quickstart: Create and deploy Azure Functions resources using Bicep

9/21/2022 • 3 minutes to read • Edit Online

In this article, you use Azure Functions with Bicep to create a function app and related resources in Azure. The function app provides an execution context for your function code executions.

Completing this quickstart incurs a small cost of a few USD cents or less in your Azure account.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

After you create the function app, you can deploy Azure Functions project code to that app.

Prerequisites

Azure account

Before you begin, you must have an Azure account with an active subscription. Create an account for free.

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('The name of the function app that you wish to create.')
param appName string = 'fnapp${uniqueString(resourceGroup().id)}'

@description('Storage Account type')
@allowed([
  'Standard_LRS'
  'Standard_GRS'
  'Standard_RAGRS'
])
param storageAccountType string = 'Standard_LRS'

@description('Location for all resources.')
param location string = resourceGroup().location

@description('Location for Application Insights')
param appInsightsLocation string

@description('The language worker runtime to load in the function app.')
@allowed([
  'node'
  'dotnet'
  'java'
])
param runtime string = 'node'

var functionAppName = appName
var hostingPlanName = appName
var applicationInsightsName = appName
var storageAccountName = '${uniqueString(resourceGroup().id)}azfunctions'
var functionWorkerRuntime = runtime

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-08-01' = {
  name: storageAccountName
```

```
  location: location
  sku: {
    name: storageAccountType
  }
  kind: 'Storage'
}

resource hostingPlan 'Microsoft.Web/serverfarms@2021-03-01' = {
  name: hostingPlanName
  location: location
  sku: {
    name: 'Y1'
    tier: 'Dynamic'
  }
  properties: {}
}

resource functionApp 'Microsoft.Web/sites@2021-03-01' = {
  name: functionAppName
  location: location
  kind: 'functionapp'
  identity: {
    type: 'SystemAssigned'
  }
  properties: {
    serverFarmId: hostingPlan.id
    siteConfig: {
      appSettings: [
        {
          name: 'AzureWebJobsStorage'
          value:
'DefaultEndpointsProtocol=https;AccountName=${storageAccountName};EndpointSuffix=${environment().suffixes.st
orage};AccountKey=${storageAccount.listKeys().keys[0].value}'
        }
        {
          name: 'WEBSITE_CONTENTAZUREFILECONNECTIONSTRING'
          value:
'DefaultEndpointsProtocol=https;AccountName=${storageAccountName};EndpointSuffix=${environment().suffixes.st
orage};AccountKey=${storageAccount.listKeys().keys[0].value}'
        }
        {
          name: 'WEBSITE_CONTENTSHARE'
          value: toLower(functionAppName)
        }
        {
          name: 'FUNCTIONS_EXTENSION_VERSION'
          value: '~2'
        }
        {
          name: 'WEBSITE_NODE_DEFAULT_VERSION'
          value: '~10'
        }
        {
          name: 'APPINSIGHTS_INSTRUMENTATIONKEY'
          value: applicationInsights.properties.InstrumentationKey
        }
        {
          name: 'FUNCTIONS_WORKER_RUNTIME'
          value: functionWorkerRuntime
        }
      ]
      ftpsState: 'FtpsOnly'
      minTlsVersion: '1.2'
    }
    httpsOnly: true
  }
}

resource applicationInsights 'Microsoft.Insights/components@2020-02-02' = {
```

```
  name: applicationInsightsName
  location: appInsightsLocation
  kind: 'web'
  properties: {
    Application_Type: 'web'
    Request_Source: 'rest'
  }
}
```

The following four Azure resources are created by this Bicep file:

- **Microsoft.Storage/storageAccounts**: create an Azure Storage account, which is required by Functions.
- **Microsoft.Web/serverfarms**: create a serverless Consumption hosting plan for the function app.
- **Microsoft.Web/sites**: create a function app.
- **microsoft.insights/components**: create an Application Insights instance for monitoring.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

   ```
   az group create --name exampleRG --location eastus
   az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
   appInsightsLocation=<app-location>
   ```

   > **NOTE**
   > Replace **<app-location>** with the region for Application Insights, which is usually the same as the resource group.

   When the deployment finishes, you should see a message indicating the deployment succeeded.

Validate the deployment

Use Azure CLI or Azure PowerShell to validate the deployment.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Visit function app welcome page

1. Use the output from the previous validation step to retrieve the unique name created for your function app.
2. Open a browser and enter the following URL: **<https://<appName.azurewebsites.net>**. Make sure to replace **<\appName>** with the unique name created for your function app.

When you visit the URL, you should see a page like this:

Clean up resources

If you continue to the next step and add an Azure Storage queue output binding, keep all your resources in place as you'll build on what you've already done.

Otherwise, if you no longer need the resources, use Azure CLI, PowerShell, or Azure portal to delete the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

Now that you've created your function app resources in Azure, you can deploy your code to the existing app by using one of the following tools:

- Visual Studio Code
- Visual Studio
- Azure Functions Core Tools

Quickstart: Create an Ubuntu Linux virtual machine using a Bicep file

9/21/2022 • 3 minutes to read • Edit Online

Applies to: ✔ Linux VMs

This quickstart shows you how to use a Bicep file to deploy an Ubuntu Linux virtual machine (VM) in Azure.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('The name of you Virtual Machine.')
param vmName string = 'simpleLinuxVM'

@description('Username for the Virtual Machine.')
param adminUsername string

@description('Type of authentication to use on the Virtual Machine. SSH key is recommended.')
@allowed([
  'sshPublicKey'
  'password'
])
param authenticationType string = 'password'

@description('SSH Key or password for the Virtual Machine. SSH key is recommended.')
@secure()
param adminPasswordOrKey string

@description('Unique DNS Name for the Public IP used to access the Virtual Machine.')
param dnsLabelPrefix string = toLower('${vmName}-${uniqueString(resourceGroup().id)}')

@description('The Ubuntu version for the VM. This will pick a fully patched image of this given Ubuntu
version.')
@allowed([
  '12.04.5-LTS'
  '14.04.5-LTS'
  '16.04.0-LTS'
  '18.04-LTS'
])
param ubuntuOSVersion string = '18.04-LTS'

@description('Location for all resources.')
param location string = resourceGroup().location

@description('The size of the VM')
param vmSize string = 'Standard_B2s'

@description('Name of the VNET')
param virtualNetworkName string = 'vNet'
```

```
@description('Name of the subnet in the virtual network')
param subnetName string = 'Subnet'

@description('Name of the Network Security Group')
param networkSecurityGroupName string = 'SecGroupNet'

var publicIPAddressName = '${vmName}PublicIP'
var networkInterfaceName = '${vmName}NetInt'
var osDiskType = 'Standard_LRS'
var subnetAddressPrefix = '10.1.0.0/24'
var addressPrefix = '10.1.0.0/16'
var linuxConfiguration = {
  disablePasswordAuthentication: true
  ssh: {
    publicKeys: [
      {
        path: '/home/${adminUsername}/.ssh/authorized_keys'
        keyData: adminPasswordOrKey
      }
    ]
  }
}

resource nic 'Microsoft.Network/networkInterfaces@2021-05-01' = {
  name: networkInterfaceName
  location: location
  properties: {
    ipConfigurations: [
      {
        name: 'ipconfig1'
        properties: {
          subnet: {
            id: subnet.id
          }
          privateIPAllocationMethod: 'Dynamic'
          publicIPAddress: {
            id: publicIP.id
          }
        }
      }
    ]
    networkSecurityGroup: {
      id: nsg.id
    }
  }
}

resource nsg 'Microsoft.Network/networkSecurityGroups@2021-05-01' = {
  name: networkSecurityGroupName
  location: location
  properties: {
    securityRules: [
      {
        name: 'SSH'
        properties: {
          priority: 1000
          protocol: 'Tcp'
          access: 'Allow'
          direction: 'Inbound'
          sourceAddressPrefix: '*'
          sourcePortRange: '*'
          destinationAddressPrefix: '*'
          destinationPortRange: '22'
        }
      }
    ]
  }
}
```

```
resource vnet 'Microsoft.Network/virtualNetworks@2021-05-01' = {
  name: virtualNetworkName
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        addressPrefix
      ]
    }
  }
}

resource subnet 'Microsoft.Network/virtualNetworks/subnets@2021-05-01' = {
  parent: vnet
  name: subnetName
  properties: {
    addressPrefix: subnetAddressPrefix
    privateEndpointNetworkPolicies: 'Enabled'
    privateLinkServiceNetworkPolicies: 'Enabled'
  }
}

resource publicIP 'Microsoft.Network/publicIPAddresses@2021-05-01' = {
  name: publicIPAddressName
  location: location
  sku: {
    name: 'Basic'
  }
  properties: {
    publicIPAllocationMethod: 'Dynamic'
    publicIPAddressVersion: 'IPv4'
    dnsSettings: {
      domainNameLabel: dnsLabelPrefix
    }
    idleTimeoutInMinutes: 4
  }
}

resource vm 'Microsoft.Compute/virtualMachines@2021-11-01' = {
  name: vmName
  location: location
  properties: {
    hardwareProfile: {
      vmSize: vmSize
    }
    storageProfile: {
      osDisk: {
        createOption: 'FromImage'
        managedDisk: {
          storageAccountType: osDiskType
        }
      }
      imageReference: {
        publisher: 'Canonical'
        offer: 'UbuntuServer'
        sku: ubuntuOSVersion
        version: 'latest'
      }
    }
    networkProfile: {
      networkInterfaces: [
        {
          id: nic.id
        }
      ]
    }
    osProfile: {
      computerName: vmName
```

```
      adminUsername: adminUsername
      adminPassword: adminPasswordOrKey
      linuxConfiguration: ((authenticationType == 'password') ? null : linuxConfiguration)
    }
  }
}

output adminUsername string = adminUsername
output hostname string = publicIP.properties.dnsSettings.fqdn
output sshCommand string = 'ssh ${adminUsername}@${publicIP.properties.dnsSettings.fqdn}'
```

Several resources are defined in the Bicep file:

- **Microsoft.Network/virtualNetworks/subnets**: create a subnet.
- **Microsoft.Storage/storageAccounts**: create a storage account.
- **Microsoft.Network/networkInterfaces**: create a NIC.
- **Microsoft.Network/networkSecurityGroups**: create a network security group.
- **Microsoft.Network/virtualNetworks**: create a virtual network.
- **Microsoft.Network/publicIPAddresses**: create a public IP address.
- **Microsoft.Compute/virtualMachines**: create a virtual machine.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

   ```
   az group create --name exampleRG --location eastus

   az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
   adminUsername=<admin-username>
   ```

   > **NOTE**
   >
   > Replace **<admin-username>** with a unique username. You'll also be prompted to enter adminPasswordOrKey.

   When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the VM and all of the

resources in the resource group.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you deployed a simple virtual machine using a Bicep file. To learn more about Azure virtual machines, continue to the tutorial for Linux VMs.

Azure Linux virtual machine tutorials

Quickstart: Create a Windows virtual machine scale set with Bicep

9/21/2022 • 8 minutes to read • Edit Online

> **NOTE**
>
> This quickstart uses Uniform Orchestration mode. We recommend using Flexible Orchestration for new workloads. For more information, see Orchesration modes for virtual machine scale sets in Azure.

A virtual machine scale set allows you to deploy and manage a set of auto-scaling virtual machines. You can scale the number of VMs in the virtual machine scale set manually, or define rules to autoscale based on resource usage like CPU, memory demand, or network traffic. An Azure load balancer then distributes traffic to the VM instances in the virtual machine scale set. In this quickstart, you create a virtual machine scale set and deploy a sample application with Bicep.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('String used as a base for naming resources. Must be 3-61 characters in length and globally
unique across Azure. A hash is prepended to this string for some resources, and resource-specific
information is appended.')
@minLength(3)
@maxLength(61)
param vmssName string

@description('Size of VMs in the VM Scale Set.')
param vmSku string = 'Standard_A1_v2'

@description('The Windows version for the VM. This will pick a fully patched image of this given Windows
version. Allowed values: 2008-R2-SP1, 2012-Datacenter, 2012-R2-Datacenter & 2016-Datacenter, 2019-
Datacenter.')
@allowed([
  '2008-R2-SP1'
  '2012-Datacenter'
  '2012-R2-Datacenter'
  '2016-Datacenter'
  '2019-Datacenter'
])
param windowsOSVersion string = '2019-Datacenter'

@description('Number of VM instances (100 or less).')
@minValue(1)
@maxValue(100)
param instanceCount int = 3

@description('When true this limits the scale set to a single placement group, of max size 100 virtual
```

```
@description('When true this limits the scale set to a single placement group, of max size 100 virtual
machines. NOTE: If singlePlacementGroup is true, it may be modified to false. However, if
singlePlacementGroup is false, it may not be modified to true.')
param singlePlacementGroup bool = true

@description('Admin username on all VMs.')
param adminUsername string = 'vmssadmin'

@description('Admin password on all VMs.')
@secure()
param adminPassword string

@description('The base URI where artifacts required by this template are located. For example, if stored on
a public GitHub repo, you\'d use the following URI: https://raw.githubusercontent.com/Azure/azure-
quickstart-templates/master/201-vmss-windows-webapp-dsc-autoscale/.')
param _artifactsLocation string = deployment().properties.templateLink.uri

@description('The sasToken required to access _artifactsLocation.  If your artifacts are stored on a public
repo or public storage account you can leave this blank.')
@secure()
param _artifactsLocationSasToken string = ''

@description('Location of the PowerShell DSC zip file relative to the URI specified in the
_artifactsLocation, i.e. DSC/IISInstall.ps1.zip')
param powershelldscZip string = 'DSC/InstallIIS.zip'

@description('Location of the  of the WebDeploy package zip file relative to the URI specified in
_artifactsLocation, i.e. WebDeploy/DefaultASPWebApp.v1.0.zip')
param webDeployPackage string = 'WebDeploy/DefaultASPWebApp.v1.0.zip'

@description('Version number of the DSC deployment. Changing this value on subsequent deployments will
trigger the extension to run.')
param powershelldscUpdateTagVersion string = '1.0'

@description('Location for all resources.')
param location string = resourceGroup().location

@description('Fault Domain count for each placement group.')
param platformFaultDomainCount int = 1

var vmScaleSetName = toLower(substring('vmssName${uniqueString(resourceGroup().id)}', 0, 9))
var longvmScaleSet = toLower(vmssName)
var addressPrefix = '10.0.0.0/16'
var subnetPrefix = '10.0.0.0/24'
var vNetName = '${vmScaleSetName}vnet'
var publicIPAddressName = '${vmScaleSetName}pip'
var subnetName = '${vmScaleSetName}subnet'
var loadBalancerName = '${vmScaleSetName}lb'
var publicIPAddressID = publicIPAddress.id
var lbProbeID = resourceId('Microsoft.Network/loadBalancers/probes', loadBalancerName, 'tcpProbe')
var natPoolName = '${vmScaleSetName}natpool'
var bePoolName = '${vmScaleSetName}bepool'
var lbPoolID = resourceId('Microsoft.Network/loadBalancers/backendAddressPools', loadBalancerName,
bePoolName)
var natStartPort = 50000
var natEndPort = 50119
var natBackendPort = 3389
var nicName = '${vmScaleSetName}nic'
var ipConfigName = '${vmScaleSetName}ipconfig'
var frontEndIPConfigID = resourceId('Microsoft.Network/loadBalancers/frontendIPConfigurations',
loadBalancerName, 'loadBalancerFrontEnd')
var osType = {
  publisher: 'MicrosoftWindowsServer'
  offer: 'WindowsServer'
  sku: windowsOSVersion
  version: 'latest'
}
var imageReference = osType
var webDeployPackageFullPath = uri(_artifactsLocation, '${webDeployPackage}${_artifactsLocationSasToken}')
var powershelldscZipFullPath = uri(_artifactsLocation, '${powershelldscZip}${_artifactsLocationSasToken}')
```

```
var powershellscZipFullPath = uri(_artifactsLocation, '${powershellscZip}${_artifactsLocationSasToken}')

resource loadBalancer 'Microsoft.Network/loadBalancers@2021-05-01' = {
  name: loadBalancerName
  location: location
  properties: {
    frontendIPConfigurations: [
      {
        name: 'LoadBalancerFrontEnd'
        properties: {
          publicIPAddress: {
            id: publicIPAddressID
          }
        }
      }
    ]
    backendAddressPools: [
      {
        name: bePoolName
      }
    ]
    inboundNatPools: [
      {
        name: natPoolName
        properties: {
          frontendIPConfiguration: {
            id: frontEndIPConfigID
          }
          protocol: 'Tcp'
          frontendPortRangeStart: natStartPort
          frontendPortRangeEnd: natEndPort
          backendPort: natBackendPort
        }
      }
    ]
    loadBalancingRules: [
      {
        name: 'LBRule'
        properties: {
          frontendIPConfiguration: {
            id: frontEndIPConfigID
          }
          backendAddressPool: {
            id: lbPoolID
          }
          protocol: 'Tcp'
          frontendPort: 80
          backendPort: 80
          enableFloatingIP: false
          idleTimeoutInMinutes: 5
          probe: {
            id: lbProbeID
          }
        }
      }
    ]
    probes: [
      {
        name: 'tcpProbe'
        properties: {
          protocol: 'Tcp'
          port: 80
          intervalInSeconds: 5
          numberOfProbes: 2
        }
      }
    ]
  }
}
```

```
resource vmScaleSet 'Microsoft.Compute/virtualMachineScaleSets@2021-11-01' = {
  name: vmScaleSetName
  location: location
  sku: {
    name: vmSku
    tier: 'Standard'
    capacity: instanceCount
  }
  properties: {
    overprovision: true
    upgradePolicy: {
      mode: 'Automatic'
    }
    singlePlacementGroup: singlePlacementGroup
    platformFaultDomainCount: platformFaultDomainCount
    virtualMachineProfile: {
      storageProfile: {
        osDisk: {
          caching: 'ReadWrite'
          createOption: 'FromImage'
        }
        imageReference: imageReference
      }
      osProfile: {
        computerNamePrefix: vmScaleSetName
        adminUsername: adminUsername
        adminPassword: adminPassword
      }
      networkProfile: {
        networkInterfaceConfigurations: [
          {
            name: nicName
            properties: {
              primary: true
              ipConfigurations: [
                {
                  name: ipConfigName
                  properties: {
                    subnet: {
                      id: vNet.properties.subnets[0].id
                    }
                    loadBalancerBackendAddressPools: [
                      {
                        id: lbPoolID
                      }
                    ]
                  }
                }
              ]
            }
          }
        ]
      }
      extensionProfile: {
        extensions: [
          {
            name: 'Microsoft.Powershell.DSC'
            properties: {
              publisher: 'Microsoft.Powershell'
              type: 'DSC'
              typeHandlerVersion: '2.9'
              autoUpgradeMinorVersion: true
              forceUpdateTag: powershelldscUpdateTagVersion
              settings: {
                configuration: {
                  url: powershelldscZipFullPath
                  script: 'InstallIIS.ps1'
                  function: 'InstallIIS'
```

```
                }
                configurationArguments: {
                  nodeName: 'localhost'
                  WebDeployPackagePath: webDeployPackageFullPath
                }
              }
            }
          }
        ]
      }
    }
  }
}

resource publicIPAddress 'Microsoft.Network/publicIPAddresses@2021-05-01' = {
  name: publicIPAddressName
  location: location
  properties: {
    publicIPAllocationMethod: 'Static'
    dnsSettings: {
      domainNameLabel: longvmScaleSet
    }
  }
}

resource vNet 'Microsoft.Network/virtualNetworks@2021-05-01' = {
  name: vNetName
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        addressPrefix
      ]
    }
    subnets: [
      {
        name: subnetName
        properties: {
          addressPrefix: subnetPrefix
        }
      }
    ]
  }
}

resource autoscalehost 'Microsoft.Insights/autoscalesettings@2021-05-01-preview' = {
  name: 'autoscalehost'
  location: location
  properties: {
    name: 'autoscalehost'
    targetResourceUri: vmScaleSet.id
    enabled: true
    profiles: [
      {
        name: 'Profile1'
        capacity: {
          minimum: '1'
          maximum: '10'
          default: '1'
        }
        rules: [
          {
            metricTrigger: {
              metricName: 'Percentage CPU'
              metricResourceUri: vmScaleSet.id
              timeGrain: 'PT1M'
              statistic: 'Average'
              timeWindow: 'PT5M'
              timeAggregation: 'Average'
```

```
              operator: 'GreaterThan'
              threshold: 50
            }
            scaleAction: {
              direction: 'Increase'
              type: 'ChangeCount'
              value: '1'
              cooldown: 'PT5M'
            }
          }
          {
            metricTrigger: {
              metricName: 'Percentage CPU'
              metricResourceUri: vmScaleSet.id
              timeGrain: 'PT1M'
              statistic: 'Average'
              timeWindow: 'PT5M'
              timeAggregation: 'Average'
              operator: 'LessThan'
              threshold: 30
            }
            scaleAction: {
              direction: 'Decrease'
              type: 'ChangeCount'
              value: '1'
              cooldown: 'PT5M'
            }
          }
        ]
      }
    ]
  }
}

output applicationUrl string = uri('http://${publicIPAddress.properties.dnsSettings.fqdn}', '/MyApp')
```

The following resources are defined in the Bicep file:

- Microsoft.Network/virtualNetworks
- Microsoft.Network/publicIPAddresses
- Microsoft.Network/loadBalancers
- Microsoft.Compute/virtualMachineScaleSets
- Microsoft.Insights/autoscaleSettings

Define a scale set

To create a virtual machine scale set with a Bicep file, you define the appropriate resources. The core parts of the virtual machine scale set resource type are:

PROPERTY	DESCRIPTION OF PROPERTY	EXAMPLE TEMPLATE VALUE
type	Azure resource type to create	Microsoft.Compute/virtualMachineScaleSets
name	The scale set name	myScaleSet
location	The location to create the scale set	East US
sku.name	The VM size for each scale set instance	Standard_A1

PROPERTY	DESCRIPTION OF PROPERTY	EXAMPLE TEMPLATE VALUE
sku.capacity	The number of VM instances to initially create	2
upgradePolicy.mode	VM instance upgrade mode when changes occur	Automatic
imageReference	The platform or custom image to use for the VM instances	Microsoft Windows Server 2016 Datacenter
osProfile.computerNamePrefix	The name prefix for each VM instance	myvmss
osProfile.adminUsername	The username for each VM instance	azureuser
osProfile.adminPassword	The password for each VM instance	P@ssw0rd!

To customize a virtual machine scale set Bicep file, you can change the VM size or initial capacity. Another option is to use a different platform or a custom image.

Add a sample application

To test your virtual machine scale set, install a basic web application. When you deploy a virtual machine scale set, VM extensions can provide post-deployment configuration and automation tasks, such as installing an app. Scripts can be downloaded from GitHub or provided to the Azure portal at extension run-time. To apply an extension to your virtual machine scale set, add the `extensionProfile` section to the resource example above. The extension profile typically defines the following properties:

- Extension type
- Extension publisher
- Extension version
- Location of configuration or install scripts
- Commands to execute on the VM instances

The Bicep file uses the PowerShell DSC extension to install an ASP.NET MVC app that runs in IIS.

An install script is downloaded from GitHub, as defined in `url`. The extension then runs `InstallIIS` from the `IISInstall.ps1` script, as defined in `function` and `Script`. The ASP.NET app itself is provided as a Web Deploy package, which is also downloaded from GitHub, as defined in `WebDeployPackagePath`:

Deploy the Bicep file

1. Save the Bicep file as `main.bicep` to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
vmssName=<vmss-name>
```

Replace *<vmss-name>* with the name of the virtual machine scale set. It must be 3-61 characters in length and globally unique across Azure. You'll be prompted to enter `adminPassword`.

> **NOTE**
> When the deployment finishes, you should see a message indicating the deployment succeeded. It can take 10-15 minutes for the virtual machine scale set to be created and apply the extension to configure the app.

Validate the deployment

To see your virtual machine scale set in action, access the sample web application in a web browser. Obtain the public IP address of your load balancer using Azure CLI or Azure PowerShell.

- CLI
- PowerShell

```
az network public-ip show --resource-group exampleRG
```

Enter the public IP address of the load balancer in to a web browser in the format *http://publicIpAddress/MyApp*. The load balancer distributes traffic to one of your VM instances, as shown in the following example:

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to remove the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you created a Windows virtual machine scale set with a Bicep file and used the PowerShell DSC extension to install a basic ASP.NET app on the VM instances. To learn more, continue to the tutorial for how to create and manage Azure virtual machine scale sets.

Create and manage Azure virtual machine scale sets

Quickstart: Create a Windows virtual machine using a Bicep file

9/21/2022 • 3 minutes to read • Edit Online

Applies to: ✔ Windows VMs

This quickstart shows you how to use a Bicep file to deploy a Windows virtual machine (VM) in Azure.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Username for the Virtual Machine.')
param adminUsername string

@description('Password for the Virtual Machine.')
@minLength(12)
@secure()
param adminPassword string

@description('Unique DNS Name for the Public IP used to access the Virtual Machine.')
param dnsLabelPrefix string = toLower('${vmName}-${uniqueString(resourceGroup().id, vmName)}')

@description('Name for the Public IP used to access the Virtual Machine.')
param publicIpName string = 'myPublicIP'

@description('Allocation method for the Public IP used to access the Virtual Machine.')
@allowed([
  'Dynamic'
  'Static'
])
param publicIPAllocationMethod string = 'Dynamic'

@description('SKU for the Public IP used to access the Virtual Machine.')
@allowed([
  'Basic'
  'Standard'
])
param publicIpSku string = 'Basic'

@description('The Windows version for the VM. This will pick a fully patched image of this given Windows
version.')
@allowed([
'2008-R2-SP1'
'2008-R2-SP1-smalldisk'
'2012-Datacenter'
'2012-datacenter-gensecond'
'2012-Datacenter-smalldisk'
'2012-datacenter-smalldisk-g2'
'2012-Datacenter-zhcn'
```

```
  '2012-datacenter-zhcn-g2'
  '2012-R2-Datacenter'
  '2012-r2-datacenter-gensecond'
  '2012-R2-Datacenter-smalldisk'
  '2012-r2-datacenter-smalldisk-g2'
  '2012-R2-Datacenter-zhcn'
  '2012-r2-datacenter-zhcn-g2'
  '2016-Datacenter'
  '2016-datacenter-gensecond'
  '2016-datacenter-gs'
  '2016-Datacenter-Server-Core'
  '2016-datacenter-server-core-g2'
  '2016-Datacenter-Server-Core-smalldisk'
  '2016-datacenter-server-core-smalldisk-g2'
  '2016-Datacenter-smalldisk'
  '2016-datacenter-smalldisk-g2'
  '2016-Datacenter-with-Containers'
  '2016-datacenter-with-containers-g2'
  '2016-datacenter-with-containers-gs'
  '2016-Datacenter-zhcn'
  '2016-datacenter-zhcn-g2'
  '2019-Datacenter'
  '2019-Datacenter-Core'
  '2019-datacenter-core-g2'
  '2019-Datacenter-Core-smalldisk'
  '2019-datacenter-core-smalldisk-g2'
  '2019-Datacenter-Core-with-Containers'
  '2019-datacenter-core-with-containers-g2'
  '2019-Datacenter-Core-with-Containers-smalldisk'
  '2019-datacenter-core-with-containers-smalldisk-g2'
  '2019-datacenter-gensecond'
  '2019-datacenter-gs'
  '2019-Datacenter-smalldisk'
  '2019-datacenter-smalldisk-g2'
  '2019-Datacenter-with-Containers'
  '2019-datacenter-with-containers-g2'
  '2019-datacenter-with-containers-gs'
  '2019-Datacenter-with-Containers-smalldisk'
  '2019-datacenter-with-containers-smalldisk-g2'
  '2019-Datacenter-zhcn'
  '2019-datacenter-zhcn-g2'
  '2022-datacenter'
  '2022-datacenter-azure-edition'
  '2022-datacenter-azure-edition-core'
  '2022-datacenter-azure-edition-core-smalldisk'
  '2022-datacenter-azure-edition-smalldisk'
  '2022-datacenter-core'
  '2022-datacenter-core-g2'
  '2022-datacenter-core-smalldisk'
  '2022-datacenter-core-smalldisk-g2'
  '2022-datacenter-g2'
  '2022-datacenter-smalldisk'
  '2022-datacenter-smalldisk-g2'
  ])
  param OSVersion string = '2022-datacenter-azure-edition-core'

  @description('Size of the virtual machine.')
  param vmSize string = 'Standard_D2s_v5'

  @description('Location for all resources.')
  param location string = resourceGroup().location

  @description('Name of the virtual machine.')
  param vmName string = 'simple-vm'

  var storageAccountName = 'bootdiags${uniqueString(resourceGroup().id)}'
  var nicName = 'myVMNic'
  var addressPrefix = '10.0.0.0/16'
  var subnetName = 'Subnet'
```

```
var subnetPrefix = '10.0.0.0/24'
var virtualNetworkName = 'MyVNET'
var networkSecurityGroupName = 'default-NSG'

resource stg 'Microsoft.Storage/storageAccounts@2021-04-01' = {
  name: storageAccountName
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'Storage'
}

resource pip 'Microsoft.Network/publicIPAddresses@2021-02-01' = {
  name: publicIpName
  location: location
  sku: {
    name: publicIpSku
  }
  properties: {
    publicIPAllocationMethod: publicIPAllocationMethod
    dnsSettings: {
      domainNameLabel: dnsLabelPrefix
    }
  }
}

resource securityGroup 'Microsoft.Network/networkSecurityGroups@2021-02-01' = {
  name: networkSecurityGroupName
  location: location
  properties: {
    securityRules: [
      {
        name: 'default-allow-3389'
        properties: {
          priority: 1000
          access: 'Allow'
          direction: 'Inbound'
          destinationPortRange: '3389'
          protocol: 'Tcp'
          sourcePortRange: '*'
          sourceAddressPrefix: '*'
          destinationAddressPrefix: '*'
        }
      }
    ]
  }
}

resource vn 'Microsoft.Network/virtualNetworks@2021-02-01' = {
  name: virtualNetworkName
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        addressPrefix
      ]
    }
    subnets: [
      {
        name: subnetName
        properties: {
          addressPrefix: subnetPrefix
          networkSecurityGroup: {
            id: securityGroup.id
          }
        }
      }
    ]
```

```
    }
  }

  resource nic 'Microsoft.Network/networkInterfaces@2021-02-01' = {
    name: nicName
    location: location
    properties: {
      ipConfigurations: [
        {
          name: 'ipconfig1'
          properties: {
            privateIPAllocationMethod: 'Dynamic'
            publicIPAddress: {
              id: pip.id
            }
            subnet: {
              id: resourceId('Microsoft.Network/virtualNetworks/subnets', vn.name, subnetName)
            }
          }
        }
      ]
    }
  }

  resource vm 'Microsoft.Compute/virtualMachines@2021-03-01' = {
    name: vmName
    location: location
    properties: {
      hardwareProfile: {
        vmSize: vmSize
      }
      osProfile: {
        computerName: vmName
        adminUsername: adminUsername
        adminPassword: adminPassword
      }
      storageProfile: {
        imageReference: {
          publisher: 'MicrosoftWindowsServer'
          offer: 'WindowsServer'
          sku: OSVersion
          version: 'latest'
        }
        osDisk: {
          createOption: 'FromImage'
          managedDisk: {
            storageAccountType: 'StandardSSD_LRS'
          }
        }
        dataDisks: [
          {
            diskSizeGB: 1023
            lun: 0
            createOption: 'Empty'
          }
        ]
      }
      networkProfile: {
        networkInterfaces: [
          {
            id: nic.id
          }
        ]
      }
      diagnosticsProfile: {
        bootDiagnostics: {
          enabled: true
          storageUri: stg.properties.primaryEndpoints.blob
        }
```

```
      }
    }
  }
}

output hostname string = pip.properties.dnsSettings.fqdn
```

Several resources are defined in the Bicep file:

- **Microsoft.Network/virtualNetworks/subnets**: create a subnet.
- **Microsoft.Storage/storageAccounts**: create a storage account.
- **Microsoft.Network/publicIPAddresses**: create a public IP address.
- **Microsoft.Network/networkSecurityGroups**: create a network security group.
- **Microsoft.Network/virtualNetworks**: create a virtual network.
- **Microsoft.Network/networkInterfaces**: create a NIC.
- **Microsoft.Compute/virtualMachines**: create a virtual machine.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
adminUsername=<admin-username>
```

> **NOTE**
> Replace **<admin-username>** with a unique username. You'll also be prompted to enter adminPassword. The minimum password length is 12 characters.

When the deployment finishes, you should see a messaged indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the VM and all of the resources in the resource group.

- CLI

- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you deployed a simple virtual machine using a Bicep file. To learn more about Azure virtual machines, continue to the tutorial for Linux VMs.

Azure Windows virtual machine tutorials

Quickstart: Deploy an Azure Kubernetes Service (AKS) cluster using Bicep

9/21/2022 • 8 minutes to read • Edit Online

Azure Kubernetes Service (AKS) is a managed Kubernetes service that lets you quickly deploy and manage clusters. In this quickstart, you'll:

- Deploy an AKS cluster using a Bicep file.
- Run a sample multi-container application with a web front-end and a Redis instance in the cluster.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

This quickstart assumes a basic understanding of Kubernetes concepts. For more information, see Kubernetes core concepts for Azure Kubernetes Service (AKS).

Prerequisites

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

- Azure CLI
- Azure PowerShell

You can use either the Azure Cloud Shell or a local Azure CLI.

- Azure Cloud Shell with the Bash environment. Or launch the Cloud Shell here.

  Launch Cloud Shell

- Local Azure CLI, see how to install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.

- Sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
- When you first use Azure CLI, install the Azure CLI extension. For more information about extensions, see Use extensions with the Azure CLI.
- Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

- This article requires version 2.20.0 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.
- To create an AKS cluster using a Bicep file, you provide an SSH public key. If you need this resource, see the following section; otherwise skip to the Review the Bicep file section.
- The identity you're using to create your cluster has the appropriate minimum permissions. For more details on access and identity for AKS, see Access and identity options for Azure Kubernetes Service (AKS).
- To deploy a Bicep file, you need write access on the resources you're deploying and access to all operations on the Microsoft.Resources/deployments resource type. For example, to deploy a virtual machine, you need Microsoft.Compute/virtualMachines/write and Microsoft.Resources/deployments/* permissions. For a list of roles and permissions, see Azure built-in roles.

Create an SSH key pair

To access AKS nodes, you connect using an SSH key pair (public and private), which you generate using the `ssh-keygen` command. By default, these files are created in the *~/.ssh* directory. Running the `ssh-keygen` command will overwrite any SSH key pair with the same name already existing in the given location.

1. Go to https://shell.azure.com to open Cloud Shell in your browser.
2. Run the `ssh-keygen` command. The following example creates an SSH key pair using RSA encryption and a bit length of 4096:

```
ssh-keygen -t rsa -b 4096
```

For more information about creating SSH keys, see Create and manage SSH keys for authentication in Azure.

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('The name of the Managed Cluster resource.')
param clusterName string = 'aks101cluster'

@description('The location of the Managed Cluster resource.')
param location string = resourceGroup().location

@description('Optional DNS prefix to use with hosted Kubernetes API server FQDN.')
param dnsPrefix string

@description('Disk size (in GB) to provision for each of the agent pool nodes. This value ranges from 0 to
1023. Specifying 0 will apply the default disk size for that agentVMSize.')
@minValue(0)
@maxValue(1023)
param osDiskSizeGB int = 0

@description('The number of nodes for the cluster.')
@minValue(1)
@maxValue(50)
param agentCount int = 3

@description('The size of the Virtual Machine.')
param agentVMSize string = 'standard_d2s_v3'

@description('User name for the Linux Virtual Machines.')
param linuxAdminUsername string

@description('Configure all linux machines with the SSH RSA public key string. Your key should include three
parts, for example \'ssh-rsa AAAAB...snip...UcyupgH azureuser@linuxvm\'')
param sshRSAPublicKey string

resource aks 'Microsoft.ContainerService/managedClusters@2022-05-02-preview' = {
  name: clusterName
  location: location
  identity: {
    type: 'SystemAssigned'
  }
  properties: {
    dnsPrefix: dnsPrefix
    agentPoolProfiles: [
      {
        name: 'agentpool'
        osDiskSizeGB: osDiskSizeGB
        count: agentCount
        vmSize: agentVMSize
        osType: 'Linux'
        mode: 'System'
      }
    ]
    linuxProfile: {
      adminUsername: linuxAdminUsername
      ssh: {
        publicKeys: [
          {
            keyData: sshRSAPublicKey
          }
        ]
      }
    }
  }
}

output controlPlaneFQDN string = aks.properties.fqdn
```

The resource defined in the Bicep file:

- Microsoft.ContainerService/managedClusters

For more AKS samples, see the AKS quickstart templates site.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

   ```
   az group create --name myResourceGroup --location eastus
   az deployment group create --resource-group myResourceGroup --template-file main.bicep --parameters
   clusterName=<cluster-name> dnsPrefix=<dns-previs> linuxAdminUsername=<linux-admin-username>
   sshRSAPublicKey='<ssh-key>'
   ```

   Provide the following values in the commands:

   - **Cluster name**: Enter a unique name for the AKS cluster, such as *myAKSCluster*.
   - **DNS prefix**: Enter a unique DNS prefix for your cluster, such as *myakscluster*.
   - **Linux Admin Username**: Enter a username to connect using SSH, such as *azureuser*.
   - **SSH RSA Public Key**: Copy and paste the *public* part of your SSH key pair (by default, the contents of *~/.ssh/id_rsa.pub*).

   It takes a few minutes to create the AKS cluster. Wait for the cluster to be successfully deployed before you move on to the next step.

Validate the Bicep deployment

Connect to the cluster

To manage a Kubernetes cluster, use the Kubernetes command-line client, kubectl. `kubectl` is already installed if you use Azure Cloud Shell.

- Azure CLI
- Azure PowerShell

1. Install `kubectl` locally using the az aks install-cli command:

   ```
   az aks install-cli
   ```

2. Configure `kubectl` to connect to your Kubernetes cluster using the az aks get-credentials command. This command downloads credentials and configures the Kubernetes CLI to use them.

   ```
   az aks get-credentials --resource-group myResourceGroup --name myAKSCluster
   ```

3. Verify the connection to your cluster using the kubectl get command. This command returns a list of the cluster nodes.

   ```
   kubectl get nodes
   ```

   The following output example shows the three nodes created in the previous steps. Make sure the node status is *Ready*:

```
NAME                        STATUS   ROLES   AGE     VERSION
aks-agentpool-41324942-0    Ready    agent   6m44s   v1.12.6
aks-agentpool-41324942-1    Ready    agent   6m46s   v1.12.6
aks-agentpool-41324942-2    Ready    agent   6m45s   v1.12.6
```

Deploy the application

A Kubernetes manifest file defines a cluster's desired state, such as which container images to run.

In this quickstart, you'll use a manifest to create all objects needed to run the Azure Vote application. This manifest includes two Kubernetes deployments:

- The sample Azure Vote Python applications.
- A Redis instance.

Two Kubernetes Services are also created:

- An internal service for the Redis instance.
- An external service to access the Azure Vote application from the internet.

1. Create a file named `azure-vote.yaml`.
   - If you use the Azure Cloud Shell, this file can be created using `code`, `vi`, or `nano` as if working on a virtual or physical system
2. Copy in the following YAML definition:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-back
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-back
  template:
    metadata:
      labels:
        app: azure-vote-back
    spec:
      nodeSelector:
        "kubernetes.io/os": linux
      containers:
      - name: azure-vote-back
        image: mcr.microsoft.com/oss/bitnami/redis:6.0.8
        env:
        - name: ALLOW_EMPTY_PASSWORD
          value: "yes"
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 250m
            memory: 256Mi
        ports:
        - containerPort: 6379
          name: redis
---
apiVersion: v1
kind: Service
metadata:
```

```
  name: azure-vote-back
spec:
  ports:
  - port: 6379
  selector:
    app: azure-vote-back
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-front
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-front
  template:
    metadata:
      labels:
        app: azure-vote-front
    spec:
      nodeSelector:
        "kubernetes.io/os": linux
      containers:
      - name: azure-vote-front
        image: mcr.microsoft.com/azuredocs/azure-vote-front:v1
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 250m
            memory: 256Mi
        ports:
        - containerPort: 80
        env:
        - name: REDIS
          value: "azure-vote-back"
---
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-front
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: azure-vote-front
```

3. Deploy the application using the kubectl apply command and specify the name of your YAML manifest:

```
kubectl apply -f azure-vote.yaml
```

The following example resembles output showing the successfully created deployments and services:

```
deployment "azure-vote-back" created
service "azure-vote-back" created
deployment "azure-vote-front" created
service "azure-vote-front" created
```

Test the application

When the application runs, a Kubernetes service exposes the application front end to the internet. This process

can take a few minutes to complete.

Monitor progress using the kubectl get service command with the `--watch` argument.

```
kubectl get service azure-vote-front --watch
```

The **EXTERNAL-IP** output for the `azure-vote-front` service will initially show as *pending*.

```
NAME               TYPE           CLUSTER-IP   EXTERNAL-IP   PORT(S)        AGE
azure-vote-front   LoadBalancer   10.0.37.27   <pending>     80:30572/TCP   6s
```

Once the **EXTERNAL-IP** address changes from *pending* to an actual public IP address, use `CTRL-C` to stop the `kubectl` watch process. The following example output shows a valid public IP address assigned to the service:

```
azure-vote-front   LoadBalancer   10.0.37.27   52.179.23.131   80:30572/TCP   2m
```

To see the Azure Vote app in action, open a web browser to the external IP address of your service.

Clean up resources

- Azure CLI
- Azure PowerShell

To avoid Azure charges, if you don't plan on going through the tutorials that follow, clean up your unnecessary resources. Use the az group delete command to remove the resource group, container service, and all related resources.

```
az group delete --name myResourceGroup --yes --no-wait
```

> **NOTE**
> In this quickstart, the AKS cluster was created with a system-assigned managed identity (the default identity option). This identity is managed by the platform and does not require removal.

Next steps

In this quickstart, you deployed a Kubernetes cluster and then deployed a sample multi-container application to it.

To learn more about AKS, and walk through a complete code to deployment example, continue to the Kubernetes cluster tutorial.

AKS tutorial

Quickstart: Deploy a container instance in Azure using Bicep

9/21/2022 • 3 minutes to read • Edit Online

Use Azure Container Instances to run serverless Docker containers in Azure with simplicity and speed. Deploy an application to a container instance on-demand when you don't need a full container orchestration platform like Azure Kubernetes Service. In this quickstart, you use a Bicep file to deploy an isolated Docker container and make its web application available with a public IP address.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Name for the container group')
param name string = 'acilinuxpublicipcontainergroup'

@description('Location for all resources.')
param location string = resourceGroup().location

@description('Container image to deploy. Should be of the form repoName/imagename:tag for images stored in
public Docker Hub, or a fully qualified URI for other registries. Images from private registries require
additional registry credentials.')
param image string = 'mcr.microsoft.com/azuredocs/aci-helloworld'

@description('Port to open on the container and the public IP address.')
param port int = 80

@description('The number of CPU cores to allocate to the container.')
param cpuCores int = 1

@description('The amount of memory to allocate to the container in gigabytes.')
param memoryInGb int = 2

@description('The behavior of Azure runtime if container has stopped.')
@allowed([
  'Always'
  'Never'
  'OnFailure'
])
param restartPolicy string = 'Always'

resource containerGroup 'Microsoft.ContainerInstance/containerGroups@2021-09-01' = {
  name: name
  location: location
  properties: {
    containers: [
      {
        name: name
        properties: {
```

```
          image: image
          ports: [
            {
              port: port
              protocol: 'TCP'
            }
          ]
          resources: {
            requests: {
              cpu: cpuCores
              memoryInGB: memoryInGb
            }
          }
        }
      }
    ]
    osType: 'Linux'
    restartPolicy: restartPolicy
    ipAddress: {
      type: 'Public'
      ports: [
        {
          port: port
          protocol: 'TCP'
        }
      ]
    }
  }
}

output containerIPv4Address string = containerGroup.properties.ipAddress.ip
```

The following resource is defined in the Bicep file:

- **Microsoft.ContainerInstance/containerGroups**: create an Azure container group. This Bicep file defines a group consisting of a single container instance.

More Azure Container Instances template samples can be found in the quickstart template gallery.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

   ```
   az group create --name exampleRG --location eastus
   az deployment group create --resource-group exampleRG --template-file main.bicep
   ```

   When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

View container logs

Viewing the logs for a container instance is helpful when troubleshooting issues with your container or the application it runs. Use the Azure portal, Azure CLI, or Azure PowerShell to view the container's logs.

- CLI
- PowerShell

```
az container logs --resource-group exampleRG --name acilinuxpublicipcontainergroup
```

> **NOTE**
> It may take a few minutes for the HTTP GET request to generate.

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the container and all of the resources in the resource group.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you created an Azure container instance using Bicep. If you'd like to build a container image and deploy it from a private Azure container registry, continue to the Azure Container Instances tutorial.

Tutorial: Create a container image for deployment to Azure Container Instances

Quickstart: Create a container registry by using a Bicep file

9/21/2022 • 2 minutes to read • Edit Online

This quickstart shows how to create an Azure Container Registry instance by using a Bicep file.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

Use Visual studio code or your favorite editor to create a file with the following content and name it **main.bicep**:

```
@minLength(5)
@maxLength(50)
@description('Provide a globally unique name of your Azure Container Registry')
param acrName string = 'acr${uniqueString(resourceGroup().id)}'

@description('Provide a location for the registry.')
param location string = resourceGroup().location

@description('Provide a tier of your Azure Container Registry.')
param acrSku string = 'Basic'

resource acrResource 'Microsoft.ContainerRegistry/registries@2021-06-01-preview' = {
  name: acrName
  location: location
  sku: {
    name: acrSku
  }
  properties: {
    adminUserEnabled: false
  }
}

@description('Output the login server property for later use')
output loginServer string = acrResource.properties.loginServer
```

The following resource is defined in the Bicep file:

- **Microsoft.ContainerRegistry/registries**: create an Azure container registry

More Azure Container Registry template samples can be found in the quickstart template gallery.

Deploy the Bicep file

To deploy the file you've created, open PowerShell or Azure CLI. If you want to use the integrated Visual Studio Code terminal, select the `ctrl` + `` ` `` key combination. Change the current directory to where the Bicep file is

located.

- CLI
- PowerShell

```
az group create --name myContainerRegRG --location centralus

az deployment group create --resource-group myContainerRegRG --template-file main.bicep --parameters
acrName={your-unique-name}
```

> **NOTE**
>
> Replace **{your-unique-name}**, including the curly braces, with a unique container registry name.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal or a tool such as the Azure CLI to review the properties of the container registry.

1. In the portal, search for **Container Registries**, and select the container registry you created.
2. On the **Overview** page, note the **Login server** of the registry. Use this URI when you use Docker to tag and push images to your registry. For information, see Push your first image using the Docker CLI.

Clean up resources

When you no longer need the resource, delete the resource group, and the registry. To do so, go to the Azure portal, select the resource group that contains the registry, and then select **Delete resource group**.

Next steps

In this quickstart, you created an Azure Container Registry with a Bicep file. Continue to the Azure Container Registry tutorials for a deeper look at ACR.

Azure Container Registry tutorials

For a step-by-step tutorial that guides you through the process of creating a Bicep file, see:

Quickstart: Create Bicep files with Visual Studio Code

Quickstart: Create a Service Fabric cluster using Bicep

9/21/2022 • 9 minutes to read • Edit Online

Azure Service Fabric is a distributed systems platform that makes it easy to package, deploy, and manage scalable and reliable microservices and containers. A Service Fabric *cluster* is a network-connected set of virtual machines into which your microservices are deployed and managed. This article describes how to deploy a Service Fabric test cluster in Azure using Bicep.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

This five-node Windows cluster is secured with a self-signed certificate and thus only intended for instructional purposes (rather than production workloads). We'll use Azure PowerShell to deploy the Bicep file.

Prerequisites

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

Install Service Fabric SDK and PowerShell modules

To complete this quickstart, you'll need to install the Service Fabric SDK and PowerShell module.

Download the sample Bicep file and certificate helper script

Clone or download the Azure Resource Manager Quickstart Templates repo. Alternatively, copy down locally the following files we'll be using from the *service-fabric-secure-cluster-5-node-1-nodetype* folder:

- New-ServiceFabricClusterCertificate.ps1
- main.bicep
- azuredeploy.parameters.json

Sign in to Azure

Sign in to Azure and designate the subscription to use for creating your Service Fabric cluster.

```
# Sign in to your Azure account
Login-AzAccount -SubscriptionId "<subscription ID>"
```

Create a self-signed certificate stored in Key Vault

Service Fabric uses X.509 certificates to secure a cluster and provide application security features, and Key Vault to manage those certificates. Successful cluster creation requires a cluster certificate to enable node-to-node communication. For the purpose of creating this quickstart test cluster, we'll create a self-signed certificate for cluster authentication. Production workloads require certificates created using a correctly configured Windows Server certificate service or one from an approved certificate authority (CA).

```
# Designate unique (within cloudapp.azure.com) names for your resources
$resourceGroupName = "SFQuickstartRG"
$keyVaultName = "SFQuickstartKV"

# Create a new resource group for your Key Vault and Service Fabric cluster
New-AzResourceGroup -Name $resourceGroupName -Location SouthCentralUS

# Create a Key Vault enabled for deployment
New-AzKeyVault -VaultName $keyVaultName -ResourceGroupName $resourceGroupName -Location SouthCentralUS -
EnabledForDeployment

# Generate a certificate and upload it to Key Vault
.\scripts\New-ServiceFabricClusterCertificate.ps1
```

The script will prompt you for the following (be sure to modify *CertDNSName* and *KeyVaultName* from the example values below):

- **Password**: Password!1
- **CertDNSName**: *sfquickstart*.southcentralus.cloudapp.azure.com
- **KeyVaultName**: *SFQuickstartKV*
- **KeyVaultSecretName**: clustercert

Upon completion, the script will provide the parameter values needed for deployment. Be sure to store these in the following variables, as they will be needed for deployment:

```
$sourceVaultId = "<Source Vault Resource Id>"
$certUrlValue = "<Certificate URL>"
$certThumbprint = "<Certificate Thumbprint>"
```

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Location of the Cluster')
param location string = resourceGroup().location

@description('Name of your cluster - Between 3 and 23 characters. Letters and numbers only')
param clusterName string

@description('Remote desktop user Id')
param adminUsername string

@description('Remote desktop user password. Must be a strong password')
@secure()
param adminPassword string

@description('VM image Publisher')
param vmImagePublisher string = 'MicrosoftWindowsServer'

@description('VM image offer')
param vmImageOffer string = 'WindowsServer'

@description('VM image SKU')
param vmImageSku string = '2019-Datacenter'

@description('VM image version')
param vmImageVersion string = 'latest'

@description('Input endpoint1 for the application to use. Replace it with what your application uses')
param loadBalancedAppPort1 int = 80
```

```
@description('Input endpoint2 for the application to use. Replace it with what your application uses')
param loadBalancedAppPort2 int = 8081

@description('The store name where the cert will be deployed in the virtual machine')
@allowed([
  'My'
])
param certificateStoreValue string = 'My'

@description('Certificate Thumbprint')
param certificateThumbprint string

@description('Resource Id of the key vault, is should be in the format of /subscriptions/<Sub
ID>/resourceGroups/<Resource group name>/providers/Microsoft.KeyVault/vaults/<vault name>')
param sourceVaultResourceId string

@description('Refers to the location URL in your key vault where the certificate was uploaded')
param certificateUrlValue string

@description('Protection level.Three values are allowed - EncryptAndSign, Sign, None. It is best to keep the
default of EncryptAndSign, unless you have a need not to')
@allowed([
  'None'
  'Sign'
  'EncryptAndSign'
])
param clusterProtectionLevel string = 'EncryptAndSign'

@description('Instance count for node type')
param nt0InstanceCount int = 5

@description('The drive to use to store data on a cluster node.')
@allowed([
  'OS'
  'Temp'
])
param nodeDataDrive string = 'Temp'

@description('The VM size to use for cluster nodes.')
param nodeTypeSize string = 'Standard_D2_v3'

param tenantId string
param clusterApplication string
param clientapplication string

var dnsName = clusterName
var vmName = 'vm'
var virtualNetworkName = 'VNet'
var addressPrefix = '10.0.0.0/16'
var nicName = 'NIC'
var lbIPName = 'PublicIP-LB-FE'
var overProvision = false
var nt0applicationStartPort = 20000
var nt0applicationEndPort = 30000
var nt0ephemeralStartPort = 49152
var nt0ephemeralEndPort = 65534
var nt0fabricTcpGatewayPort = 19000
var nt0fabricHttpGatewayPort = 19080
var subnet0Name = 'Subnet-0'
var subnet0Prefix = '10.0.0.0/24'
var subnet0Ref = resourceId('Microsoft.Network/virtualNetworks/subnets/', virtualNetworkName, subnet0Name)
var supportLogStorageAccountName = '${uniqueString(resourceGroup().id)}2'
var applicationDiagnosticsStorageAccountName = '${uniqueString(resourceGroup().id)}3'
var lbName = 'LB-${clusterName}-${vmNodeType0Name}'
var lbIPConfig0 = resourceId('Microsoft.Network/loadBalancers/frontendIPConfigurations/', lbName,
'LoadBalancerIPConfig')
var lbPoolID0 = resourceId('Microsoft.Network/loadBalancers/backendAddressPools', lbName,
'LoadBalancerBEAddressPool')
var lbProbeID0 = resourceId('Microsoft.Network/loadBalancers/probes', lbName, 'FabricGatewayProbe')
```

```
var lbHttpProbeID0 = resourceId('Microsoft.Network/loadBalancers/probes', lbName, 'FabricHttpGatewayProbe')
var lbNatPoolID0 = resourceId('Microsoft.Network/loadBalancers/inboundNatPools', lbName,
'LoadBalancerBEAddressNatPool')
var vmNodeType0Name = toLower('NT1${vmName}')
var vmNodeType0Size = nodeTypeSize

resource supportLogStorageAccount 'Microsoft.Storage/storageAccounts@2021-09-01' = {
  name: supportLogStorageAccountName
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'Storage'
  tags: {
    resourceType: 'Service Fabric'
    clusterName: clusterName
  }
  properties: {}
}

resource applicationDiagnosticsStorageAccount 'Microsoft.Storage/storageAccounts@2021-09-01' = {
  name: applicationDiagnosticsStorageAccountName
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'Storage'
  tags: {
    resourceType: 'Service Fabric'
    clusterName: clusterName
  }
  properties: {}
}

resource virtualNetwork 'Microsoft.Network/virtualNetworks@2021-08-01' = {
  name: virtualNetworkName
  location: location
  tags: {
    resourceType: 'Service Fabric'
    clusterName: clusterName
  }
  properties: {
    addressSpace: {
      addressPrefixes: [
        addressPrefix
      ]
    }
    subnets: [
      {
        name: subnet0Name
        properties: {
          addressPrefix: subnet0Prefix
        }
      }
    ]
  }
}

resource lbIP 'Microsoft.Network/publicIPAddresses@2021-08-01' = {
  name: lbIPName
  location: location
  tags: {
    resourceType: 'Service Fabric'
    clusterName: clusterName
  }
  properties: {
    dnsSettings: {
      domainNameLabel: dnsName
    }
```

```
    }
    publicIPAllocationMethod: 'Dynamic'
  }
}

resource lb 'Microsoft.Network/loadBalancers@2021-08-01' = {
  name: lbName
  location: location
  tags: {
    resourceType: 'Service Fabric'
    clusterName: clusterName
  }
  properties: {
    frontendIPConfigurations: [
      {
        name: 'LoadBalancerIPConfig'
        properties: {
          publicIPAddress: {
            id: lbIP.id
          }
        }
      }
    ]
    backendAddressPools: [
      {
        name: 'LoadBalancerBEAddressPool'
        properties: {}
      }
    ]
    loadBalancingRules: [
      {
        name: 'LBRule'
        properties: {
          backendAddressPool: {
            id: lbPoolID0
          }
          backendPort: nt0fabricTcpGatewayPort
          enableFloatingIP: false
          frontendIPConfiguration: {
            id: lbIPConfig0
          }
          frontendPort: nt0fabricTcpGatewayPort
          idleTimeoutInMinutes: 5
          probe: {
            id: lbProbeID0
          }
          protocol: 'Tcp'
        }
      }
      {
        name: 'LBHttpRule'
        properties: {
          backendAddressPool: {
            id: lbPoolID0
          }
          backendPort: nt0fabricHttpGatewayPort
          enableFloatingIP: false
          frontendIPConfiguration: {
            id: lbIPConfig0
          }
          frontendPort: nt0fabricHttpGatewayPort
          idleTimeoutInMinutes: 5
          probe: {
            id: lbHttpProbeID0
          }
          protocol: 'Tcp'
        }
      }
      {
        name: 'AppPortLBRule1'
```

```
        name: 'AppPortLBRule1'
        properties: {
          backendAddressPool: {
            id: lbPoolID0
          }
          backendPort: loadBalancedAppPort1
          enableFloatingIP: false
          frontendIPConfiguration: {
            id: lbIPConfig0
          }
          frontendPort: loadBalancedAppPort1
          idleTimeoutInMinutes: 5
          probe: {
            id: resourceId('Microsoft.Network/loadBalancers/probes', lbName, 'AppPortProbe1')
          }
          protocol: 'Tcp'
        }
      }
      {
        name: 'AppPortLBRule2'
        properties: {
          backendAddressPool: {
            id: lbPoolID0
          }
          backendPort: loadBalancedAppPort2
          enableFloatingIP: false
          frontendIPConfiguration: {
            id: lbIPConfig0
          }
          frontendPort: loadBalancedAppPort2
          idleTimeoutInMinutes: 5
          probe: {
            id: resourceId('Microsoft.Network/loadBalancers/probes', lbName, 'AppPortProbe2')
          }
          protocol: 'Tcp'
        }
      }
    ]
    probes: [
      {
        name: 'FabricGatewayProbe'
        properties: {
          intervalInSeconds: 5
          numberOfProbes: 2
          port: nt0fabricTcpGatewayPort
          protocol: 'Tcp'
        }
      }
      {
        name: 'FabricHttpGatewayProbe'
        properties: {
          intervalInSeconds: 5
          numberOfProbes: 2
          port: nt0fabricHttpGatewayPort
          protocol: 'Tcp'
        }
      }
      {
        name: 'AppPortProbe1'
        properties: {
          intervalInSeconds: 5
          numberOfProbes: 2
          port: loadBalancedAppPort1
          protocol: 'Tcp'
        }
      }
      {
        name: 'AppPortProbe2'
        properties: {
```

```
          intervalInSeconds: 5
          numberOfProbes: 2
          port: loadBalancedAppPort2
          protocol: 'Tcp'
        }
      }
    ]
    inboundNatPools: [
      {
        name: 'LoadBalancerBEAddressNatPool'
        properties: {
          backendPort: 3389
          frontendIPConfiguration: {
            id: lbIPConfig0
          }
          frontendPortRangeEnd: 4500
          frontendPortRangeStart: 3389
          protocol: 'Tcp'
        }
      }
    ]
  }
}

resource vmNodeType0 'Microsoft.Compute/virtualMachineScaleSets@2021-11-01' = {
  name: vmNodeType0Name
  location: location
  sku: {
    name: vmNodeType0Size
    capacity: nt0InstanceCount
    tier: 'Standard'
  }
  tags: {
    resourceType: 'Service Fabric'
    clusterName: clusterName
  }
  properties: {
    overprovision: overProvision
    upgradePolicy: {
      mode: 'Automatic'
    }
    virtualMachineProfile: {
      extensionProfile: {
        extensions: [
          {
            name: 'ServiceFabricNodeVmExt_vmNodeType0Name'
            properties: {
              type: 'ServiceFabricNode'
              autoUpgradeMinorVersion: true
              protectedSettings: {
                StorageAccountKey1:  supportLogStorageAccount.listKeys().keys[0].value
                StorageAccountKey2: supportLogStorageAccount.listkeys().keys[1].value
              }
              publisher: 'Microsoft.Azure.ServiceFabric'
              settings: {
                clusterEndpoint: cluster.properties.clusterEndpoint
                nodeTypeRef: vmNodeType0Name
                dataPath: '${((nodeDataDrive == 'OS') ? 'C' : 'D')}:\\\\SvcFab'
                durabilityLevel: 'Silver'
                nicPrefixOverride: subnet0Prefix
                certificate: {
                  thumbprint: certificateThumbprint
                  x509StoreName: certificateStoreValue
                }
              }
              typeHandlerVersion: '1.0'
            }
          }
          {
```

```
            name: 'VMDiagnosticsVmExt_vmNodeType0Name'
            properties: {
              type: 'IaaSDiagnostics'
              autoUpgradeMinorVersion: true
              protectedSettings: {
                storageAccountName: applicationDiagnosticsStorageAccountName
                storageAccountKey: listKeys(applicationDiagnosticsStorageAccount.id, '2021-01-
01').keys[0].value
                storageAccountEndPoint: 'https://${environment().suffixes.storage}'
              }
              publisher: 'Microsoft.Azure.Diagnostics'
              settings: {
                WadCfg: {
                  DiagnosticMonitorConfiguration: {
                    overallQuotaInMB: '50000'
                    EtwProviders: {
                      EtwEventSourceProviderConfiguration: [
                        {
                          provider: 'Microsoft-ServiceFabric-Actors'
                          scheduledTransferKeywordFilter: '1'
                          scheduledTransferPeriod: 'PT5M'
                          DefaultEvents: {
                            eventDestination: 'ServiceFabricReliableActorEventTable'
                          }
                        }
                        {
                          provider: 'Microsoft-ServiceFabric-Services'
                          scheduledTransferPeriod: 'PT5M'
                          DefaultEvents: {
                            eventDestination: 'ServiceFabricReliableServiceEventTable'
                          }
                        }
                      ]
                      EtwManifestProviderConfiguration: [
                        {
                          provider: 'cbd93bc2-71e5-4566-b3a7-595d8eeca6e8'
                          scheduledTransferLogLevelFilter: 'Information'
                          scheduledTransferKeywordFilter: '4611686018427387904'
                          scheduledTransferPeriod: 'PT5M'
                          DefaultEvents: {
                            eventDestination: 'ServiceFabricSystemEventTable'
                          }
                        }
                      ]
                    }
                  }
                }
                StorageAccount: applicationDiagnosticsStorageAccountName
              }
              typeHandlerVersion: '1.5'
            }
          }
        ]
      }
      networkProfile: {
        networkInterfaceConfigurations: [
          {
            name: '${nicName}-0'
            properties: {
              ipConfigurations: [
                {
                  name: '${nicName}-0'
                  properties: {
                    loadBalancerBackendAddressPools: [
                      {
                        id: lbPoolID0
                      }
                    ]
                    loadBalancerInboundNatPools: [
```

```
                      {
                        id: lbNatPoolID0
                      }
                    ]
                    subnet: {
                      id: subnet0Ref
                    }
                  }
                }
              }
              ]
              primary: true
            }
          }
        ]
      }
      osProfile: {
        adminPassword: adminPassword
        adminUsername: adminUsername
        computerNamePrefix: vmNodeType0Name
        secrets: [
          {
            sourceVault: {
              id: sourceVaultResourceId
            }
            vaultCertificates: [
              {
                certificateStore: certificateStoreValue
                certificateUrl: certificateUrlValue
              }
            ]
          }
        ]
      }
      storageProfile: {
        imageReference: {
          publisher: vmImagePublisher
          offer: vmImageOffer
          sku: vmImageSku
          version: vmImageVersion
        }
        osDisk: {
          managedDisk: {
            storageAccountType: 'StandardSSD_LRS'
          }
          caching: 'ReadOnly'
          createOption: 'FromImage'
        }
      }
    }
  }
  dependsOn: [
    virtualNetwork
    lb
  ]
}

resource cluster 'Microsoft.ServiceFabric/clusters@2021-06-01' = {
  name: clusterName
  location: location
  tags: {
    resourceType: 'Service Fabric'
    clusterName: clusterName
  }
  properties: {
    azureActiveDirectory: {
      clientApplication: clientapplication
      clusterApplication: clusterApplication
      tenantId: tenantId
    }
```

```
    certificate: {
      thumbprint: certificateThumbprint
      x509StoreName: certificateStoreValue
    }
    diagnosticsStorageAccountConfig: {
      blobEndpoint: reference(supportLogStorageAccount.id, '2021-01-01').primaryEndpoints.blob
      protectedAccountKeyName: 'StorageAccountKey1'
      queueEndpoint: reference(supportLogStorageAccount.id, '2021-01-01').primaryEndpoints.queue
      storageAccountName: supportLogStorageAccountName
      tableEndpoint: reference(supportLogStorageAccount.id, '2021-01-01').primaryEndpoints.table
    }
    fabricSettings: [
      {
        parameters: [
          {
            name: 'ClusterProtectionLevel'
            value: clusterProtectionLevel
          }
        ]
        name: 'Security'
      }
    ]
    managementEndpoint: 'https://${lbIP.properties.dnsSettings.fqdn}:${nt0fabricHttpGatewayPort}'
    nodeTypes: [
      {
        name: vmNodeType0Name
        applicationPorts: {
          endPort: nt0applicationEndPort
          startPort: nt0applicationStartPort
        }
        clientConnectionEndpointPort: nt0fabricTcpGatewayPort
        durabilityLevel: 'Silver'
        ephemeralPorts: {
          endPort: nt0ephemeralEndPort
          startPort: nt0ephemeralStartPort
        }
        httpGatewayEndpointPort: nt0fabricHttpGatewayPort
        isPrimary: true
        vmInstanceCount: nt0InstanceCount
      }
    ]
    reliabilityLevel: 'Silver'
    upgradeMode: 'Automatic'
    vmImage: 'Windows'
  }
}

output clusterProperties object = cluster.properties
```

Multiple Azure resources are defined in the Bicep file:

- Microsoft.Storage/storageAccounts
- Microsoft.Network/virtualNetworks
- Microsoft.Network/publicIPAddresses
- Microsoft.Network/loadBalancers
- Microsoft.Compute/virtualMachineScaleSets
- Microsoft.ServiceFabric/clusters

Customize the parameters file

Open *azuredeploy.parameters.json* and edit the parameter values so that:

- **clusterName** matches the value you supplied for *CertDNSName* when creating your cluster certificate
- **adminUserName** is some value other than the default *GEN-UNIQUE* token

- **adminPassword** is some value other than the default *GEN-PASSWORD* token
- **certificateThumbprint**, **sourceVaultResourceId**, and **certificateUrlValue** are all empty string (`""`)

For example:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "clusterName": {
      "value": "sfquickstart"
    },
    "adminUsername": {
      "value": "testadm"
    },
    "adminPassword": {
      "value": "Password#1234"
    },
    "certificateThumbprint": {
      "value": ""
    },
    "sourceVaultResourceId": {
      "value": ""
    },
    "certificateUrlValue": {
      "value": ""
    }
  }
}
```

Deploy the Bicep file

Store the paths of your Bicep file and parameter file in variables, then deploy the Bicep file.

```
$templateFilePath = "<full path to main.bicep>"
$parameterFilePath = "<full path to azuredeploy.parameters.json>"

New-AzResourceGroupDeployment `
    -ResourceGroupName $resourceGroupName `
    -TemplateFile $templateFilePath `
    -TemplateParameterFile $parameterFilePath `
    -CertificateThumbprint $certThumbprint `
    -CertificateUrlValue $certUrlValue `
    -SourceVaultResourceId $sourceVaultId `
    -Verbose
```

Review deployed resources

Once the deployment completes, find the `managementEndpoint` value in the output and open the address in a web browser to view your cluster in Service Fabric Explorer.

You can also find the Service Fabric Explorer endpoint from your Service Explorer resource blade in Azure portal.

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next, remove the cluster certificate from your local store. List installed certificates to find the thumbprint for

your cluster:

```
Get-ChildItem Cert:\CurrentUser\My\
```

Then remove the certificate:

```
Get-ChildItem Cert:\CurrentUser\My\{THUMBPRINT} | Remove-Item
```

Next steps

To learn how to create Bicep files with Visual Studio Code, see:

Quickstart: Create Bicep files with Visual Studio Code

Quickstart: Create an Azure Cache for Redis using Bicep

9/21/2022 • 3 minutes to read • Edit Online

Learn how to use Bicep to deploy a cache using Azure Cache for Redis. After you deploy the cache, use it with an existing storage account to keep diagnostic data. Learn how to define which resources are deployed and how to define parameters that are specified when the deployment is executed. You can use this Bicep file for your own deployments, or customize it to meet your requirements.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

- **Azure subscription**: If you don't have an Azure subscription, create a free account before you begin.
- **A storage account**: To create one, see Create an Azure Storage account. The storage account is used for diagnostic data. Create the storage account in a new resource group named **exampleRG**.

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Specify the name of the Azure Redis Cache to create.')
param redisCacheName string = 'redisCache-${uniqueString(resourceGroup().id)}'

@description('Location of all resources')
param location string = resourceGroup().location

@description('Specify the pricing tier of the new Azure Redis Cache.')
@allowed([
  'Basic'
  'Standard'
  'Premium'
])
param redisCacheSKU string = 'Standard'

@description('Specify the family for the sku. C = Basic/Standard, P = Premium.')
@allowed([
  'C'
  'P'
])
param redisCacheFamily string = 'C'

@description('Specify the size of the new Azure Redis Cache instance. Valid values: for C (Basic/Standard)
family (0, 1, 2, 3, 4, 5, 6), for P (Premium) family (1, 2, 3, 4)')
@allowed([
  0
  1
  2
  3
  4
  5
  6
])
param redisCacheCapacity int = 1
```

```
@description('Specify a boolean value that indicates whether to allow access via non-SSL ports.')
param enableNonSslPort bool = false

@description('Specify a boolean value that indicates whether diagnostics should be saved to the specified
storage account.')
param diagnosticsEnabled bool = false

@description('Specify the name of an existing storage account for diagnostics.')
param existingDiagnosticsStorageAccountName string

@description('Specify the resource group name of an existing storage account for diagnostics.')
param existingDiagnosticsStorageAccountResourceGroup string


resource diagnosticsStorage 'Microsoft.Storage/storageAccounts@2021-09-01' existing = {
  scope: resourceGroup(existingDiagnosticsStorageAccountResourceGroup)
  name: existingDiagnosticsStorageAccountName
}

resource redisCache 'Microsoft.Cache/Redis@2020-06-01' = {
  name: redisCacheName
  location: location
  properties: {
    enableNonSslPort: enableNonSslPort
    minimumTlsVersion: '1.2'
    sku: {
      capacity: redisCacheCapacity
      family: redisCacheFamily
      name: redisCacheSKU
    }
  }
}


resource Microsoft_Insights_diagnosticsettings_redisCacheName 'Microsoft.Insights/diagnosticSettings@2021-
05-01-preview' = {
  scope: redisCache
  name: redisCache.name
  properties: {
    storageAccountId: diagnosticsStorage.id
    metrics: [
      {
        timeGrain: 'AllMetrics'
        enabled: diagnosticsEnabled
        retentionPolicy: {
          days: 90
          enabled: diagnosticsEnabled
        }
      }
    ]
  }
}
```

The following resources are defined in the Bicep file:

- Microsoft.Cache/Redis
- Microsoft.Insights/diagnosticsettings

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.
   - CLI

- PowerShell

```
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
existingDiagnosticsStorageAccountName=<storage-name> existingDiagnosticsStorageAccountResourceGroup=
<resource-group>
```

> **NOTE**
>
> Replace **<storage-name>** with the name of the storage account you created at the beginning of this quickstart. Replace **<resource-group>** with the name of the resource group name in which your storage account is located.

When the deployment finishes, you see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When no longer needed, delete the resource group, which deletes the resources in the resource group.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this tutorial, you learned how to use Bicep to deploy a cache using Azure Cache for Redis. To learn more about Azure Cache for Redis and Bicep, see the articles below:

- Learn more about Azure Cache for Redis.
- Learn more about Bicep.

Quickstart: Create an Azure Cosmos DB and a container using Bicep

9/21/2022 • 4 minutes to read • Edit Online

APPLIES TO: SQL API

Azure Cosmos DB is Microsoft's fast NoSQL database with open APIs for any scale. You can use Azure Cosmos DB to quickly create and query key/value databases, document databases, and graph databases. Without a credit card or an Azure subscription, you can set up a free Try Azure Cosmos DB account. This quickstart focuses on the process of deploying a Bicep file to create an Azure Cosmos database and a container within that database. You can later store data in this container.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

An Azure subscription or free Azure Cosmos DB trial account.

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Azure Cosmos DB account name, max length 44 characters')
param accountName string = 'sql-${uniqueString(resourceGroup().id)}'

@description('Location for the Azure Cosmos DB account.')
param location string = resourceGroup().location

@description('The primary region for the Azure Cosmos DB account.')
param primaryRegion string

@description('The secondary region for the Azure Cosmos DB account.')
param secondaryRegion string

@allowed([
  'Eventual'
  'ConsistentPrefix'
  'Session'
  'BoundedStaleness'
  'Strong'
])
@description('The default consistency level of the Cosmos DB account.')
param defaultConsistencyLevel string = 'Session'

@minValue(10)
@maxValue(2147483647)
@description('Max stale requests. Required for BoundedStaleness. Valid ranges, Single Region: 10 to
2147483647. Multi Region: 100000 to 2147483647.')
param maxStalenessPrefix int = 100000

@minValue(5)
@maxValue(86400)
@description('Max lag time (minutes). Required for BoundedStaleness. Valid ranges, Single Region: 5 to
```

```
84600. Multi Region: 300 to 86400.')
param maxIntervalInSeconds int = 300

@allowed([
  true
  false
])
@description('Enable system managed failover for regions')
param systemManagedFailover bool = true

@description('The name for the database')
param databaseName string = 'myDatabase'

@description('The name for the container')
param containerName string = 'myContainer'

@minValue(400)
@maxValue(1000000)
@description('The throughput for the container')
param throughput int = 400

var consistencyPolicy = {
  Eventual: {
    defaultConsistencyLevel: 'Eventual'
  }
  ConsistentPrefix: {
    defaultConsistencyLevel: 'ConsistentPrefix'
  }
  Session: {
    defaultConsistencyLevel: 'Session'
  }
  BoundedStaleness: {
    defaultConsistencyLevel: 'BoundedStaleness'
    maxStalenessPrefix: maxStalenessPrefix
    maxIntervalInSeconds: maxIntervalInSeconds
  }
  Strong: {
    defaultConsistencyLevel: 'Strong'
  }
}
var locations = [
  {
    locationName: primaryRegion
    failoverPriority: 0
    isZoneRedundant: false
  }
  {
    locationName: secondaryRegion
    failoverPriority: 1
    isZoneRedundant: false
  }
]

resource account 'Microsoft.DocumentDB/databaseAccounts@2022-05-15' = {
  name: toLower(accountName)
  location: location
  kind: 'GlobalDocumentDB'
  properties: {
    consistencyPolicy: consistencyPolicy[defaultConsistencyLevel]
    locations: locations
    databaseAccountOfferType: 'Standard'
    enableAutomaticFailover: systemManagedFailover
  }
}

resource database 'Microsoft.DocumentDB/databaseAccounts/sqlDatabases@2022-05-15' = {
  name: '${account.name}/${databaseName}'
  properties: {
    resource: {
```

```
    resource: {
      id: databaseName
    }
  }
}

resource container 'Microsoft.DocumentDB/databaseAccounts/sqlDatabases/containers@2022-05-15' = {
  name: '${database.name}/${containerName}'
  properties: {
    resource: {
      id: containerName
      partitionKey: {
        paths: [
          '/myPartitionKey'
        ]
        kind: 'Hash'
      }
      indexingPolicy: {
        indexingMode: 'consistent'
        includedPaths: [
          {
            path: '/*'
          }
        ]
        excludedPaths: [
          {
            path: '/myPathToNotIndex/*'
          }
          {
            path: '/_etag/?'
          }
        ]
        compositeIndexes: [
          [
            {
              path: '/name'
              order: 'ascending'
            }
            {
              path: '/age'
              order: 'descending'
            }
          ]
        ]
        spatialIndexes: [
          {
            path: '/location/*'
            types: [
              'Point'
              'Polygon'
              'MultiPolygon'
              'LineString'
            ]
          }
        ]
      }
      defaultTtl: 86400
      uniqueKeyPolicy: {
        uniqueKeys: [
          {
            paths: [
              '/phoneNumber'
            ]
          }
        ]
      }
    }
    options: {
      throughput: throughput
    }
```

```
    }
  }
}
```

Three Azure resources are defined in the Bicep file:

- Microsoft.DocumentDB/databaseAccounts: Create an Azure Cosmos account.
- Microsoft.DocumentDB/databaseAccounts/sqlDatabases: Create an Azure Cosmos database.
- Microsoft.DocumentDB/databaseAccounts/sqlDatabases/containers: Create an Azure Cosmos container.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

   ```
   az group create --name exampleRG --location eastus
   az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
   primaryRegion=<primary-region> secondaryRegion=<secondary-region>
   ```

   > **NOTE**
   > Replace **<primary-region>** with the primary replica region for the Cosmos DB account, such as **WestUS**.
   > Replace **<secondary-region>** with the secondary replica region for the Cosmos DB account, such as **EastUS**.

   When the deployment finishes, you should see a message indicating the deployment succeeded.

Validate the deployment

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

If you plan to continue working with subsequent quickstarts and tutorials, you might want to leave these resources in place. When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you created an Azure Cosmos account, a database and a container by using a Bicep file and validated the deployment. To learn more about Azure Cosmos DB and Bicep, continue on to the articles below.

- Read an Overview of Azure Cosmos DB.
- Learn more about Bicep.
- Trying to do capacity planning for a migration to Azure Cosmos DB? You can use information about your existing database cluster for capacity planning.
  - If all you know is the number of vCores and servers in your existing database cluster, read about estimating request units using vCores or vCPUs.
  - If you know typical request rates for your current database workload, read about estimating request units using Azure Cosmos DB capacity planner.

Quickstart: Use Bicep to create an Azure Database for MariaDB server

9/21/2022 • 3 minutes to read • Edit Online

Azure Database for MariaDB is a managed service that you use to run, manage, and scale highly available MariaDB databases in the cloud. In this quickstart, you use Bicep to create an Azure Database for MariaDB server in PowerShell or Azure CLI.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

You'll need an Azure account with an active subscription. Create one for free.

Review the Bicep file

You create an Azure Database for MariaDB server with a defined set of compute and storage resources. To learn more, see Azure Database for MariaDB pricing tiers. You create the server within an Azure resource group.

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Server Name for Azure database for MariaDB')
param serverName string

@description('Database administrator login name')
@minLength(1)
param administratorLogin string

@description('Database administrator password')
@minLength(8)
@secure()
param administratorLoginPassword string

@description('Azure database for MariaDB compute capacity in vCores (2,4,8,16,32)')
param skuCapacity int = 2

@description('Azure database for MariaDB sku name ')
param skuName string = 'GP_Gen5_2'

@description('Azure database for MariaDB Sku Size ')
param skuSizeMB int = 51200

@description('Azure database for MariaDB pricing tier')
param skuTier string = 'GeneralPurpose'

@description('Azure database for MariaDB sku family')
param skuFamily string = 'Gen5'

@description('MariaDB version')
@allowed([
  '10.2'
  '10.3'
])
param mariadbVersion string = '10.3'
```

```
@description('Location for all resources.')
param location string = resourceGroup().location

@description('MariaDB Server backup retention days')
param backupRetentionDays int = 7

@description('Geo-Redundant Backup setting')
param geoRedundantBackup string = 'Disabled'

@description('Virtual Network Name')
param virtualNetworkName string = 'azure_mariadb_vnet'

@description('Subnet Name')
param subnetName string = 'azure_mariadb_subnet'

@description('Virtual Network RuleName')
param virtualNetworkRuleName string = 'AllowSubnet'

@description('Virtual Network Address Prefix')
param vnetAddressPrefix string = '10.0.0.0/16'

@description('Subnet Address Prefix')
param subnetPrefix string = '10.0.0.0/16'

var firewallrules = [
  {
    Name: 'rule1'
    StartIpAddress: '0.0.0.0'
    EndIpAddress: '255.255.255.255'
  }
  {
    Name: 'rule2'
    StartIpAddress: '0.0.0.0'
    EndIpAddress: '255.255.255.255'
  }
]

resource vnet 'Microsoft.Network/virtualNetworks@2021-05-01' = {
  name: virtualNetworkName
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        vnetAddressPrefix
      ]
    }
  }
}

resource subnet 'Microsoft.Network/virtualNetworks/subnets@2021-05-01' = {
  parent: vnet
  name: subnetName
  properties: {
    addressPrefix: subnetPrefix
  }
}

resource mariaDbServer 'Microsoft.DBforMariaDB/servers@2018-06-01' = {
  name: serverName
  location: location
  sku: {
    name: skuName
    tier: skuTier
    capacity: skuCapacity
    size: '${skuSizeMB}' //a string is expected here but a int for the storageProfile...
    family: skuFamily
  }
  properties: {
    createMode: 'Default'
```

```
    version: mariadbVersion
    administratorLogin: administratorLogin
    administratorLoginPassword: administratorLoginPassword
    storageProfile: {
      storageMB: skuSizeMB
      backupRetentionDays: backupRetentionDays
      geoRedundantBackup: geoRedundantBackup
    }
  }

  resource virtualNetworkRule 'virtualNetworkRules@2018-06-01' = {
    name: virtualNetworkRuleName
    properties: {
      virtualNetworkSubnetId: subnet.id
      ignoreMissingVnetServiceEndpoint: true
    }
  }
}

@batchSize(1)
resource firewallRules 'Microsoft.DBforMariaDB/servers/firewallRules@2018-06-01' = [for rule in
firewallrules: {
  name: '${mariaDbServer.name}/${rule.Name}'
  properties: {
    startIpAddress: rule.StartIpAddress
    endIpAddress: rule.EndIpAddress
  }
}]
```

The Bicep file defines five Azure resources:

- **Microsoft.Network/virtualNetworks**
- **Microsoft.Network/virtualNetworks/subnets**
- **Microsoft.DBforMariaDB/servers**
- **Microsoft.DBforMariaDB/servers/virtualNetworkRules**
- **Microsoft.DBforMariaDB/servers/firewallRules**

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
serverName=<server-name> administratorLogin=<admin-login>
```

> **NOTE**
>
> Replace **<server-name>** with the name of the server. Replace **<admin-login>** with the database administrator login name. The minimum required length is one character. You'll also be prompted to enter **administratorLoginPassword**. The minimum password length is eight characters.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

For a step-by-step tutorial that guides you through the process of creating a Bicep file using Visual Studio Code, see:

Quickstart: Create Bicep files with Visual Studio Code

Quickstart: Use Bicep to create an Azure Database for MySQL server

9/21/2022 • 3 minutes to read • Edit Online

APPLIES TO: Azure Database for MySQL - Single Server

Azure Database for MySQL is a managed service that you use to run, manage, and scale highly available MySQL databases in the cloud. In this quickstart, you use Bicep to create an Azure Database for MySQL server with virtual network integration. You can create the server in the Azure portal, Azure CLI, or Azure PowerShell.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

You need an Azure account with an active subscription. Create one for free.

- PowerShell
- CLI

- If you want to run the code locally, Azure PowerShell.

Review the Bicep file

You create an Azure Database for MySQL server with a defined set of compute and storage resources. To learn more, see Azure Database for MySQL pricing tiers. You create the server within an Azure resource group.

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Server Name for Azure database for MySQL')
param serverName string

@description('Database administrator login name')
@minLength(1)
param administratorLogin string

@description('Database administrator password')
@minLength(8)
@secure()
param administratorLoginPassword string

@description('Azure database for MySQL compute capacity in vCores (2,4,8,16,32)')
param skuCapacity int = 2

@description('Azure database for MySQL sku name ')
param skuName string = 'GP_Gen5_2'

@description('Azure database for MySQL Sku Size ')
param SkuSizeMB int = 5120

@description('Azure database for MySQL pricing tier')
@allowed([
  'Basic'
  'GeneralPurpose'
  'MemoryOptimized'
```

```
])
param SkuTier string = 'GeneralPurpose'

@description('Azure database for MySQL sku family')
param skuFamily string = 'Gen5'

@description('MySQL version')
@allowed([
  '5.6'
  '5.7'
  '8.0'
])
param mysqlVersion string = '8.0'

@description('Location for all resources.')
param location string = resourceGroup().location

@description('MySQL Server backup retention days')
param backupRetentionDays int = 7

@description('Geo-Redundant Backup setting')
param geoRedundantBackup string = 'Disabled'

@description('Virtual Network Name')
param virtualNetworkName string = 'azure_mysql_vnet'

@description('Subnet Name')
param subnetName string = 'azure_mysql_subnet'

@description('Virtual Network RuleName')
param virtualNetworkRuleName string = 'AllowSubnet'

@description('Virtual Network Address Prefix')
param vnetAddressPrefix string = '10.0.0.0/16'

@description('Subnet Address Prefix')
param subnetPrefix string = '10.0.0.0/16'

var firewallrules = [
  {
    Name: 'rule1'
    StartIpAddress: '0.0.0.0'
    EndIpAddress: '255.255.255.255'
  }
  {
    Name: 'rule2'
    StartIpAddress: '0.0.0.0'
    EndIpAddress: '255.255.255.255'
  }
]

resource vnet 'Microsoft.Network/virtualNetworks@2021-05-01' = {
  name: virtualNetworkName
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        vnetAddressPrefix
      ]
    }
  }
}

resource subnet 'Microsoft.Network/virtualNetworks/subnets@2021-05-01' = {
  parent: vnet
  name: subnetName
  properties: {
    addressPrefix: subnetPrefix
  }
```

```
}

resource mysqlDbServer 'Microsoft.DBforMySQL/servers@2017-12-01' = {
  name: serverName
  location: location
  sku: {
    name: skuName
    tier: SkuTier
    capacity: skuCapacity
    size: '${SkuSizeMB}'  //a string is expected here but a int for the storageProfile...
    family: skuFamily
  }
  properties: {
    createMode: 'Default'
    version: mysqlVersion
    administratorLogin: administratorLogin
    administratorLoginPassword: administratorLoginPassword
    storageProfile: {
      storageMB: SkuSizeMB
      backupRetentionDays: backupRetentionDays
      geoRedundantBackup: geoRedundantBackup
    }
  }

  resource virtualNetworkRule 'virtualNetworkRules@2017-12-01' = {
    name: virtualNetworkRuleName
    properties: {
      virtualNetworkSubnetId: subnet.id
      ignoreMissingVnetServiceEndpoint: true
    }
  }
}

@batchSize(1)
resource firewallRules 'Microsoft.DBforMySQL/servers/firewallRules@2017-12-01' = [for rule in firewallrules:
{
  name: '${mysqlDbServer.name}/${rule.Name}'
  properties: {
    startIpAddress: rule.StartIpAddress
    endIpAddress: rule.EndIpAddress
  }
}]
```

The Bicep file defines five Azure resources:

- Microsoft.Network/virtualNetworks
- Microsoft.Network/virtualNetworks/subnets
- Microsoft.DBforMySQL/servers
- Microsoft.DBforMySQL/servers/virtualNetworkRules
- Microsoft.DBforMySQL/servers/firewallRules

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.

2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
New-AzResourceGroup -Name exampleRG -Location eastus
New-AzResourceGroupDeployment -ResourceGroupName exampleRG -TemplateFile ./main.bicep -serverName "
<server-name>" -administratorLogin "<admin-login>"
```

> **NOTE**
>
> Replace **<server-name>** with the server name for Azure database for MySQL. Replace **<admin-login>** with the database administrator login name. You'll also be prompted to enter **administratorLoginPassword**. The minimum password length is eight characters.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
Get-AzResource -ResourceGroupName exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
Remove-AzResourceGroup -Name exampleRG
```

Next steps

For a step-by-step tutorial that guides you through the process of creating a Bicep file with Visual Studio Code, see:

Quickstart: Create Bicep files with Visual Studio Code

Quickstart: Use a Bicep to create an Azure Database for PostgreSQL - Flexible Server

9/21/2022 • 3 minutes to read • Edit Online

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

Flexible server is a managed service that you use to run, manage, and scale highly available PostgreSQL databases in the cloud. You can use Bicep to provision a PostgreSQL Flexible Server to deploy multiple servers or multiple databases on a server.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

An Azure account with an active subscription. Create one for free.

Review the Bicep

An Azure Database for PostgresSQL Server is the parent resource for one or more databases within a region. It provides the scope for management policies that apply to its databases: login, firewall, users, roles, and configurations.

Create a *postgres-flexible-server.bicep* file and copy the following Bicep into it.

```
param administratorLogin string

@secure()
param administratorLoginPassword string
param location string = resourceGroup().location
param serverName string
param serverEdition string = 'GeneralPurpose'
param skuSizeGB int = 128
param dbInstanceType string = 'Standard_D4ds_v4'
param haMode string = 'ZoneRedundant'
param availabilityZone string = '1'
param version string = '12'
param virtualNetworkExternalId string = ''
param subnetName string = ''
param privateDnsZoneArmResourceId string = ''

resource serverName_resource 'Microsoft.DBforPostgreSQL/flexibleServers@2021-06-01' = {
  name: serverName
  location: location
  sku: {
    name: dbInstanceType
    tier: serverEdition
  }
  properties: {
    version: version
    administratorLogin: administratorLogin
    administratorLoginPassword: administratorLoginPassword
    network: {
      delegatedSubnetResourceId: (empty(virtualNetworkExternalId) ? json('null') :
json('${virtualNetworkExternalId}/subnets/${subnetName}'))
      privateDnsZoneArmResourceId: (empty(virtualNetworkExternalId) ? json('null') :
privateDnsZoneArmResourceId)
    }
    highAvailability: {
      mode: haMode
    }
    storage: {
      storageSizeGB: skuSizeGB
    }
    backup: {
      backupRetentionDays: 7
      geoRedundantBackup: 'Disabled'
    }
    availabilityZone: availabilityZone
  }
}
```

These resources are defined in the Bicep file:

- Microsoft.DBforPostgreSQL/flexibleServers

Deploy the Bicep file

Select **Try it** from the following PowerShell code block to open Azure Cloud Shell.

```
$serverName = Read-Host -Prompt "Enter a name for the new Azure Database for PostgreSQL server"
$resourceGroupName = Read-Host -Prompt "Enter a name for the new resource group where the server will exist"
$location = Read-Host -Prompt "Enter an Azure region (for example, centralus) for the resource group"
$adminUser = Read-Host -Prompt "Enter the Azure Database for PostgreSQL server's administrator account name"
$adminPassword = Read-Host -Prompt "Enter the administrator password" -AsSecureString

New-AzResourceGroup -Name $resourceGroupName -Location $location # Use this command when you need to create
a new resource group for your deployment
New-AzResourceGroupDeployment -ResourceGroupName $resourceGroupName `
    -TemplateFile "postgres-flexible-server.bicep" `
    -serverName $serverName `
    -administratorLogin $adminUser `
    -administratorLoginPassword $adminPassword

Read-Host -Prompt "Press [ENTER] to continue ..."
```

Review deployed resources

Follow these steps to verify if your server was created in Azure.

- Azure portal
- PowerShell
- CLI

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

1. In the Azure portal, search for and select **Azure Database for PostgreSQL Flexible Servers**.
2. In the database list, select your new server to view the **Overview** page to manage the server.

Clean up resources

Keep this resource group, server, and single database if you want to go to the Next steps. The next steps show you how to connect and query your database using different methods.

To delete the resource group:

- Portal
- PowerShell
- Azure CLI

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

In the portal, select the resource group you want to delete.

1. Select **Delete resource group**.
2. To confirm the deletion, type the name of the resource group

Next steps

Migrate your database using dump and restore

Quickstart: Create instance of Azure Database Migration Service using Bicep

9/21/2022 • 2 minutes to read • Edit Online

Use Bicep to deploy an instance of the Azure Database Migration Service.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Name of the new migration service.')
param serviceName string

@description('Location where the resources will be deployed.')
param location string = resourceGroup().location

@description('Name of the new virtual network.')
param vnetName string

@description('Name of the new subnet associated with the virtual network.')
param subnetName string

resource vnet 'Microsoft.Network/virtualNetworks@2021-05-01' = {
  name: vnetName
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        '10.0.0.0/16'
      ]
    }
  }
}

resource subnet 'Microsoft.Network/virtualNetworks/subnets@2021-05-01' = {
  parent: vnet
  name: subnetName
  properties: {
    addressPrefix: '10.0.0.0/24'
  }
}

resource dataMigration 'Microsoft.DataMigration/services@2021-10-30-preview' = {
  name: serviceName
  location: location
  sku: {
    tier: 'Standard'
    size: '1 vCores'
    name: 'Standard_1vCores'
  }
  properties: {
    virtualSubnetId: subnet.id
  }
}
```

Three Azure resources are defined in the Bicep file:

- Microsoft.Network/virtualNetworks: Creates the virtual network.
- Microsoft.Network/virtualNetworks/subnets: Creates the subnet.
- Microsoft.DataMigration/services: Deploys an instance of the Azure Database Migration Service.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.
   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
serviceName=<service-name> vnetName=<vnet-name> subnetName=<subnet-name>
```

> **NOTE**
>
> Replace **<service-name>** with the name of the new migration service. Replace **<vnet-name>** with the name of the new virtual network. Replace **<subnet-name>** with the name of the new subnet associated with the virtual network.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

For other ways to deploy Azure Database Migration Service, see Azure portal.

To learn more, see an overview of Azure Database Migration Service.

Quickstart: Create an Azure App Configuration store using Bicep

9/21/2022 • 3 minutes to read • Edit Online

This quickstart describes how you can use Bicep to:

- Deploy an App Configuration store.
- Create key-values in an App Configuration store.
- Read key-values in an App Configuration store.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

> **NOTE**
> Bicep files use the same underlying engine as ARM templates. All of the tips, notes, and important information found in the ARM template quickstart apply here. It's recommended to reference this information when working with Bicep files.

```
@description('Specifies the name of the App Configuration store.')
param configStoreName string

@description('Specifies the Azure location where the app configuration store should be created.')
param location string = resourceGroup().location

@description('Specifies the names of the key-value resources. The name is a combination of key and label
with $ as delimiter. The label is optional.')
param keyValueNames array = [
  'myKey'
  'myKey$myLabel'
]

@description('Specifies the values of the key-value resources. It\'s optional')
param keyValueValues array = [
  'Key-value without label'
  'Key-value with label'
]

@description('Specifies the content type of the key-value resources. For feature flag, the value should be
application/vnd.microsoft.appconfig.ff+json;charset=utf-8. For Key Value reference, the value should be
application/vnd.microsoft.appconfig.keyvaultref+json;charset=utf-8. Otherwise, it\'s optional.')
param contentType string = 'the-content-type'

@description('Adds tags for the key-value resources. It\'s optional')
param tags object = {
  tag1: 'tag-value-1'
  tag2: 'tag-value-2'
}

resource configStore 'Microsoft.AppConfiguration/configurationStores@2021-10-01-preview' = {
  name: configStoreName
  location: location
  sku: {
    name: 'standard'
  }
}

resource configStoreKeyValue 'Microsoft.AppConfiguration/configurationStores/keyValues@2021-10-01-preview' =
[for (item, i) in keyValueNames: {
  parent: configStore
  name: item
  properties: {
    value: keyValueValues[i]
    contentType: contentType
    tags: tags
  }
}]

output reference_key_value_value string = configStoreKeyValue[0].properties.value
output reference_key_value_object object = configStoreKeyValue[1]
```

Two Azure resources are defined in the Bicep file:

- Microsoft.AppConfiguration/configurationStores: create an App Configuration store.
- Microsoft.AppConfiguration/configurationStores/keyValues: create a key-value inside the App Configuration store.

With this Bicep file, we create one key with two different values, one of which has a unique label.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.

2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
configStoreName=<store-name>
```

> **NOTE**
>
> Replace **<store-name>** with the name of the App Configuration store.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use Azure CLI or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

You can also use the Azure portal to list the resources:

1. Sign in to the Azure portal.
2. In the search box, enter *App Configuration*, then select **App Configuration** from the list.
3. Select the newly created App Configuration resource.
4. Under **Operations**, select **Configuration explorer**.
5. Verify that two key-values exist.

Clean up resources

When no longer needed, use Azure CLI or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

You can also use the Azure portal to delete the resource group:

1. Navigate to your resource group.
2. Select **Delete resource group**.
3. A tab will appear. Enter the resource group name and select **Delete**.

Next steps

To learn about adding feature flag and Key Vault reference to an App Configuration store, check out the ARM

template examples.

- app-configuration-store-ff
- app-configuration-store-keyvaultref

Quickstart: Use Bicep to create a lab in DevTest Labs

9/21/2022 • 2 minutes to read • Edit Online

This quickstart uses Bicep to create a lab in Azure DevTest Labs that has one Windows Server 2019 Datacenter virtual machine (VM) in it.

In this quickstart, you take the following actions:

- Review the Bicep file.
- Deploy the Bicep file to create a lab and VM.
- Verify the deployment.
- Clean up resources.

Prerequisites

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

Review the Bicep file

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

The Bicep file defines the following resource types:

- Microsoft.DevTestLab/labs creates the lab.
- Microsoft.DevTestLab/labs/virtualnetworks creates a virtual network.
- Microsoft.DevTestLab/labs/virtualmachines creates the lab VM.

```
@description('The name of the new lab instance to be created')
param labName string

@description('Location for all resources.')
param location string = resourceGroup().location

@description('The name of the vm to be created.')
param vmName string

@description('The size of the vm to be created.')
param vmSize string = 'Standard_D4_v3'

@description('The username for the local account that will be created on the new vm.')
param userName string

@description('The password for the local account that will be created on the new vm.')
@secure()
param password string

var labSubnetName = '${labVirtualNetworkName}Subnet'
var labVirtualNetworkId = labVirtualNetwork.id
var labVirtualNetworkName = 'Dtl${labName}'

resource lab 'Microsoft.DevTestLab/labs@2018-09-15' = {
  name: labName
  location: location
}

resource labVirtualNetwork 'Microsoft.DevTestLab/labs/virtualnetworks@2018-09-15' = {
  parent: lab
  name: labVirtualNetworkName
}

resource labVirtualMachine 'Microsoft.DevTestLab/labs/virtualmachines@2018-09-15' = {
  parent: lab
  name: vmName
  location: location
  properties: {
    userName: userName
    password: password
    labVirtualNetworkId: labVirtualNetworkId
    labSubnetName: labSubnetName
    size: vmSize
    allowClaim: false
    galleryImageReference: {
      offer: 'WindowsServer'
      publisher: 'MicrosoftWindowsServer'
      sku: '2019-Datacenter'
      osType: 'Windows'
      version: 'latest'
    }
  }
}

output labId string = lab.id
```

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
labName=<lab-name> vmName=<vm-name> userName=<user-name>
```

> **NOTE**
>
> Replace **<lab-name>** with the name of the new lab instance. Replace **<vm-name>** with the name of the new VM. Replace **<user-name>** with username of the local account that will be created on the new VM. You'll also be prompted to enter a password for the local account.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Validate the deployment

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

> **NOTE**
>
> The deployment also creates a resource group for the VM. The resource group contains VM resources like the IP address, network interface, and disk. The resource group appears in your subscription's **Resource groups** list with the name **<lab name>-<vm name>-<numerical string>**.

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and all of its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you created a lab that has a Windows VM. To learn how to connect to and manage lab VMs, see the next tutorial:

Tutorial: Work with lab VMs

Deploy a FHIR service within Azure Health Data Services using Bicep

9/21/2022 • 2 minutes to read • Edit Online

In this article, you'll learn how to deploy FHIR service within the Azure Health Data Services using Bicep.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

- PowerShell
- CLI

- An Azure account with an active subscription. Create one for free.
- If you want to run the code locally:
  - Azure PowerShell.

Review the Bicep file

The Bicep file used in this article is from Azure Quickstart Templates.

```
@description('The name of the service.')
param serviceName string

@description('Location of Azure API for FHIR')
@allowed([
  'australiaeast'
  'eastus'
  'eastus2'
  'japaneast'
  'northcentralus'
  'northeurope'
  'southcentralus'
  'southeastasia'
  'uksouth'
  'ukwest'
  'westcentralus'
  'westeurope'
  'westus2'
])
param location string

resource service 'Microsoft.HealthcareApis/services@2021-11-01' = {
  name: serviceName
  location: location
  kind: 'fhir-R4'
  properties: {
    authenticationConfiguration: {
      audience: 'https://${serviceName}.azurehealthcareapis.com'
      authority: uri(environment().authentication.loginEndpoint, subscription().tenantId)
    }
  }
}
```

The Bicep file defines three Azure resources:

- Microsoft.HealthcareApis/workspaces: create a Microsoft.HealthcareApis/workspaces resource.
- Microsoft.HealthcareApis/workspaces/fhirservices: create a Microsoft.HealthcareApis/workspaces/fhirservices resource.
- Microsoft.Storage/storageAccounts: create a Microsoft.Storage/storageAccounts resource.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

   ```
   New-AzResourceGroup -Name exampleRG -Location eastus
   New-AzResourceGroupDeployment -ResourceGroupName exampleRG -TemplateFile ./main.bicep -serviceName "
   <service-name>" -location "<location>"
   ```

   Replace **<service-name>** with the name of the service. Replace **<location>** with the location of the Azure API for FHIR. Location options include:

   - australiaeast
   - eastus
   - eastus2
   - japaneast
   - northcentralus
   - northeurope
   - southcentralus
   - southeastasia
   - uksouth
   - ukwest
   - westcentralus
   - westeurope
   - westus2

   > **NOTE**
   > When the deployment finishes, you should see a message indicating the deployment succeeded.

Review the deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
Get-AzResource -ResourceGroupName exampleRG
```

> **NOTE**
> You can also verify that the FHIR service is up and running by opening a browser and navigating to `https://<yourfhirservice>.azurehealthcareapis.com/metadata`. If the capability statement is automatically displayed or downloaded, your deployment was successful. Make sure to replace **<yourfhirservice>** with the **<service-name>** you used in the deployment step of this quickstart.

Clean up the resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
Remove-AzResourceGroup -Name exampleRG
```

Next steps

In this quickstart guide, you've deployed the FHIR service within Azure Health Data Services using Bicep. For more information about FHIR service supported features, proceed to the following article:

Supported FHIR Features

Quickstart: Route Blob storage events to web endpoint by using Bicep

9/21/2022 • 3 minutes to read • Edit Online

Azure Event Grid is an eventing service for the cloud. In this article, you use a Bicep file to create a Blob storage account, subscribe to events for that blob storage, and trigger an event to view the result. Typically, you send events to an endpoint that processes the event data and takes actions. However, to simplify this article, you send the events to a web app that collects and displays the messages.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Create a message endpoint

Before subscribing to the events for the Blob storage, let's create the endpoint for the event message. Typically, the endpoint takes actions based on the event data. To simplify this quickstart, you deploy a pre-built web app that displays the event messages. The deployed solution includes an App Service plan, an App Service web app, and source code from GitHub.

1. Select **Deploy to Azure** to deploy the solution to your subscription. In the Azure portal, provide values for the parameters.

   Deploy to Azure

2. The deployment may take a few minutes to complete. After the deployment has succeeded, view your web app to make sure it's running. In a web browser, navigate to: `https://<your-site-name>.azurewebsites.net`

3. You see the site but no events have been posted to it yet.

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Provide a unique name for the Blob Storage account.')
param storageAccountName string = 'storage${uniqueString(resourceGroup().id)}'

@description('Provide a location for the Blob Storage account that supports Event Grid.')
param location string = resourceGroup().location

@description('Provide a name for the Event Grid subscription.')
param eventSubName string = 'subToStorage'

@description('Provide the URL for the WebHook to receive events. Create your own endpoint for events.')
param endpoint string

@description('Provide a name for the system topic.')
param systemTopicName string = 'mystoragesystemtopic'

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-08-01' = {
  name: storageAccountName
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
  properties: {
    accessTier: 'Hot'
  }
}

resource systemTopic 'Microsoft.EventGrid/systemTopics@2021-12-01' = {
  name: systemTopicName
  location: location
  properties: {
    source: storageAccount.id
    topicType: 'Microsoft.Storage.StorageAccounts'
  }
}

resource eventSubscription 'Microsoft.EventGrid/systemTopics/eventSubscriptions@2021-12-01' = {
  parent: systemTopic
  name: eventSubName
  properties: {
    destination: {
      properties: {
        endpointUrl: endpoint
      }
      endpointType: 'WebHook'
    }
    filter: {
      includedEventTypes: [
        'Microsoft.Storage.BlobCreated'
        'Microsoft.Storage.BlobDeleted'
      ]
    }
  }
}
```

Two Azure resources are defined in the Bicep file:

- **Microsoft.Storage/storageAccounts**: create an Azure Storage account.
- **Microsoft.EventGrid/systemTopics**: create a system topic with the specified name for the storage account.
- **Microsoft.EventGrid/systemTopics/eventSubscriptions**: create an Azure Event Grid subscription for the system topic.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
endpoint=<endpoint>
```

> **NOTE**
>
> Replace **<endpoint >** with the URL of your web app and append `api/updates` to the URL.

When the deployment finishes, you should see a message indicating the deployment succeeded.

> **NOTE**
>
> You can find more Azure Event Grid template samples here.

Validate the deployment

View your web app again, and notice that a subscription validation event has been sent to it. Select the eye icon to expand the event data. Event Grid sends the validation event so the endpoint can verify that it wants to receive event data. The web app includes code to validate the subscription.

Now, let's trigger an event to see how Event Grid distributes the message to your endpoint.

You trigger an event for the Blob storage by uploading a file. The file doesn't need any specific content. The articles assumes you have a file named testfile.txt, but you can use any file.

When you upload the file to the Azure Blob storage, Event Grid sends a message to the endpoint you configured when subscribing. The message is in the JSON format and it contains an array with one or more events. In the following example, the JSON message contains an array with one event. View your web app and notice that a blob created event was received.

Clean up resources

When no longer needed, delete the resource group.

Next steps

For more information about Azure Resource Manager templates and Bicep, see the following articles:

- Azure Resource Manager documentation
- Define resources in Azure Resource Manager templates
- Azure Quickstart Templates
- Azure Event Grid templates.

Quickstart: Create and deploy a Consumption logic app workflow in multi-tenant Azure Logic Apps with Bicep

9/21/2022 • 2 minutes to read • Edit Online

Applies to: **Azure Logic Apps (Consumption)**

Azure Logic Apps is a cloud service that helps you create and run automated workflows that integrate data, apps, cloud-based services, and on-premises systems by choosing from hundreds of connectors. This quickstart focuses on the process for deploying a Bicep file to create a basic Consumption logic app workflow that checks the status for Azure on an hourly schedule and runs in multi-tenant Azure Logic Apps.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

The quickstart template creates a Consumption logic app workflow that uses the *built-in* Recurrence trigger, which is set to run every hour, and a built-in HTTP action, which calls a URL that returns the status for Azure. Built-in operations run natively on Azure Logic Apps platform.

This Bicep file creates the following Azure resource:

- **Microsoft.Logic/workflows**, which creates the workflow for a logic app.

```
@description('The name of the logic app to create.')
param logicAppName string

@description('A test URI')
param testUri string = 'https://status.azure.com/en-us/status/'

@description('Location for all resources.')
param location string = resourceGroup().location

var frequency = 'Hour'
var interval = '1'
var type = 'recurrence'
var actionType = 'http'
var method = 'GET'
var workflowSchema = 'https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-
01/workflowdefinition.json#'

resource stg 'Microsoft.Logic/workflows@2019-05-01' = {
  name: logicAppName
  location: location
  tags: {
    displayName: logicAppName
  }
  properties: {
    definition: {
      '$schema': workflowSchema
      contentVersion: '1.0.0.0'
      parameters: {
        testUri: {
          type: 'string'
          defaultValue: testUri
        }
      }
      triggers: {
        recurrence: {
          type: type
          recurrence: {
            frequency: frequency
            interval: interval
          }
        }
      }
      actions: {
        actionType: {
          type: actionType
          inputs: {
            method: method
            uri: testUri
          }
        }
      }
    }
  }
}
```

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.

2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
logicAppName=<logic-name>
```

> **NOTE**
>
> Replace **<logic-name>** with the name of the logic app to create.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When you no longer need the logic app, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

Quickstart: Create Bicep files with Visual Studio Code

Quickstart: Create a Service Bus namespace and a queue using a Bicep file

9/21/2022 • 2 minutes to read • Edit Online

This article shows how to use a Bicep file that creates a Service Bus namespace and a queue within that namespace. The article explains how to specify which resources are deployed and how to define parameters that are specified when the deployment is executed. You can use this Bicep file for your own deployments, or customize it to meet your requirements.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Name of the Service Bus namespace')
param serviceBusNamespaceName string

@description('Name of the Queue')
param serviceBusQueueName string

@description('Location for all resources.')
param location string = resourceGroup().location

resource serviceBusNamespace 'Microsoft.ServiceBus/namespaces@2022-01-01-preview' = {
  name: serviceBusNamespaceName
  location: location
  sku: {
    name: 'Standard'
  }
  properties: {}
}

resource serviceBusQueue 'Microsoft.ServiceBus/namespaces/queues@2022-01-01-preview' = {
  parent: serviceBusNamespace
  name: serviceBusQueueName
  properties: {
    lockDuration: 'PT5M'
    maxSizeInMegabytes: 1024
    requiresDuplicateDetection: false
    requiresSession: false
    defaultMessageTimeToLive: 'P10675199DT2H48M5.4775807S'
    deadLetteringOnMessageExpiration: false
    duplicateDetectionHistoryTimeWindow: 'PT10M'
    maxDeliveryCount: 10
    autoDeleteOnIdle: 'P10675199DT2H48M5.4775807S'
    enablePartitioning: false
    enableExpress: false
  }
}
```

The resources defined in the Bicep file include:

- **Microsoft.ServiceBus/namespaces**
- **Microsoft.ServiceBus/namespaces/queues**

> **NOTE**
>
> The following ARM templates are available for download and deployment.
>
> - Create a Service Bus namespace with queue and authorization rule
> - Create a Service Bus namespace with topic and subscription
> - Create a Service Bus namespace
> - Create a Service Bus namespace with topic, subscription, and rule

You can find more Bicep/ARM templates from Azure Quickstart Templates

Deploy the Bicep file

With this Bicep file, you deploy a Service Bus namespace with a queue.

Service Bus queues offer First In, First Out (FIFO) message delivery to one or more competing consumers.

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

   ```
   az group create --name exampleRG --location eastus
   az deployment group create --resource-group exampleRG --template-file main.bicep
   ```

   You will be prompted to enter the following parameter values:

   - **serviceBusNamespaceName**: Name of the Service Bus namespace.
   - **serviceBusQueueName**: Name of the Queue.

   When the deployment finishes, you should see a message indicating the deployment succeeded.

Validate the deployment

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the VM and all of the resources in the resource group.

- CLI

- PowerShell

```
az group delete --name exampleRG
```

Next steps

See the following topic that shows how to create an authorization rule for the namespace/queue:

Create a Service Bus authorization rule for namespace and queue using an ARM template

Learn how to manage these resources by viewing these articles:

- Manage Service Bus with PowerShell
- Manage Service Bus resources with the Service Bus Explorer

Run Azure IoT Edge on Ubuntu Virtual Machines by using Bicep

9/21/2022 • 4 minutes to read • Edit Online

Applies to: ✅ IoT Edge 1.1 **Other versions:** IoT Edge 1.2, IoT Edge 1.3

Applies to: ✅ IoT Edge 1.2 ✅ IoT Edge 1.3 **Other versions:** IoT Edge 1.1

The Azure IoT Edge runtime is what turns a device into an IoT Edge device. The runtime can be deployed on devices as small as a Raspberry Pi or as large as an industrial server. Once a device is configured with the IoT Edge runtime, you can start deploying business logic to it from the cloud.

To learn more about how the IoT Edge runtime works and what components are included, see Understand the Azure IoT Edge runtime and its architecture.

This article lists the steps to deploy an Ubuntu 18.04 LTS virtual machine with the Azure IoT Edge runtime installed and configured using a pre-supplied device connection string. The deployment is accomplished using a cloud-init based Bicep file maintained in the iotedge-vm-deploy project repository.

On first boot, the virtual machine installs the latest version of the Azure IoT Edge runtime via cloud-init. It also sets a supplied connection string before the runtime starts, allowing you to easily configure and connect the IoT Edge device without the need to start an SSH or remote desktop session.

This article lists the steps to deploy an Ubuntu 20.04 LTS virtual machine with the Azure IoT Edge runtime installed and configured using a pre-supplied device connection string. The deployment is accomplished using a cloud-init based Bicep file maintained in the iotedge-vm-deploy project repository.

On first boot, the virtual machine installs the latest version of the Azure IoT Edge runtime via cloud-init. It also sets a supplied connection string before the runtime starts, allowing you to easily configure and connect the IoT Edge device without the need to start an SSH or remote desktop session.

Deploy from Azure CLI

You can't deploy a remote Bicep file. Save a copy of the Bicep file locally as **main.bicep**.

1. Ensure that you have installed the Azure CLI iot extension with:

```
az extension add --name azure-iot
```

2. Next, if you're using Azure CLI on your desktop, start by logging in:

```
az login
```

3. If you have multiple subscriptions, select the subscription you'd like to use:

   a. List your subscriptions:

```
az account list --output table
```

   b. Copy the SubscriptionID field for the subscription you'd like to use.

c. Set your working subscription with the ID that you copied:

```
az account set -s <SubscriptionId>
```

4. Create a new resource group (or specify an existing one in the next steps):

```
az group create --name IoTEdgeResources --location westus2
```

5. Create a new virtual machine:

   To use an **authenticationType** of `password`, see the example below:

```
az deployment group create \
--resource-group IoTEdgeResources \
--template-file "main.bicep" \
--parameters dnsLabelPrefix='my-edge-vm1' \
--parameters deviceConnectionString=$(az iot hub device-identity connection-string show --device-id
<REPLACE_WITH_DEVICE-NAME> --hub-name <REPLACE-WITH-HUB-NAME> -o tsv) \
--parameters authenticationType='password' \
--parameters adminUsername='<REPLACE_WITH_USERNAME>' \
--parameters adminPasswordOrKey="<REPLACE_WITH_SECRET_PASSWORD>"
```

   To authenticate with an SSH key, you may do so by specifying an **authenticationType** of `sshPublicKey`, then provide the value of the SSH key in the **adminPasswordOrKey** parameter. An example is shown below.

```
#Generate the SSH Key
ssh-keygen -m PEM -t rsa -b 4096 -q -f ~/.ssh/iotedge-vm-key -N ""

#Create a VM using the iotedge-vm-deploy script
az deployment group create \
--resource-group IoTEdgeResources \
--template-file "main.bicep" \
--parameters dnsLabelPrefix='my-edge-vm1' \
--parameters deviceConnectionString=$(az iot hub device-identity connection-string show --device-id
<REPLACE_WITH_DEVICE-NAME> --hub-name <REPLACE-WITH-HUB-NAME> -o tsv) \
--parameters authenticationType='sshPublicKey' \
--parameters adminUsername='<REPLACE_WITH_USERNAME>' \
--parameters adminPasswordOrKey="$(< ~/.ssh/iotedge-vm-key.pub)"
```

   To use an **authenticationType** of `password`, see the example below:

```
az deployment group create \
--resource-group IoTEdgeResources \
--template-file "main.bicep" \
--parameters dnsLabelPrefix='my-edge-vm1' \
--parameters deviceConnectionString=$(az iot hub device-identity connection-string show --device-id
<REPLACE_WITH_DEVICE-NAME> --hub-name <REPLACE-WITH-HUB-NAME> -o tsv) \
--parameters authenticationType='password' \
--parameters adminUsername='<REPLACE_WITH_USERNAME>' \
--parameters adminPasswordOrKey="<REPLACE_WITH_SECRET_PASSWORD>"
```

   To authenticate with an SSH key, you may do so by specifying an **authenticationType** of `sshPublicKey`, then provide the value of the SSH key in the **adminPasswordOrKey** parameter. An example is shown below.

```
#Generate the SSH Key
ssh-keygen -m PEM -t rsa -b 4096 -q -f ~/.ssh/iotedge-vm-key -N ""

#Create a VM using the iotedge-vm-deploy script
az deployment group create \
--resource-group IoTEdgeResources \
--template-file "main.bicep" \
--parameters dnsLabelPrefix='my-edge-vm1' \
--parameters deviceConnectionString=$(az iot hub device-identity connection-string show --device-id
<REPLACE_WITH_DEVICE-NAME> --hub-name <REPLACE-WITH-HUB-NAME> -o tsv) \
--parameters authenticationType='sshPublicKey' \
--parameters adminUsername='<REPLACE_WITH_USERNAME>' \
--parameters adminPasswordOrKey="$(< ~/.ssh/iotedge-vm-key.pub)"
```

6. Verify that the deployment has completed successfully. A virtual machine resource should have been deployed into the selected resource group. Take note of the machine name, this should be in the format `vm-0000000000000`. Also, take note of the associated **DNS Name**, which should be in the format `<dnsLabelPrefix>`.`<location>`.cloudapp.azure.com.

   The **DNS Name** can be obtained from the JSON-formatted output of the previous step, within the **outputs** section as part of the **public SSH** entry. The value of this entry can be used to SSH into to the newly deployed machine.

```
"outputs": {
  "public SSH": {
    "type": "String",
    "value": "ssh <adminUsername>@<DNS_Name>"
  }
}
```

   The **DNS Name** can also be obtained from the **Overview** section of the newly deployed virtual machine within the Azure portal.

7. If you want to SSH into this VM after setup, use the associated **DNS Name** with the command:
   `ssh <adminUsername>@<DNS_Name>`

Next steps

Now that you have an IoT Edge device provisioned with the runtime installed, you can deploy IoT Edge modules.

If you are having problems with the IoT Edge runtime installing properly, check out the troubleshooting page.

To update an existing installation to the newest version of IoT Edge, see Update the IoT Edge security daemon and runtime.

If you'd like to open up ports to access the VM through SSH or other inbound connections, refer to the Azure Virtual Machines documentation on opening up ports and endpoints to a Linux VM

Quickstart: Deploy an Azure IoT Hub and a storage account using Bicep

9/21/2022 • 6 minutes to read • Edit Online

In this quickstart, you use Bicep to create an IoT Hub that will route messages to Azure Storage and a storage account to hold the messages. After manually adding a virtual IoT device to the hub to submit the messages, you configure that connection information in an application called *arm-read-write* to submit messages from the device to the hub. The hub is configured so the messages sent to the hub are automatically routed to the storage account. At the end of this quickstart, you can open the storage account and see the messages sent.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is called `101-iothub-auto-route-messages` from Azure Quickstart Templates.

```
@description('Define the project name or prefix for all objects.')
@minLength(1)
@maxLength(11)
param projectName string = 'contoso'

@description('The datacenter to use for the deployment.')
param location string = resourceGroup().location

@description('The SKU to use for the IoT Hub.')
param skuName string = 'S1'

@description('The number of IoT Hub units.')
param skuUnits int = 1

@description('Partitions used for the event stream.')
param d2cPartitions int = 4

var iotHubName = '${projectName}Hub${uniqueString(resourceGroup().id)}'
var storageAccountName = '${toLower(projectName)}${uniqueString(resourceGroup().id)}'
var storageEndpoint = '${projectName}StorageEndpont'
var storageContainerName = '${toLower(projectName)}results'

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-08-01' = {
  name: storageAccountName
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'Storage'
}

resource container 'Microsoft.Storage/storageAccounts/blobServices/containers@2021-08-01' = {
  name: '${storageAccountName}/default/${storageContainerName}'
  properties: {
    publicAccess: 'None'
```

```
  }
  dependsOn: [
    storageAccount
  ]
}

resource IoTHub 'Microsoft.Devices/IotHubs@2021-07-02' = {
  name: iotHubName
  location: location
  sku: {
    name: skuName
    capacity: skuUnits
  }
  properties: {
    eventHubEndpoints: {
      events: {
        retentionTimeInDays: 1
        partitionCount: d2cPartitions
      }
    }
    routing: {
      endpoints: {
        storageContainers: [
          {
            connectionString:
'DefaultEndpointsProtocol=https;AccountName=${storageAccountName};EndpointSuffix=${environment().suffixes.st
orage};AccountKey=${storageAccount.listKeys().keys[0].value}'
            containerName: storageContainerName
            fileNameFormat: '{iothub}/{partition}/{YYYY}/{MM}/{DD}/{HH}/{mm}'
            batchFrequencyInSeconds: 100
            maxChunkSizeInBytes: 104857600
            encoding: 'JSON'
            name: storageEndpoint
          }
        ]
      }
      routes: [
        {
          name: 'ContosoStorageRoute'
          source: 'DeviceMessages'
          condition: 'level="storage"'
          endpointNames: [
            storageEndpoint
          ]
          isEnabled: true
        }
      ]
      fallbackRoute: {
        name: '$fallback'
        source: 'DeviceMessages'
        condition: 'true'
        endpointNames: [
          'events'
        ]
        isEnabled: true
      }
    }
    messagingEndpoints: {
      fileNotifications: {
        lockDurationAsIso8601: 'PT1M'
        ttlAsIso8601: 'PT1H'
        maxDeliveryCount: 10
      }
    }
    enableFileUploadNotifications: false
    cloudToDevice: {
      maxDeliveryCount: 10
      defaultTtlAsIso8601: 'PT1H'
      feedback: {
```

```
        lockDurationAsIso8601: 'PT1M'
        ttlAsIso8601: 'PT1H'
        maxDeliveryCount: 10
      }
    }
  }
}
```

Two Azure resources are defined in the Bicep file:

- Microsoft.Storage/storageAccounts
- Microsoft.Devices/IotHubs

Deploy the Bicep file and run the sample app

This section provides the steps to deploy the Bicep file, create a virtual device, and run the arm-read-write application to send the messages.

1. Create the resources by deploying the Bicep file using Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

   ```
   az group create --name ContosoResourceGrp --location eastus
   az deployment group create --resource-group exampleRG --template-file main.bicep
   ```

   When the deployment finishes, you should see a message indicating the deployment succeeded.

2. Download and unzip the IoT C# SDK.

3. Open a command window and go to the folder where you unzipped the IoT C# SDK. Find the folder with the arm-read-write.csproj file. You create the environment variables in this command window. Log into the Azure portal to get the keys. Select **Resource Groups** then select the resource group used for this quickstart.

4. You see the IoT Hub and storage account that were created when you deployed the Bicep file. Wait until the file is fully deployed before continuing. Then select your resource group to see your resources.

5. You need the **hub name**. Select the hub in the list of resources. Copy the name of the hub from the top of the IoT Hub section to the Windows clipboard.

   Substitute the hub name in this command where noted, and execute this command in the command window:

```
SET IOT_HUB_URI=<hub name goes here>.azure-devices-net;
```

   which will look this example:

```
SET IOT_HUB_URI=ContosoTestHubdlxlud5h.azure-devices-net;
```

6. The next environment variable is the IoT Device Key. Add a new device to the hub by selecting **Devices** from the IoT Hub menu for the hub.

7. On the right side of the screen, select **+ Add Device** to add a new device.

   Fill in the new device name. This quickstart uses a name starting with **Contoso-Test-Device**. Save the device and then open that screen again to retrieve the device key. (The key is generated for you when you close the pane.) Select either the primary or secondary key and copy it to the Windows clipboard. In the command window, set the command to execute and then press **Enter**. The command should look like this

one but with the device key pasted in:

```
SET IOT_DEVICE_KEY=<device-key-goes-here>
```

8. The last environment variable is the **Device ID**. In the command window, set up the command and execute it.

```
SET IOT_DEVICE_ID=<device-id-goes-here>
```

which will look like this example:

```
SET IOT_DEVICE_ID=Contoso-Test-Device
```

9. To see the environment variables you've defined, type SET on the command line and press **Enter**, then look for the ones starting with **IoT**.

```
IOT_DEVICE_ID=Contoso-Test-Device
IOT_DEVICE_KEY=RClD0LGxZCYav
IOT_HUB_URI=ContosoTestHubdlxlud5h.azure-devices-net
```

Now the environment variables are set, run the application from the same command window. Because you're using the same window, the variables will be accessible in memory when you run the application.

10. To run the application, type the following command in the command window and press **Enter**.

```
dotnet run arm-read-write
```

The application generates and displays messages on the console as it sends each message to the IoT hub. The hub was configured in the Bicep file to have automated routing. Messages containing the text `level = storage` are automatically routed to the storage account. Let the app run for 10 to 15 minutes, then press **Enter** once or twice until it stops running.

Review deployed resources

1. Log in to the Azure portal and select the Resource Group, then select the storage account.

2. Drill down into the storage account until you find files.

3. Select one of the files and select **Download** and download the file to a location you can find later. It will have a name that's numeric, like 47. Add *.txt* to the end and then double-click on the file to open it.

4. When you open the file, each row is for a different message. The body of each message is also encrypted. It must be in order for you to perform queries against the body of the message.


```
{"EnqueuedTimeUtc":"2020-08-13T04:47:20.2810000Z","Properties":{"level":"normal"},"SystemProperties":{"connectionDevice
AAYQAAAGwAAAAgAAAAbQAAAGUAAABzAAAAcwAAAGEAAABnAAAAZQAAAC4AAAAiAAAAfQAAAA==}
{"EnqueuedTimeUtc":"2020-08-13T04:47:21.3290000Z","Properties":{"level":"critical"},"SystemProperties":{"connectionDevi
AAAAaQAAAGMAAABhAAAAbAAAACAAAABtAAAAZQAAAHMAAABzAAAAYQAAAGcAAABlAAAALgAAACIAAAB9AAAA"}
{"EnqueuedTimeUtc":"2020-08-13T04:47:22.3760000Z","Properties":{"level":"critical"},"SystemProperties":{"connectionDevi
AAAAaQAAAGMAAABhAAAAbAAAACAAAABtAAAAZQAAAHMAAABzAAAAYQAAAGcAAABlAAAALgAAACIAAAB9AAAA"}
{"EnqueuedTimeUtc":"2020-08-13T04:47:23.4390000Z","Properties":{"level":"normal"},"SystemProperties":{"connectionDevice
AAbAAAACAAAABtAAAAZQAAAHMAAABzAAAAYQAAAGcAAABlAAAALgAAACIAAAB9AAAA"}
{"EnqueuedTimeUtc":"2020-08-13T04:47:24.4860000Z","Properties":{"level":"normal"},"SystemProperties":{"connectionDevice
AAYQAAAGwAAAAgAAAAbQAAAGUAAABzAAAAcwAAAGEAAABnAAAAZQAAAC4AAAAiAAAAfQAAAA==}
{"EnqueuedTimeUtc":"2020-08-13T04:47:25.5330000Z","Properties":{"level":"normal"},"SystemProperties":{"connectionDevice
AAYQAAAGwAAAAgAAAAbQAAAGUAAABzAAAAcwAAAGEAAABnAAAAZQAAAC4AAAAiAAAAfQAAAA==}
{"EnqueuedTimeUtc":"2020-08-13T04:47:26.5960000Z","Properties":{"level":"critical"},"SystemProperties":{"connectionDevi
AAAAYQAAAGwAAAAgAAAAbQAAAGUAAABzAAAAcwAAAGEAAABnAAAAZQAAAC4AAAAiAAAAfQAAAA==}
```


> **NOTE**
>
> These messages are encoded in UTF-32 and base64. If you read the message back, you have to decode it from base64 and utf-32 in order to read it as ASCII. If you're interested, you can use the method ReadOneRowFromFile in the Routing Tutorial to read one for from one of these message files and decode it into ASCII. ReadOneRowFromFile is in the IoT C# SDK repository that you unzipped for this quickstart. Here is the path from the top of that folder: *./iothub/device/samples/getting started/RoutingTutorial/SimulatedDevice/Program.cs* Set the boolean `readTheFile` to true, and hardcode the path to the file on disk, and it will open and translate the first row in the file.

You have deployed a Bicep file to create an IoT Hub and a storage account, and run a program to send messages to the hub. The messages are then automatically stored in the storage account where they can be viewed.

Clean up resources

When you no longer need the resources that you created, delete the resource group.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

Quickstart: Create Bicep files with Visual Studio Code

Quickstart: Set up the IoT Hub Device Provisioning Service (DPS) with Bicep

9/21/2022 • 5 minutes to read • Edit Online

You can use a Bicep file to programmatically set up the Azure cloud resources necessary for provisioning your devices. These steps show how to create an IoT hub and a new IoT Hub Device Provisioning Service instance with a Bicep file. The IoT Hub is also linked to the DPS resource using the Bicep file. This linking allows the DPS resource to assign devices to the hub based on allocation policies you configure.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

This quickstart uses Azure PowerShell, and the Azure CLI to perform the programmatic steps necessary to create a resource group and deploy the Bicep file, but you can easily use .NET, Ruby, or other programming languages to perform these steps and deploy your Bicep file.

Prerequisites

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

You can use either the Azure Cloud Shell or a local Azure CLI.

- Azure Cloud Shell with the Bash environment. Or launch the Cloud Shell here.

  Launch Cloud Shell

- Local Azure CLI, see how to install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - Sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you first use Azure CLI, install the Azure CLI extension. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.
- If you choose to use Azure PowerShell locally:
  - Install the Az PowerShell module.
  - Connect to your Azure account using the Connect-AzAccount cmdlet.
- If you choose to use Azure Cloud Shell:
  - See Overview of Azure Cloud Shell for more information.

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

> **NOTE**
> Currently there is no Bicep file support for creating enrollments with new DPS resources. This is a common and understood request that is being considered for implementation.

```
@description('Specify the name of the Iot hub.')
param iotHubName string

@description('Specify the name of the provisioning service.')
param provisioningServiceName string

@description('Specify the location of the resources.')
param location string = resourceGroup().location

@description('The SKU to use for the IoT Hub.')
param skuName string = 'S1'

@description('The number of IoT Hub units.')
param skuUnits int = 1

var iotHubKey = 'iothubowner'

resource iotHub 'Microsoft.Devices/IotHubs@2021-07-02' = {
  name: iotHubName
  location: location
  sku: {
    name: skuName
    capacity: skuUnits
  }
  properties: {}
}

resource provisioningService 'Microsoft.Devices/provisioningServices@2022-02-05' = {
  name: provisioningServiceName
  location: location
  sku: {
    name: skuName
    capacity: skuUnits
  }
  properties: {
    iotHubs: [
      {
        connectionString:
'HostName=${iotHub.properties.hostName};SharedAccessKeyName=${iotHubKey};SharedAccessKey=${iotHub.listkeys()
.value}'
        location: location
      }
    ]
  }
}
```

Two Azure resources are defined in the Bicep file above:

- **Microsoft.Devices/iothubs**: Creates a new Azure IoT Hub.
- **Microsoft.Devices/provisioningservices**: Creates a new Azure IoT Hub Device Provisioning Service with the new IoT Hub already linked to it.

Save a copy of the Bicep file locally as **main.bicep**.

Deploy the Bicep file

Sign in to your Azure account and select your subscription.

1. To log in Azure at the command prompt:

   - CLI
   - PowerShell

   ```
   az login
   ```

   Follow the instructions to authenticate using the code and sign in to your Azure account through a web browser.

2. If you have multiple Azure subscriptions, signing in to Azure grants you access to all the Azure accounts associated with your credentials.

   - CLI
   - PowerShell

   ```
   az account list -o table
   ```

   Use the following command to select the subscription that you want to use to run the commands to create your IoT hub and DPS resources. You can use either the subscription name or ID from the output of the previous command:

   - CLI
   - PowerShell

   ```
   az account set --subscription {your subscription name or id}
   ```

3. Deploy the Bicep file with the following commands.

   > **TIP**
   > The commands will prompt for a resource group location. You can view a list of available locations by first running the command:
   >
   > - CLI
   > - PowerShell
   >
   > `az account list-locations -o table`

   - CLI
   - PowerShell

   ```
   az group create --name exampleRG --location eastus
   az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
   iotHubName={IoT-Hub-name} provisioningServiceName={DPS-name}
   ```

   Replace **{IoT-Hub-name}** with a globally unique IoT Hub name, replace **{DPS-name}** with a globally unique Device Provisioning Service (DPS) resource name.

   It takes a few moments to create the resources.

Review deployed resources

1. To verify the deployment, run the following command and look for the new provisioning service and IoT hub in the output:

   - CLI
   - PowerShell

   ```
   az resource list -g exampleRg
   ```

2. To verify that the hub is already linked to the DPS resource, run the following command.

   - CLI
   - PowerShell

   ```
   az iot dps show --name <Your provisioningServiceName>
   ```

Clean up resources

Other quickstarts in this collection build upon this quickstart. If you plan to continue on to work with subsequent quickstarts or with the tutorials, do not clean up the resources created in this quickstart. If you do not plan to continue, you can use the Azure PowerShell or Azure CLI to delete the resource group and all of its resources.

To delete a resource group and all its resources from the Azure portal, just open the resource group and click **Delete resource group** and the top.

To delete the resource group deployed:

- CLI
- PowerShell

```
az group delete --name exampleRG
```

You can also delete resource groups and individual resources using the Azure portal, PowerShell, or REST APIs, as well as with supported platform SDKs published for Azure Resource Manager or IoT Hub Device Provisioning Service.

Next steps

In this quickstart, you've deployed an IoT hub and a Device Provisioning Service instance, and linked the two resources. To learn how to use this setup to provision a device, continue to the quickstart for creating a device.

Quickstart: Provision a simulated symmetric key device

Quickstart: Create Azure Advisor alerts on new recommendations using Bicep

9/21/2022 • 2 minutes to read • Edit Online

This article shows you how to set up an alert for new recommendations from Azure Advisor using Bicep.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Whenever Azure Advisor detects a new recommendation for one of your resources, an event is stored in Azure Activity log. You can set up alerts for these events from Azure Advisor using a recommendation-specific alerts creation experience. You can select a subscription and optionally select a resource group to specify the resources that you want to receive alerts on.

You can also determine the types of recommendations by using these properties:

- Category
- Impact level
- Recommendation type

You can also configure the action that will take place when an alert is triggered by:

- Selecting an existing action group
- Creating a new action group

To learn more about action groups, see Create and manage action groups.

> **NOTE**
> Advisor alerts are currently only available for High Availability, Performance, and Cost recommendations. Security recommendations are not supported.

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin.
- To run the commands from your local computer, install Azure CLI or the Azure PowerShell modules. For more information, see Install the Azure CLI and Install Azure PowerShell.

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Specify the name of alert')
param alertName string

@description('Specify a description of alert')
@allowed([
  'Active'
  'InProgress'
  'Resolved'
])
param status string = 'Active'

@description('Specify the email address where the alerts are sent to.')
param emailAddress string = 'email@example.com'

@description('Specify the email address name where the alerts are sent to.')
param emailName string = 'Example'

resource emailActionGroup 'microsoft.insights/actionGroups@2021-09-01' = {
  name: 'emailActionGroupName'
  location: 'global'
  properties: {
    groupShortName: 'string'
    enabled: true
    emailReceivers: [
      {
        name: emailName
        emailAddress: emailAddress
        useCommonAlertSchema: true
      }
    ]
  }
}

resource alert 'Microsoft.Insights/activityLogAlerts@2020-10-01' = {
  name: alertName
  location: 'global'
  properties: {
    enabled: true
    scopes: [
      subscription().id
    ]
    condition: {
      allOf: [
        {
          field: 'category'
          equals: 'ResourceHealth'
        }
        {
          field: 'status'
          equals: status
        }
      ]
    }
    actions: {
      actionGroups: [
        {
          actionGroupId: emailActionGroup.id
        }
      ]
    }
  }
}
```

The Bicep file defines two resources:

- Microsoft.Insights/actionGroups

- Microsoft.Insights/activityLogAlerts

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
alertName=<alert-name>
```

> **NOTE**
>
> Replace **<alert-name>** with the name of the alert.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Validate the deployment

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

- Get an overview of activity log alerts, and learn how to receive alerts.
- Learn more about action groups.

Back up a virtual machine in Azure with a Bicep template

9/21/2022 • 5 minutes to read • Edit Online

Azure Backup allows you to back up your Azure VM using multiple options - such as Azure portal, PowerShell, CLI, Azure Resource Manager, Bicep, and so on. This article describes how to back up an Azure VM with an Azure Bicep template and Azure PowerShell. This quickstart focuses on the process of deploying a Bicep template to create a Recovery Services vault. For more information on developing Bicep templates, see the Bicep documentation and the template reference.

Bicep is a language for declaratively deploying Azure resources. You can use Bicep instead of JSON to develop your Azure Resource Manager templates (ARM templates). Bicep syntax reduces the complexity and improves the development experience. Bicep is a transparent abstraction over ARM template JSON that provides all JSON template capabilities. During deployment, the Bicep CLI converts a Bicep file into an ARM template JSON. A Bicep file states the Azure resources and resource properties, without writing a sequence of programming commands to create resources.

Resource types, API versions, and properties that are valid in an ARM template, are also valid in a Bicep file.

Prerequisites

To set up your environment for Bicep development, see Install Bicep tools.

> **NOTE**
>
> Install the latest Azure PowerShell module and the Bicep CLI as detailed in article.

Review the template

The template used below is from Azure quickstart templates. This template allows you to deploy simple Windows VM and Recovery Services vault configured with *DefaultPolicy* for *Protection*.

```
@description('Specifies a name for generating resource names.')
@maxLength(8)
param projectName string

@description('Specifies the location for all resources.')
param location string = resourceGroup().location

@description('Specifies the administrator username for the Virtual Machine.')
param adminUsername string

@description('Specifies the administrator password for the Virtual Machine.')
@secure()
param adminPassword string

@description('Specifies the unique DNS Name for the Public IP used to access the Virtual Machine.')
param dnsLabelPrefix string

@description('Virtual machine size.')
param vmSize string = 'Standard_A2'

@description('Specifies the Windows version for the VM. This will pick a fully patched image of this given
Windows version.')
```

```
windows version. )
@allowed([
  '2008-R2-SP1'
  '2012-Datacenter'
  '2012-R2-Datacenter'
  '2016-Nano-Server'
  '2016-Datacenter-with-Containers'
  '2016-Datacenter'
  '2019-Datacenter'
  '2019-Datacenter-Core'
  '2019-Datacenter-Core-smalldisk'
  '2019-Datacenter-Core-with-Containers'
  '2019-Datacenter-Core-with-Containers-smalldisk'
  '2019-Datacenter-smalldisk'
  '2019-Datacenter-with-Containers'
  '2019-Datacenter-with-Containers-smalldisk'
])
param windowsOSVersion string = '2016-Datacenter'

var storageAccountName = '${projectName}store'
var networkInterfaceName = '${projectName}-nic'
var vNetAddressPrefix = '10.0.0.0/16'
var vNetSubnetName = 'default'
var vNetSubnetAddressPrefix = '10.0.0.0/24'
var publicIPAddressName = '${projectName}-ip'
var vmName = '${projectName}-vm'
var vNetName = '${projectName}-vnet'
var vaultName = '${projectName}-vault'
var backupFabric = 'Azure'
var backupPolicyName = 'DefaultPolicy'
var protectionContainer = 'iaasvmcontainer;iaasvmcontainerv2;${resourceGroup().name};${vmName}'
var protectedItem = 'vm;iaasvmcontainerv2;${resourceGroup().name};${vmName}'
var networkSecurityGroupName = 'default-NSG'

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-06-01' = {
  name: storageAccountName
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'Storage'
  properties: {}
}

resource publicIPAddress 'Microsoft.Network/publicIPAddresses@2020-06-01' = {
  name: publicIPAddressName
  location: location
  properties: {
    publicIPAllocationMethod: 'Dynamic'
    dnsSettings: {
      domainNameLabel: dnsLabelPrefix
    }
  }
}

resource networkSecurityGroup 'Microsoft.Network/networkSecurityGroups@2020-06-01' = {
  name: networkSecurityGroupName
  location: location
  properties: {
    securityRules: [
      {
        name: 'default-allow-3389'
        properties: {
          priority: 1000
          access: 'Allow'
          direction: 'Inbound'
          destinationPortRange: '3389'
          protocol: 'Tcp'
          sourceAddressPrefix: '*'
          sourcePortRange: '*'
```

```
          sourcePortRange: '*'
          destinationAddressPrefix: '*'
        }
      }
    ]
  }
}

resource vNet 'Microsoft.Network/virtualNetworks@2020-06-01' = {
  name: vNetName
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        vNetAddressPrefix
      ]
    }
    subnets: [
      {
        name: vNetSubnetName
        properties: {
          addressPrefix: vNetSubnetAddressPrefix
          networkSecurityGroup: {
            id: networkSecurityGroup.id
          }
        }
      }
    ]
  }
}

resource networkInterface 'Microsoft.Network/networkInterfaces@2020-06-01' = {
  name: networkInterfaceName
  location: location
  properties: {
    ipConfigurations: [
      {
        name: 'ipconfig1'
        properties: {
          privateIPAllocationMethod: 'Dynamic'
          publicIPAddress: {
            id: publicIPAddress.id
          }
          subnet: {
            id: '${vNet.id}/subnets/${vNetSubnetName}'
          }
        }
      }
    ]
  }
}

resource virtualMachine 'Microsoft.Compute/virtualMachines@2020-06-01' = {
  name: vmName
  location: location
  properties: {
    hardwareProfile: {
      vmSize: vmSize
    }
    osProfile: {
      computerName: vmName
      adminUsername: adminUsername
      adminPassword: adminPassword
    }
    storageProfile: {
      imageReference: {
        publisher: 'MicrosoftWindowsServer'
        offer: 'WindowsServer'
        sku: windowsOSVersion
```

```
        version: 'latest'
      }
      osDisk: {
        createOption: 'FromImage'
      }
      dataDisks: [
        {
          diskSizeGB: 1023
          lun: 0
          createOption: 'Empty'
        }
      ]
    }
    networkProfile: {
      networkInterfaces: [
        {
          id: networkInterface.id
        }
      ]
    }
    diagnosticsProfile: {
      bootDiagnostics: {
        enabled: true
        storageUri: storageAccount.properties.primaryEndpoints.blob
      }
    }
  }
}

resource recoveryServicesVault 'Microsoft.RecoveryServices/vaults@2020-02-02' = {
  name: vaultName
  location: location
  sku: {
    name: 'RS0'
    tier: 'Standard'
  }
  properties: {}
}

resource vaultName_backupFabric_protectionContainer_protectedItem
'Microsoft.RecoveryServices/vaults/backupFabrics/protectionContainers/protectedItems@2020-02-02' = {
  name: '${vaultName}/${backupFabric}/${protectionContainer}/${protectedItem}'
  properties: {
    protectedItemType: 'Microsoft.Compute/virtualMachines'
    policyId: '${recoveryServicesVault.id}/backupPolicies/${backupPolicyName}'
    sourceResourceId: virtualMachine.id
  }
}
```

The resources defined in the above template are:

- Microsoft.Storage/storageAccounts
- Microsoft.Network/publicIPAddresses
- Microsoft.Network/networkSecurityGroups
- Microsoft.Network/virtualNetworks
- Microsoft.Network/networkInterfaces
- Microsoft.Compute/virtualMachines
- Microsoft.RecoveryServices/vaults
- Microsoft.RecoveryServices/vaults/backupFabrics/protectionContainers/protectedItems

Deploy the template

To deploy the template, select *Try it* to open the Azure Cloud Shell, and then paste the following PowerShell script in the shell window. To paste the code, right-click the shell window and then select **Paste**.

```
$projectName = Read-Host -Prompt "Enter a project name (limited to eight characters) that is used to
generate Azure resource names"
$location = Read-Host -Prompt "Enter the location (for example, centralus)"
$adminUsername = Read-Host -Prompt "Enter the administrator username for the virtual machine"
$adminPassword = Read-Host -Prompt "Enter the administrator password for the virtual machine" -
AsSecureString
$dnsPrefix = Read-Host -Prompt "Enter the unique DNS Name for the Public IP used to access the virtual
machine"

$resourceGroupName = "${projectName}rg"
$templateUri = "https://raw.githubusercontent.com/Azure/azure-quickstart-
templates/master/quickstarts/microsoft.recoveryservices/recovery-services-create-vm-and-configure-
backup/main.bicep"

New-AzResourceGroup -Name $resourceGroupName -Location $location
New-AzResourceGroupDeployment -ResourceGroupName $resourceGroupName -TemplateUri $templateUri -projectName
$projectName -adminUsername $adminUsername -adminPassword $adminPassword -dnsLabelPrefix $dnsPrefix
```

Validate the deployment

Start a backup job

The template creates a VM and enables backup on the VM. After you deploy the template, you need to start a backup job. For more information, see Start a backup job.

Monitor the backup job

To monitor the backup job, see Monitor the backup job.

Clean up resources

- If you no longer need to back up the VM, you can clean it up.
- To try out restoring the VM, skip the clean-up process.
- If you've used an existing VM, you can skip the final Remove-AzResourceGroup cmdlet to keep the resource group and VM.

Follow these steps:

1. Disable protection, remove the restore points and vault.

2. Delete the resource group and associated VM resources, as follows:

```
Disable-AzRecoveryServicesBackupProtection -Item $item -RemoveRecoveryPoints
$vault = Get-AzRecoveryServicesVault -Name "myRecoveryServicesVault"
Remove-AzRecoveryServicesVault -Vault $vault
Remove-AzResourceGroup -Name "myResourceGroup"
```

Next steps

In this quickstart, you created a Recovery Services vault, enabled protection on a VM, and created the initial recovery point.

- Learn how to back up VMs in the Azure portal.
- Learn how to quickly restore a VM

- Learn how to create Bicep templates.
- Learn how to decompile Azure Resource Manager templates (ARM templates) to Bicep files.

Quickstart: Create a budget with Bicep

9/21/2022 • 10 minutes to read • Edit Online

Budgets in Cost Management help you plan for and drive organizational accountability. With budgets, you can account for the Azure services you consume or subscribe to during a specific period. They help you inform others about their spending to proactively manage costs and monitor how spending progresses over time. When the budget thresholds you've created are exceeded, notifications are triggered. None of your resources are affected and your consumption isn't stopped. You can use budgets to compare and track spending as you analyze costs. This quickstart shows you how to create a budget named 'MyBudget' using Bicep.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

If you have a new subscription, you can't immediately create a budget or use other Cost Management features. It might take up to 48 hours before you can use all Cost Management features.

Budgets are supported for the following types of Azure account types and scopes:

- Azure role-based access control (Azure RBAC) scopes
  - Management groups
  - Subscription
- Enterprise Agreement scopes
  - Billing account
  - Department
  - Enrollment account
- Individual agreements
  - Billing account
- Microsoft Customer Agreement scopes
  - Billing account
  - Billing profile
  - Invoice section
  - Customer
- AWS scopes
  - External account
  - External subscription

To view budgets, you need at least read access for your Azure account.

For Azure EA subscriptions, you must have read access to view budgets. To create and manage budgets, you must have contributor permission.

The following Azure permissions, or scopes, are supported per subscription for budgets by user and group. For more information about scopes, see Understand and work with scopes.

- Owner: Can create, modify, or delete budgets for a subscription.

- Contributor and Cost Management contributor: Can create, modify, or delete their own budgets. Can modify the budget amount for budgets created by others.
- Reader and Cost Management reader: Can view budgets that they have permission to.

For more information about assigning permission to Cost Management data, see Assign access to Cost Management data.

No filter

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
targetScope = 'subscription'

@description('Name of the Budget. It should be unique within a resource group.')
param budgetName string = 'MyBudget'

@description('The total amount of cost or usage to track with the budget')
param amount int = 1000

@description('The time covered by a budget. Tracking of the amount will be reset based on the time grain.')
@allowed([
  'Monthly'
  'Quarterly'
  'Annually'
])
param timeGrain string = 'Monthly'

@description('The start date must be first of the month in YYYY-MM-DD format. Future start date should not
be more than three months. Past start date should be selected within the timegrain preiod.')
param startDate string

@description('The end date for the budget in YYYY-MM-DD format. If not provided, we default this to 10 years
from the start date.')
param endDate string

@description('Threshold value associated with a notification. Notification is sent when the cost exceeded
the threshold. It is always percent and has to be between 0.01 and 1000.')
param firstThreshold int = 90

@description('Threshold value associated with a notification. Notification is sent when the cost exceeded
the threshold. It is always percent and has to be between 0.01 and 1000.')
param secondThreshold int = 110

@description('The list of email addresses to send the budget notification to when the threshold is
exceeded.')
param contactEmails array

resource budget 'Microsoft.Consumption/budgets@2021-10-01' = {
  name: budgetName
  properties: {
    timePeriod: {
      startDate: startDate
      endDate: endDate
    }
    timeGrain: timeGrain
    amount: amount
    category: 'Cost'
    notifications: {
      NotificationForExceededBudget1: {
        enabled: true
        operator: 'GreaterThan'
        threshold: firstThreshold
        contactEmails: contactEmails
      }
      NotificationForExceededBudget2: {
        enabled: true
        operator: 'GreaterThan'
        threshold: secondThreshold
        contactEmails: contactEmails
      }
    }
  }
}
```

One Azure resource is defined in the Bicep file:

- Microsoft.Consumption/budgets: Create an Azure budget.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
myContactEmails ='("user1@contoso.com", "user2@contoso.com")'

az deployment sub create --name demoSubDeployment --location centralus --template-file main.bicep --
parameters startDate=<start-date> endDate=<end-date> contactEmails=$myContactEmails
```

You need to enter the following parameters:

- **startDate**: Replace **<start-date>** with the start date. It must be the first of the month in YYYY-MM-DD format. A future start date shouldn't be more than three months in the future. A past start date should be selected within the timegrain period.
- **endDate**: Replace **<end-date>** with the end date in YYYY-MM-DD format. If not provided, it defaults to ten years from the start date.
- **contactEmails**: First create a variable that holds your emails and then pass that variable. Replace the sample emails with the email addresses to send the budget notification to when the threshold is exceeded.

> **NOTE**
> When the deployment finishes, you should see a message indicating the deployment succeeded.

One filter

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
targetScope = 'subscription'

@description('Name of the Budget. It should be unique within a resource group.')
param budgetName string = 'MyBudget'

@description('The total amount of cost or usage to track with the budget')
param amount int = 1000

@description('The time covered by a budget. Tracking of the amount will be reset based on the time grain.')
@allowed([
  'Monthly'
  'Quarterly'
  'Annually'
])
param timeGrain string = 'Monthly'

@description('The start date must be first of the month in YYYY-MM-DD format. Future start date should not
be more than three months. Past start date should be selected within the timegrain preiod.')
param startDate string

@description('The end date for the budget in YYYY-MM-DD format. If not provided, we default this to 10 years
from the start date.')
param endDate string

@description('Threshold value associated with a notification. Notification is sent when the cost exceeded
```

```
the threshold. It is always percent and has to be between 0.01 and 1000.')
param firstThreshold int = 90

@description('Threshold value associated with a notification. Notification is sent when the cost exceeded
the threshold. It is always percent and has to be between 0.01 and 1000.')
param secondThreshold int = 110

@description('The list of email addresses to send the budget notification to when the threshold is
exceeded.')
param contactEmails array

@description('The set of values for the resource group filter.')
param resourceGroupFilterValues array

resource budget 'Microsoft.Consumption/budgets@2021-10-01' = {
  name: budgetName
  properties: {
    timePeriod: {
      startDate: startDate
      endDate: endDate
    }
    timeGrain: timeGrain
    amount: amount
    category: 'Cost'
    notifications: {
      NotificationForExceededBudget1: {
        enabled: true
        operator: 'GreaterThan'
        threshold: firstThreshold
        contactEmails: contactEmails
      }
      NotificationForExceededBudget2: {
        enabled: true
        operator: 'GreaterThan'
        threshold: secondThreshold
        contactEmails: contactEmails
      }
    }
    filter: {
      dimensions: {
        name: 'ResourceGroupName'
        operator: 'In'
        values: resourceGroupFilterValues
      }
    }
  }
}
```

One Azure resource is defined in the Bicep file:

- Microsoft.Consumption/budgets: Create an Azure budget.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.
   - CLI
   - PowerShell

```
myContactEmails ='("user1@contoso.com", "user2@contoso.com")'
myRgFilterValues ='("resource-group-01", "resource-group-02")'

az deployment sub create --name demoSubDeployment --location centralus --template-file main.bicep --
parameters startDate=<start-date> endDate=<end-date> contactEmails=$myContactEmails
resourceGroupFilterValues=$myRgFilterValues
```

You need to enter the following parameters:

- **startDate**: Replace **<start-date>** with the start date. It must be the first of the month in YYYY-MM-DD format. A future start date shouldn't be more than three months in the future. A past start date should be selected within the timegrain period.
- **endDate**: Replace **<end-date>** with the end date in YYYY-MM-DD format. If not provided, it defaults to ten years from the start date.
- **contactEmails**: First create a variable that holds your emails and then pass that variable. Replace the sample emails with the email addresses to send the budget notification to when the threshold is exceeded.
- **resourceGroupFilterValues** First create a variable that holds your resource group filter values and then pass that variable. Replace the sample filter values with the set of values for your resource group filter.

> **NOTE**
> When the deployment finishes, you should see a message indicating the deployment succeeded.

Two or more filters

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
targetScope = 'subscription'

@description('Name of the Budget. It should be unique within a resource group.')
param budgetName string = 'MyBudget'

@description('The total amount of cost or usage to track with the budget')
param amount int = 1000

@description('The time covered by a budget. Tracking of the amount will be reset based on the time grain.')
@allowed([
  'Monthly'
  'Quarterly'
  'Annually'
])
param timeGrain string = 'Monthly'

@description('The start date must be first of the month in YYYY-MM-DD format. Future start date should not
be more than three months. Past start date should be selected within the timegrain preiod.')
param startDate string

@description('The end date for the budget in YYYY-MM-DD format. If not provided, we default this to 10 years
from the start date.')
param endDate string

@description('Threshold value associated with a notification. Notification is sent when the cost exceeded
the threshold. It is always percent and has to be between 0.01 and 1000.')
param firstThreshold int = 90

@description('Threshold value associated with a notification. Notification is sent when the cost exceeded
```

```
the threshold. It is always percent and has to be between 0.01 and 1000.')
param secondThreshold int = 110

@description('The list of contact roles to send the budget notification to when the threshold is exceeded.')
param contactRoles array = [
  'Owner'
  'Contributor'
  'Reader'
]

@description('The list of email addresses to send the budget notification to when the threshold is
exceeded.')
param contactEmails array

@description('The list of action groups to send the budget notification to when the threshold is exceeded.
It accepts array of strings.')
param contactGroups array

@description('The set of values for the resource group filter.')
param resourceGroupFilterValues array

@description('The set of values for the meter category filter.')
param meterCategoryFilterValues array

resource budget 'Microsoft.Consumption/budgets@2021-10-01' = {
  name: budgetName
  properties: {
    timePeriod: {
      startDate: startDate
      endDate: endDate
    }
    timeGrain: timeGrain
    amount: amount
    category: 'Cost'
    notifications: {
      NotificationForExceededBudget1: {
        enabled: true
        operator: 'GreaterThan'
        threshold: firstThreshold
        contactEmails: contactEmails
        contactRoles: contactRoles
        contactGroups: contactGroups
      }
      NotificationForExceededBudget2: {
        enabled: true
        operator: 'GreaterThan'
        threshold: secondThreshold
        contactEmails: contactEmails
        contactRoles: contactRoles
        contactGroups: contactGroups
        thresholdType: 'Forecasted'
      }
    }
    filter: {
      and: [
        {
          dimensions: {
            name: 'ResourceGroupName'
            operator: 'In'
            values: resourceGroupFilterValues
          }
        }
        {
          dimensions: {
            name: 'MeterCategory'
            operator: 'In'
            values: meterCategoryFilterValues
          }
        }
```

```
      ]
    }
  }
}
```

One Azure resource is defined in the Bicep file:

- Microsoft.Consumption/budgets: Create an Azure budget.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
myContactEmails ='("user1@contoso.com", "user2@contoso.com")'
myContactGroups ='("/subscriptions/{sub-id}/resourceGroups/{rg-
name}/providers/microsoft.insights/actionGroups/groupone", "/subscriptions/{sub-
id}/resourceGroups/{rg-name}/providers/microsoft.insights/actionGroups/grouptwo")'
myRgFilterValues ='("resource-group-01", "resource-group-02")'
myMeterCategoryFilterValues ='("meter-category-01", "meter-category-02")'

az deployment sub create --name demoSubDeployment --location centralus --template-file main.bicep --
parameters startDate=<start-date> endDate=<end-date> contactEmails=$myContactEmails
contactGroups=$myContactGroups resourceGroupFilterValues=$myRgFilterValues
meterCategoryFilterValues=$myMeterCategoryFilterValues
```

You need to enter the following parameters:

- **startDate**: Replace **<start-date>** with the start date. It must be the first of the month in YYYY-MM-DD format. A future start date shouldn't be more than three months in the future. A past start date should be selected within the timegrain period.
- **endDate**: Replace **<end-date>** with the end date in YYYY-MM-DD format. If not provided, it defaults to ten years from the start date.
- **contactEmails**: First create a variable that holds your emails and then pass that variable. Replace the sample emails with the email addresses to send the budget notification to when the threshold is exceeded.
- **contactGroups**: First create a variable that holds your contact groups and then pass that variable. Replace the sample contact groups with the list of action groups to send the budget notification to when the threshold is exceeded. You must pass the resource ID of the action group, which you can get with az monitor action-group show or Get-AzActionGroup.
- **resourceGroupFilterValues**: First create a variable that holds your resource group filter values and then pass that variable. Replace the sample filter values with the set of values for your resource group filter.
- **meterCategoryFilterValues**: First create a variable that holds your meter category filter values and then pass that variable. Replace the sample filter values within parentheses with the set of values for your meter category filter.

> **NOTE**
>
> When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az consumption budget list
```

Clean up resources

When you no longer need the budget, use the Azure portal, Azure CLI, or Azure PowerShell to delete it:

- CLI
- PowerShell

```
az consumption budget delete --budget-name MyBudget
```

Next steps

In this quickstart, you created an Azure budget and deployed it using Bicep. To learn more about Cost Management and Billing and Bicep, continue on to the articles below.

- Read the Cost Management and Billing overview.
- Create budgets in the Azure portal.
- Learn more about Bicep.

Resource Manager template samples for diagnostic settings in Azure Monitor

9/21/2022 • 19 minutes to read • Edit Online

This article includes sample Azure Resource Manager templates to create diagnostic settings for an Azure resource. Each sample includes a template file and a parameters file with sample values to provide to the template.

To create a diagnostic setting for an Azure resource, add a resource of type `<resource namespace>/providers/diagnosticSettings` to the template. This article provides examples for some resource types, but the same pattern can be applied to other resource types. The collection of allowed logs and metrics will vary for each resource type.

> **NOTE**
>
> See Azure Monitor resource manager samples for a list of samples that are available and guidance on deploying them in your Azure subscription.

Diagnostic setting for Activity log

The following sample creates a diagnostic setting for an Activity log by adding a resource of type `Microsoft.Insights/diagnosticSettings` to the template.

> **IMPORTANT**
>
> Diagnostic settings for Activity logs are created for a subscription, not for a resource group like settings for Azure resources. To deploy the Resource management template, use `New-AzSubscriptionDeployment` for PowerShell or `az deployment sub create` for Azure CLI.

Template file

- Bicep
- JSON

```
targetScope = 'subscription'

@description('The name of the diagnostic setting.')
param settingName string

@description('The resource Id for the workspace.')
param workspaceId string

@description('The resource Id for the storage account.')
param storageAccountId string

@description('The resource Id for the event hub authorization rule.')
param eventHubAuthorizationRuleId string

@description('The name of the event hub.')
param eventHubName string

resource setting 'Microsoft.Insights/diagnosticSettings@2021-05-01-preview' = {
  name: settingName
  properties: {
    workspaceId: workspaceId
    storageAccountId: storageAccountId
    eventHubAuthorizationRuleId: eventHubAuthorizationRuleId
    eventHubName: eventHubName
    logs: [
      {
        category: 'Administrative'
        enabled: true
      }
      {
        category: 'Security'
        enabled: true
      }
      {
        category: 'ServiceHealth'
        enabled: true
      }
      {
        category: 'Alert'
        enabled: true
      }
      {
        category: 'Recommendation'
        enabled: true
      }
      {
        category: 'Policy'
        enabled: true
      }
      {
        category: 'Autoscale'
        enabled: true
      }
      {
        category: 'ResourceHealth'
        enabled: true
      }
    ]
  }
}
```

Parameter file

```
{
  "$schema": "https://schema.management.azure.com/schemas/2018-05-01/subscriptionDeploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "settingName": {
      "value": "Send to all locations"
    },
    "workspaceId": {
      "value": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourcegroups/MyResourceGroup/providers/microsoft.operationalinsights/workspaces/MyWorkspace"
    },
    "storageAccountId": {
      "value": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/MyResourceGroup/providers/Microsoft.Storage/storageAccounts/mystorageaccount"
    },
    "eventHubAuthorizationRuleId": {
      "value": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/MyResourceGroup/providers/Microsoft.EventHub/namespaces/MyNameSpace/authorizatio
nrules/RootManageSharedAccessKey"
    },
    "eventHubName": {
      "value": "my-eventhub"
    }
  }
}
```

Diagnostic setting for Azure Data Explorer

The following sample creates a diagnostic setting for an Azure Data Explorer cluster by adding a resource of type `Microsoft.Kusto/clusters/providers/diagnosticSettings` to the template.

Template file

- Bicep
- JSON

```
param clusterName string
param settingName string
param workspaceId string
param storageAccountId string
param eventHubAuthorizationRuleId string
param eventHubName string

resource cluster 'Microsoft.Kusto/clusters@2022-02-01' existing = {
  name: clusterName
}

resource setting 'Microsoft.Insights/diagnosticSettings@2021-05-01-preview' = {
  name: settingName
  scope: cluster
  properties: {
    workspaceId: workspaceId
    storageAccountId: storageAccountId
    eventHubAuthorizationRuleId: eventHubAuthorizationRuleId
    eventHubName: eventHubName
    metrics: []
    logs: [
      {
        category: 'Command'
        categoryGroup: null
        enabled: true
        retentionPolicy: {
          enabled: false
          days: 0
```

```
        days: 0
      }
    }
    {
      category: 'Query'
      categoryGroup: null
      enabled: true
      retentionPolicy: {
        enabled: false
        days: 0
      }
    }
    {
      category: 'Journal'
      categoryGroup: null
      enabled: true
      retentionPolicy: {
        enabled: false
        days: 0
      }
    }
    {
      category: 'SucceededIngestion'
      categoryGroup: null
      enabled: false
      retentionPolicy: {
        enabled: false
        days: 0
      }
    }
    {
      category: 'FailedIngestion'
      categoryGroup: null
      enabled: false
      retentionPolicy: {
        enabled: false
        days: 0
      }
    }
    {
      category: 'IngestionBatching'
      categoryGroup: null
      enabled: false
      retentionPolicy: {
        enabled: false
        days: 0
      }
    }
    {
      category: 'TableUsageStatistics'
      categoryGroup: null
      enabled: false
      retentionPolicy: {
        enabled: false
        days: 0
      }
    }
    {
      category: 'TableDetails'
      categoryGroup: null
      enabled: false
      retentionPolicy: {
        enabled: false
        days: 0
      }
    }
  ]
}
}
```

Parameter file

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "clusterName": {
      "value": "kustoClusterName"
    },
    "diagnosticSettingName": {
      "value": "A new Diagnostic Settings configuration"
    },
    "workspaceId": {
      "value": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourcegroups/MyResourceGroup/providers/microsoft.operationalinsights/workspaces/MyWorkspace"
    },
    "storageAccountId": {
      "value": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/MyResourceGroup/providers/Microsoft.Storage/storageAccounts/mystorageaccount"
    },
    "eventHubAuthorizationRuleId": {
      "value": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/MyResourceGroup/providers/Microsoft.EventHub/namespaces/MyNameSpace/authorizatio
nrules/RootManageSharedAccessKey"
    },
    "eventHubName": {
      "value": "myEventhub"
    }
  }
}
```

Template file - enabling the 'audit' category group

- Bicep
- JSON

```
param clusterName string
param settingName string
param workspaceId string
param storageAccountId string
param eventHubAuthorizationRuleId string
param eventHubName string

resource cluster 'Microsoft.Kusto/clusters@2022-02-01' existing = {
  name: clusterName
}

resource setting 'Microsoft.Insights/diagnosticSettings@2021-05-01-preview' = {
  name: settingName
  scope: cluster
  properties: {
    workspaceId: workspaceId
    storageAccountId: storageAccountId
    eventHubAuthorizationRuleId: eventHubAuthorizationRuleId
    eventHubName: eventHubName
    logs: [
      {
        category: null
        categoryGroup: 'audit'
        enabled: true
        retentionPolicy: {
          enabled: false
          days: 0
        }
      }
    ]
  }
}
```

Diagnostic setting for Azure Key Vault

The following sample creates a diagnostic setting for an Azure Key Vault by adding a resource of type `Microsoft.KeyVault/vaults/providers/diagnosticSettings` to the template.

> **IMPORTANT**
>
> For Azure Key Vault, the event hub must be in the same region as the key vault.

Template file

- Bicep
- JSON

```
@description('The name of the diagnostic setting.')
param settingName string

@description('The name of the key vault.')
param vaultName string

@description('The resource Id of the workspace.')
param workspaceId string

@description('The resource Id of the storage account.')
param storageAccountId string

@description('The resource Id for the event hub authorization rule.')
param eventHubAuthorizationRuleId string

@description('The name of the event hub.')
param eventHubName string

resource vault 'Microsoft.KeyVault/vaults@2021-11-01-preview' existing = {
  name: vaultName
}

resource setting 'Microsoft.Insights/diagnosticSettings@2021-05-01-preview' = {
  name: settingName
  scope: vault
  properties: {
    workspaceId: workspaceId
    storageAccountId: storageAccountId
    eventHubAuthorizationRuleId: eventHubAuthorizationRuleId
    eventHubName: eventHubName
    logs: [
      {
        category: 'AuditEvent'
        enabled: true
      }
    ]
    metrics: [
      {
        category: 'AllMetrics'
        enabled: true
      }
    ]
  }
}
```

Parameter file

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "settingName": {
        "value": "Send to all locations"
    },
    "vaultName": {
      "value": "MyVault"
    },
    "workspaceId": {
      "value": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourcegroups/MyResourceGroup/providers/microsoft.operationalinsights/workspaces/MyWorkspace"
    },
    "storageAccountId": {
      "value": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/MyResourceGroup/providers/Microsoft.Storage/storageAccounts/mystorageaccount"
    },
    "eventHubAuthorizationRuleId": {
      "value": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/MyResourceGroup/providers/Microsoft.EventHub/namespaces/MyNameSpace/authorizatio
nrules/RootManageSharedAccessKey"
    },
    "eventHubName": {
      "value": "my-eventhub"
    }
  }
}
```

Diagnostic setting for Azure SQL database

The following sample creates a diagnostic setting for an Azure SQL database by adding a resource of type `microsoft.sql/servers/databases/providers/diagnosticSettings` to the template.

Template file

- Bicep
- JSON

```
@description('The name of the diagnostic setting.')
param settingName string

@description('The name of the Azure SQL database server.')
param serverName string

@description('The name of the SQL database.')
param dbName string

@description('The resource Id of the workspace.')
param workspaceId string

@description('The resource Id of the storage account.')
param storageAccountId string

@description('The resource Id of the event hub authorization rule.')
param eventHubAuthorizationRuleId string

@description('The name of the event hub.')
param eventHubName string

resource dbServer 'Microsoft.Sql/servers@2021-11-01-preview' existing = {
  name: serverName
}
```

```
resource db 'Microsoft.Sql/servers/databases@2021-11-01-preview' existing = {
  parent: dbServer
  name: dbName
}

resource setting 'Microsoft.Insights/diagnosticSettings@2021-05-01-preview' = {
  name: settingName
  scope: db
  properties: {
    workspaceId: workspaceId
    storageAccountId: storageAccountId
    eventHubAuthorizationRuleId: eventHubAuthorizationRuleId
    eventHubName: eventHubName
    logs: [
      {
        category: 'SQLInsights'
        enabled: true
      }
      {
        category: 'AutomaticTuning'
        enabled: true
      }
      {
        category: 'QueryStoreRuntimeStatistics'
        enabled: true
      }
      {
        category: 'QueryStoreWaitStatistics'
        enabled: true
      }
      {
        category: 'Errors'
        enabled: true
      }
      {
        category: 'DatabaseWaitStatistics'
        enabled: true
      }
      {
        category: 'Timeouts'
        enabled: true
      }
      {
        category: 'Blocks'
        enabled: true
      }
      {
        category: 'Deadlocks'
        enabled: true
      }
    ]
    metrics: [
      {
        category: 'Basic'
        enabled: true
      }
      {
        category: 'InstanceAndAppAdvanced'
        enabled: true
      }
      {
        category: 'WorkloadManagement'
        enabled: true
      }
    ]
  }
}
```

Parameter file

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "settingName": {
        "value": "Send to all locations"
    },
    "serverName": {
      "value": "MySqlServer"
    },
    "dbName": {
      "value": "MySqlDb"
    },
    "workspaceId": {
      "value": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourcegroups/MyResourceGroup/providers/microsoft.operationalinsights/workspaces/MyWorkspace"
    },
    "storageAccountId": {
      "value": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/MyResourceGroup/providers/Microsoft.Storage/storageAccounts/mystorageaccount"
    },
    "eventHubAuthorizationRuleId": {
      "value": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/MyResourceGroup/providers/Microsoft.EventHub/namespaces/MyNameSpace/authorizatio
nrules/RootManageSharedAccessKey"
    },
    "eventHubName": {
      "value": "my-eventhub"
    }
  }
}
```

Diagnostic setting for Azure SQL managed instance

The following sample creates a diagnostic setting for an Azure SQL managed instance by adding a resource of type `microsoft.sql/managedInstances/providers/diagnosticSettings` to the template.

Template file

- Bicep
- JSON

```
param sqlManagedInstanceName string
param diagnosticSettingName string
param diagnosticWorkspaceId string
param storageAccountId string
param eventHubAuthorizationRuleId string
param eventHubName string

resource instance 'Microsoft.Sql/managedInstances@2021-11-01-preview' existing = {
  name: sqlManagedInstanceName
}

resource setting 'Microsoft.Insights/diagnosticSettings@2021-05-01-preview' = {
  name: diagnosticSettingName
  scope: instance
  properties: {
    workspaceId: diagnosticWorkspaceId
    storageAccountId: storageAccountId
    eventHubAuthorizationRuleId: eventHubAuthorizationRuleId
    eventHubName: eventHubName
    logs: [
      {
        category: 'ResourceUsageStats'
        enabled: true
      }
      {
        category: 'DevOpsOperationsAudit'
        enabled: true
      }
      {
        category: 'SQLSecurityAuditEvents'
        enabled: true
      }
    ]
  }
}
```

Parameter file

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "sqlManagedInstanceName": {
        "value": "MyInstanceName"
    },
    "diagnosticSettingName": {
        "value": "Send to all locations"
    },
    "diagnosticWorkspaceId": {
        "value": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourcegroups/MyResourceGroup/providers/microsoft.operationalinsights/workspaces/MyWorkspace"
    },
    "storageAccountId": {
        "value": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/MyResourceGroup/providers/Microsoft.Storage/storageAccounts/mystorageaccount"
    },
    "eventHubAuthorizationRuleId": {
        "value": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/MyResourceGroup/providers/Microsoft.EventHub/namespaces/MyNameSpace/authorizatio
nrules/RootManageSharedAccessKey"
    },
    "eventHubName": {
        "value": "myEventhub"
    }
  }
}
```

Diagnostic setting for Azure SQL managed database

The following sample creates a diagnostic setting for an Azure SQL managed database by adding a resource of type `microsoft.sql/managedInstances/databases/providers/diagnosticSettings` to the template.

Template file

- Bicep
- JSON

```
param sqlManagedInstanceName string
param sqlManagedDatabaseName string
param diagnosticSettingName string
param diagnosticWorkspaceId string
param storageAccountId string
param eventHubAuthorizationRuleId string
param eventHubName string

resource dbInstance 'Microsoft.Sql/managedInstances@2021-11-01-preview' existing = {
  name:sqlManagedInstanceName
}

resource db 'Microsoft.Sql/managedInstances/databases@2021-11-01-preview' existing = {
  name: sqlManagedDatabaseName
  parent: dbInstance
}

resource setting 'Microsoft.Insights/diagnosticSettings@2021-05-01-preview' = {
  name: diagnosticSettingName
  scope: db
  properties: {
    workspaceId: diagnosticWorkspaceId
    storageAccountId: storageAccountId
    eventHubAuthorizationRuleId: eventHubAuthorizationRuleId
    eventHubName: eventHubName
    logs: [
      {
        category: 'SQLInsights'
        enabled: true
      }
      {
        category: 'QueryStoreRuntimeStatistics'
        enabled: true
      }
      {
        category: 'QueryStoreWaitStatistics'
        enabled: true
      }
      {
        category: 'Errors'
        enabled: true
      }
    ]
  }
}
```

Parameter file

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "sqlManagedInstanceName": {
      "value": "MyInstanceName"
    },
    "sqlManagedDatabaseName": {
      "value": "MyManagedDatabaseName"
    },
    "diagnosticSettingName": {
      "value": "Send to all locations"
    },
    "diagnosticWorkspaceId": {
      "value": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourcegroups/MyResourceGroup/providers/microsoft.operationalinsights/workspaces/MyWorkspace"
    },
    "storageAccountId": {
      "value": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/MyResourceGroup/providers/Microsoft.Storage/storageAccounts/mystorageaccount"
    },
    "eventHubAuthorizationRuleId": {
      "value": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/MyResourceGroup/providers/Microsoft.EventHub/namespaces/MyNameSpace/authorizatio
nrules/RootManageSharedAccessKey"
    },
    "eventHubName": {
      "value": "myEventhub"
    }
  }
}
```

Diagnostic setting for Recovery Services vault

The following sample creates a diagnostic setting for an Azure Recovery Services vault by adding a resource of type `microsoft.recoveryservices/vaults/providers/diagnosticSettings` to the template. This example specifies the collection mode as described in Azure resource logs. Specify `Dedicated` or `AzureDiagnostics` for the `logAnalyticsDestinationType` property.

Template file

- Bicep
- JSON

```
param recoveryServicesName string
param settingName string
param workspaceId string
param storageAccountId string
param eventHubAuthorizationRuleId string
param eventHubName string

resource vault 'Microsoft.RecoveryServices/vaults@2021-08-01' existing = {
  name: recoveryServicesName
}

resource setting 'Microsoft.Insights/diagnosticSettings@2021-05-01-preview' = {
  name: settingName
  scope: vault
  properties: {
    workspaceId: workspaceId
    storageAccountId: storageAccountId
    eventHubAuthorizationRuleId: eventHubAuthorizationRuleId
    eventHubName: eventHubName
```

```
    logs: [
      {
        category: 'AzureBackupReport'
        enabled: false
      }
      {
        category: 'CoreAzureBackup'
        enabled: true
      }
      {
        category: 'AddonAzureBackupJobs'
        enabled: true
      }
      {
        category: 'AddonAzureBackupAlerts'
        enabled: true
      }
      {
        category: 'AddonAzureBackupPolicy'
        enabled: true
      }
      {
        category: 'AddonAzureBackupStorage'
        enabled: true
      }
      {
        category: 'AddonAzureBackupProtectedInstance'
        enabled: true
      }
      {
        category: 'AzureSiteRecoveryJobs'
        enabled: false
      }
      {
        category: 'AzureSiteRecoveryEvents'
        enabled: false
      }
      {
        category: 'AzureSiteRecoveryReplicatedItems'
        enabled: false
      }
      {
        category: 'AzureSiteRecoveryReplicationStats'
        enabled: false
      }
      {
        category: 'AzureSiteRecoveryRecoveryPoints'
        enabled: false
      }
      {
        category: 'AzureSiteRecoveryReplicationDataUploadRate'
        enabled: false
      }
      {
        category: 'AzureSiteRecoveryProtectedDiskDataChurn'
        enabled: false
      }
    ]
    logAnalyticsDestinationType: 'Dedicated'
  }
}
```

Parameter file

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "settingName": {
      "value": "Send to all locations"
    },
    "recoveryServicesName": {
      "value": "my-vault"
    },
    "workspaceId": {
      "value": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourcegroups/MyResourceGroup/providers/microsoft.operationalinsights/workspaces/MyWorkspace"
    },
    "storageAccountId": {
      "value": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/MyResourceGroup/providers/Microsoft.Storage/storageAccounts/mystorageaccount"
    },
    "eventHubAuthorizationRuleId": {
      "value": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/MyResourceGroup/providers/Microsoft.EventHub/namespaces/MyNameSpace/authorizatio
nrules/RootManageSharedAccessKey"
    },
    "eventHubName": {
      "value": "my-eventhub"
    }
  }
}
```

Diagnostic setting for Log Analytics workspace

The following sample creates a diagnostic setting for a Log Analytics workspace vault by adding a resource of type `Microsoft.OperationalInsights/workspaces/providers/diagnosticSettings` to the template. This example sends audit data about queries executed in the workspace to the same workspace.

Template file

- Bicep
- JSON

```
param workspaceName string
param settingName string
param workspaceId string
param storageAccountId string
param eventHubAuthorizationRuleId string
param eventHubName string

resource workspace 'Microsoft.OperationalInsights/workspaces@2021-12-01-preview' existing = {
  name: workspaceName
}
resource setting 'Microsoft.Insights/diagnosticSettings@2021-05-01-preview' = {
  name: settingName
  scope: workspace
  properties: {
    workspaceId: workspaceId
    storageAccountId: storageAccountId
    eventHubAuthorizationRuleId: eventHubAuthorizationRuleId
    eventHubName: eventHubName
    logs: [
      {
        category: 'Audit'
        enabled: true
      }
    ]
  }
}
```

Parameter file

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "settingName": {
        "value": "Send to all locations"
    },
    "workspaceName": {
      "value": "MyWorkspace"
    },
    "workspaceId": {
      "value": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourcegroups/MyResourceGroup/providers/microsoft.operationalinsights/workspaces/MyWorkspace"
    },
    "storageAccountId": {
      "value": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/MyResourceGroup/providers/Microsoft.Storage/storageAccounts/mystorageaccount"
    },
    "eventHubAuthorizationRuleId": {
      "value": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/MyResourceGroup/providers/Microsoft.EventHub/namespaces/MyNameSpace/authorizatio
nrules/RootManageSharedAccessKey"
    },
    "eventHubName": {
      "value": "my-eventhub"
    }
  }
}
```

Diagnostic setting for Azure Storage

The following sample creates a diagnostic setting for each storage service endpoint that is available in the storage account. A setting is applied to each individual storage service that is available on the account. The storage services that are available depend on the type of storage account. This template creates a diagnostic

setting for a storage service in the account only if it exists for the account. For each available service, the diagnostic setting enables transaction metrics, and the collection of resource logs for read, write, and delete operations.

Template file

- Bicep
- JSON

main.bicep

```
param storageAccountName string
param settingName string
param storageSyncName string
param workspaceId string

module nested './module.bicep' = {
  name: 'nested'
  params: {
    endpoints: reference(resourceId('Microsoft.Storage/storageAccounts', storageAccountName), '2019-06-01',
'Full').properties.primaryEndpoints
    settingName: settingName
    storageAccountName: storageAccountName
    storageSyncName: storageSyncName
    workspaceId: workspaceId
  }
}
```

module.bicep

```
param endpoints object
param settingName string
param storageAccountName string
param storageSyncName string
param workspaceId string

var hasblob = contains(endpoints, 'blob')
var hastable = contains(endpoints, 'table')
var hasfile = contains(endpoints, 'file')
var hasqueue = contains(endpoints, 'queue')

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-09-01' existing = {
  name: storageAccountName
}

resource diagnosticSetting 'Microsoft.Insights/diagnosticSettings@2021-05-01-preview' = {
  name: settingName
  scope: storageAccount
  properties: {
    workspaceId: workspaceId
    storageAccountId: resourceId('Microsoft.Storage/storageAccounts', storageSyncName)
    metrics: [
      {
        category: 'Transaction'
        enabled: true
      }
    ]
  }
}

resource blob 'Microsoft.Storage/storageAccounts/blobServices@2021-09-01' existing = {
  name:storageAccountName
}

resource blobSetting 'Microsoft.Insights/diagnosticSettings@2021-05-01-preview' = if (hasblob) {
```

```
  name: settingName
  scope: blob
  properties: {
    workspaceId: workspaceId
    storageAccountId: resourceId('Microsoft.Storage/storageAccounts', storageSyncName)
    logs: [
      {
        category: 'StorageRead'
        enabled: true
      }
      {
        category: 'StorageWrite'
        enabled: true
      }
      {
        category: 'StorageDelete'
        enabled: true
      }
    ]
    metrics: [
      {
        category: 'Transaction'
        enabled: true
      }
    ]
  }
}

resource table 'Microsoft.Storage/storageAccounts/tableServices@2021-09-01' existing = {
  name:storageAccountName
}

resource tableSetting 'Microsoft.Insights/diagnosticSettings@2021-05-01-preview' = if (hastable) {
  name: settingName
  scope: table
  properties: {
    workspaceId: workspaceId
    storageAccountId: resourceId('Microsoft.Storage/storageAccounts', storageSyncName)
    logs: [
      {
        category: 'StorageRead'
        enabled: true
      }
      {
        category: 'StorageWrite'
        enabled: true
      }
      {
        category: 'StorageDelete'
        enabled: true
      }
    ]
    metrics: [
      {
        category: 'Transaction'
        enabled: true
      }
    ]
  }
}

resource file 'Microsoft.Storage/storageAccounts/fileServices@2021-09-01' existing = {
  name:storageAccountName
}

resource fileSetting 'Microsoft.Insights/diagnosticSettings@2021-05-01-preview' = if (hasfile) {
  name: settingName
  scope: file
  properties: {
```

```
  properties: {
    workspaceId: workspaceId
    storageAccountId: resourceId('Microsoft.Storage/storageAccounts', storageSyncName)
    logs: [
      {
        category: 'StorageRead'
        enabled: true
      }
      {
        category: 'StorageWrite'
        enabled: true
      }
      {
        category: 'StorageDelete'
        enabled: true
      }
    ]
    metrics: [
      {
        category: 'Transaction'
        enabled: true
      }
    ]
  }
}

resource queue 'Microsoft.Storage/storageAccounts/queueServices@2021-09-01' existing = {
  name:storageAccountName
}


resource queueSetting 'Microsoft.Insights/diagnosticSettings@2021-05-01-preview' = if (hasqueue) {
  name: settingName
  properties: {
    workspaceId: workspaceId
    storageAccountId: resourceId('Microsoft.Storage/storageAccounts', storageSyncName)
    logs: [
      {
        category: 'StorageRead'
        enabled: true
      }
      {
        category: 'StorageWrite'
        enabled: true
      }
      {
        category: 'StorageDelete'
        enabled: true
      }
    ]
    metrics: [
      {
        category: 'Transaction'
        enabled: true
      }
    ]
  }
}
```

Parameter file

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "storageAccountName": {
      "value": "mymonitoredstorageaccount"
    },
    "settingName": {
      "value": "Send to all locations"
    },
    "storageSyncName": {
      "value": "mystorageaccount"
    },
    "workspaceId": {
      "value": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourcegroups/MyResourceGroup/providers/microsoft.operationalinsights/workspaces/MyWorkspace"
    }
  }
}
```

Next steps

- Get other sample templates for Azure Monitor.
- Learn more about diagnostic settings.

Quickstart: Create a policy assignment to identify non-compliant resources by using a Bicep file

9/21/2022 • 2 minutes to read • Edit Online

The first step in understanding compliance in Azure is to identify the status of your resources. This quickstart steps you through the process of using a Bicep file compiled to an Azure Resource Manager (ARM) deployment template to create a policy assignment to identify virtual machines that aren't using managed disks. At the end of this process, you'll successfully identify virtual machines that aren't using managed disks. They're *non-compliant* with the policy assignment.

An ARM template is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project. The template uses declarative syntax. In declarative syntax, you describe your intended deployment without writing the sequence of programming commands to create the deployment.

If your environment meets the prerequisites and you're familiar with using ARM templates, select the **Deploy to Azure** button. The template opens in the Azure portal.

Deploy to Azure

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin.
- Bicep version `0.3` or higher installed. If you don't yet have Bicep CLI or need to update, see Install Bicep.

Review the Bicep file

In this quickstart, you create a policy assignment and assign a built-in policy definition called *Audit VMs that do not use managed disks*. For a partial list of available built-in policies, see Azure Policy samples.

Create the following Bicep file as `assignment.bicep`:

```
param policyAssignmentName string = 'audit-vm-manageddisks'
param policyDefinitionID string = '/providers/Microsoft.Authorization/policyDefinitions/06a78e20-9358-41c9-923c-fb736d382a4d'

resource assignment 'Microsoft.Authorization/policyAssignments@2021-09-01' = {
    name: policyAssignmentName
    scope: subscriptionResourceId('Microsoft.Resources/resourceGroups', resourceGroup().name)
    properties: {
        policyDefinitionId: policyDefinitionID
    }
}

output assignmentId string = assignment.id
```

The resource defined in the file is:

- Microsoft.Authorization/policyAssignments

Deploy the template

> **NOTE**
> Azure Policy service is free. For more information, see Overview of Azure Policy.

After the Bicep CLI is installed and file created, you can deploy the Bicep file with:

- PowerShell
- Azure CLI

```
New-AzResourceGroupDeployment `
  -Name PolicyDeployment `
  -ResourceGroupName PolicyGroup `
  -TemplateFile assignment.bicep
```

Some other resources:

- To find more samples templates, see Azure Quickstart Template.
- To see the template reference, go to Azure template reference.
- To learn how to develop ARM templates, see Azure Resource Manager documentation.
- To learn subscription-level deployment, see Create resource groups and resources at the subscription level.

Validate the deployment

Select **Compliance** in the left side of the page. Then locate the *Audit VMs that do not use managed disks* policy assignment you created.

If there are any existing resources that aren't compliant with this new assignment, they appear under **Non-compliant resources**.

For more information, see How compliance works.

Clean up resources

To remove the assignment created, follow these steps:

1. Select **Compliance** (or **Assignments**) in the left side of the Azure Policy page and locate the *Audit VMs that do not use managed disks* policy assignment you created.

2. Right-click the *Audit VMs that do not use managed disks* policy assignment and select **Delete assignment**.

3. Delete the `assignment.bicep` file.

Next steps

In this quickstart, you assigned a built-in policy definition to a scope and evaluated its compliance report. The policy definition validates that all the resources in the scope are compliant and identifies which ones aren't.

To learn more about assigning policies to validate that new resources are compliant, continue to the tutorial for:

Creating and managing policies

Quickstart: Create a shared query using Bicep

9/21/2022 • 2 minutes to read • Edit Online

Azure Resource Graph is an Azure service designed to extend Azure Resource Management by providing efficient and performant resource exploration with the ability to query at scale across a given set of subscriptions so you can effectively govern your environment. With Resource Graph queries, you can:

- Query resources with complex filtering, grouping, and sorting by resource properties.
- Explore resources iteratively based on governance requirements.
- Assess the impact of applying policies in a vast cloud environment.
- Query changes made to resource properties (preview).

Resource Graph queries can be saved as a *private query* or a *shared query*. A private query is saved to the individual's Azure portal profile and isn't visible to others. A shared query is a Resource Manager object that can be shared with others through permissions and role-based access. A shared query provides common and consistent execution of resource discovery. This quickstart uses Bicep to create a shared query.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

In this quickstart, you create a shared query called *Count VMs by OS*. To try this query in SDK or in portal with Resource Graph Explorer, see Samples - Count virtual machines by OS type.

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('The name of the shared query.')
param queryName string = 'Count VMs by OS'

@description('The Azure Resource Graph query to be saved to the shared query.')
param queryCode string = 'Resources | where type =~ \'Microsoft.Compute/virtualMachines\' | summarize
count() by tostring(properties.storageProfile.osDisk.osType)'

@description('The description of the saved Azure Resource Graph query.')
param queryDescription string = 'This shared query counts all virtual machine resources and summarizes by
the OS type.'

resource query 'Microsoft.ResourceGraph/queries@2018-09-01-preview' = {
  name: queryName
  location: 'global'
  properties: {
    query: queryCode
    description: queryDescription
  }
}
```

The resource defined in the Bicep file is:

- Microsoft.ResourceGraph/queries

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.

> **NOTE**
> The Bicep file isn't required to be named **main.bicep**. If you save the file with a different name, you must change the name of the template file in the deployment step below.

2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep
```

   When the deployment finishes, you should see a message indicating the deployment succeeded.

Some other resources:

- To see the template reference, go to Azure template reference.
- To learn how to create Bicep files, see Quickstart: Create Bicep files with Visual Studio Code.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When you no longer need the resource that you created, delete the resource group using Azure CLI or Azure PowerShell.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you created a Resource Graph shared query using Bicep.

To learn more about shared queries, continue to the tutorial for:

Manage queries in Azure portal

Create or update Azure custom roles using Bicep

9/21/2022 • 3 minutes to read • Edit Online

If the Azure built-in roles don't meet the specific needs of your organization, you can create your own custom roles. This article describes how to create or update a custom role using Bicep.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

To create a custom role, you specify a role name, role permissions, and where the role can be used. In this article, you create a role named *Custom Role - RG Reader* with resource permissions that can be assigned at a subscription scope or lower.

Prerequisites

To create a custom role, you must have permissions to create custom roles, such as Owner or User Access Administrator.

You also must have an active Azure subscription. If you don't have one, you can create a free account before you begin.

Review the Bicep file

The Bicep file used in this article is from Azure Quickstart Templates. The Bicep file has four parameters and a resources section. The four parameters are:

- Array of actions with a default value of `["Microsoft.Resources/subscriptions/resourceGroups/read"]`.
- Array of `notActions` with an empty default value.
- Role name with a default value of `Custom Role - RG Reader`.
- Role description with a default value of `Subscription Level Deployment of a Role Definition`.

The scope where this custom role can be assigned is set to the current subscription.

```
targetScope = 'subscription'

@description('Array of actions for the roleDefinition')
param actions array = [
  'Microsoft.Resources/subscriptions/resourceGroups/read'
]

@description('Array of notActions for the roleDefinition')
param notActions array = []

@description('Friendly name of the role definition')
param roleName string = 'Custom Role - RG Reader'

@description('Detailed description of the role definition')
param roleDescription string = 'Subscription Level Deployment of a Role Definition'

var roleDefName = guid(subscription().id, string(actions), string(notActions))

resource roleDef 'Microsoft.Authorization/roleDefinitions@2018-07-01' = {
  name: roleDefName
  properties: {
    roleName: roleName
    description: roleDescription
    type: 'customRole'
    permissions: [
      {
        actions: actions
        notActions: notActions
      }
    ]
    assignableScopes: [
      subscription().id
    ]
  }
}
```

The resource defined in the Bicep file is:

- Microsoft.Authorization/roleDefinitions

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
$myActions='("Microsoft.Resources/resources/read","Microsoft.Resources/subscriptions/resourceGroups/read")'

az deployment sub create --location eastus --name customRole --template-file main.bicep --parameters actions=$myActions
```

> **NOTE**
>
> Create a variable called **myActions** and then pass that variable. Replace the sample actions with the actions for the roleDefinition.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to verify that the custom role was created.

- CLI
- PowerShell

```
az role definition list --name "Custom Role - RG Reader"
```

Update a custom role

Similar to creating a custom role, you can update an existing custom role using Bicep. To update a custom role, you need to specify the role you want to update.

Here are the changes you would need to make to the previous Bicep file to update the custom role.

1. Include the role ID as a parameter.

```
...
@description('ID of the role definition')
param roleDefName string
...
```

2. Remove the roleDefName variable. You'll get a warning if you have a parameter and variable with the same name.

3. Use Azure CLI or Azure PowerShell to get the roleDefName.

   - CLI
   - PowerShell

```
az role definition list --name "Custom Role - RG Reader"
```

4. Use Azure CLI or Azure PowerShell to deploy the updated Bicep file, replacing `<name-id>` with the roleDefName, and replacing the sample actions with the updated actions for the roleDefinition.

   - CLI
   - PowerShell

```
$myActions='("Microsoft.Resources/resources/read","Microsoft.Resources/subscriptions/resourceGroups/r
ead")'

az deployment sub create --location eastus --name customrole --template-file main.bicep --parameters
actions=$myActions roleDefName="name-id" roleName="Custom Role - RG Reader"
```

> **NOTE**
>
> It may take several minutes for the updated role definition to be propagated.

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to remove the custom role.

- CLI
- PowerShell

```
az role definition delete --name "Custom Role - RG Reader"
```

Next steps

- Understand Azure role definitions
- Bicep documentation

Quickstart: Assign an Azure role using Bicep

9/21/2022 • 2 minutes to read • Edit Online

Azure role-based access control (Azure RBAC) is the way that you manage access to Azure resources. In this quickstart, you create a resource group and grant a user access to create and manage virtual machines in the resource group. This quickstart uses Bicep to grant the access.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

To assign Azure roles and remove role assignments, you must have:

- If you don't have an Azure subscription, create a free account before you begin.
- `Microsoft.Authorization/roleAssignments/write` and `Microsoft.Authorization/roleAssignments/delete` permissions, such as User Access Administrator or Owner.
- To assign a role, you must specify three elements: security principal, role definition, and scope. For this quickstart, the security principal is you or another user in your directory, the role definition is Virtual Machine Contributor, and the scope is a resource group that you specify.

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates. The Bicep file has two parameters and a resources section. In the resources section, notice that it has the three elements of a role assignment: security principal, role definition, and scope.

```
@description('Specifies the role definition ID used in the role assignment.')
param roleDefinitionID string

@description('Specifies the principal ID assigned to the role.')
param principalId string

var roleAssignmentName= guid(principalId, roleDefinitionID, resourceGroup().id)
resource roleAssignment 'Microsoft.Authorization/roleAssignments@2021-04-01-preview' = {
  name: roleAssignmentName
  properties: {
    roleDefinitionId: resourceId('Microsoft.Authorization/roleDefinitions', roleDefinitionID)
    principalId: principalId
  }
}
```

The resource defined in the Bicep file is:

- Microsoft.Authorization/roleAssignments

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.
   - CLI

- PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
roleDefinitionID=9980e02c-c2be-4d73-94e8-173b1dc7cf3c principalId=<principal-id>
```

> **NOTE**
>
> Replace **<principal-id>** with the principal ID assigned to the role.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az role assignment list --resource-group exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to remove the role assignment. For more information, see Remove Azure role assignments.

Use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

Tutorial: Grant a user access to Azure resources using Azure PowerShell

Quickstart: Create activity log alerts on service notifications using a Bicep file

9/21/2022 • 2 minutes to read • Edit Online

This article shows you how to set up activity log alerts for service health notifications by using a Bicep file.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Service health notifications are stored in the Azure activity log. Given the possibly large volume of information stored in the activity log, there is a separate user interface to make it easier to view and set up alerts on service health notifications.

You can receive an alert when Azure sends service health notifications to your Azure subscription. You can configure the alert based on:

- The class of service health notification (Service issues, Planned maintenance, Health advisories).
- The subscription affected.
- The service(s) affected.
- The region(s) affected.

> **NOTE**
> Service health notifications does not send an alert regarding resource health events.

You also can configure who the alert should be sent to:

- Select an existing action group.
- Create a new action group (that can be used for future alerts).

To learn more about action groups, see Create and manage action groups.

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin.
- To run the commands from your local computer, install Azure CLI or the Azure PowerShell modules. For more information, see Install the Azure CLI and Install Azure PowerShell.

Review the Bicep file

The following Bicep file creates an action group with an email target and enables all service health notifications for the target subscription. Save this Bicep as *CreateServiceHealthAlert.bicep*.

```
param actionGroups_name string = 'SubHealth'
param activityLogAlerts_name string = 'ServiceHealthActivityLogAlert'
param emailAddress string

var alertScope = '/subscriptions/${subscription().subscriptionId}'

resource actionGroups_name_resource 'microsoft.insights/actionGroups@2019-06-01' = {
  name: actionGroups_name
  location: 'Global'
  properties: {
    groupShortName: actionGroups_name
    enabled: true
    emailReceivers: [
      {
        name: actionGroups_name
        emailAddress: emailAddress
      }
    ]
    smsReceivers: []
    webhookReceivers: []
  }
}

resource activityLogAlerts_name_resource 'microsoft.insights/activityLogAlerts@2017-04-01' = {
  name: activityLogAlerts_name
  location: 'Global'
  properties: {
    scopes: [
      alertScope
    ]
    condition: {
      allOf: [
        {
          field: 'category'
          equals: 'ServiceHealth'
        }
        {
          field: 'properties.incidentType'
          equals: 'Incident'
        }
      ]
    }
    actions: {
      actionGroups: [
        {
          actionGroupId: actionGroups_name_resource.id
          webhookProperties: {}
        }
      ]
    }
    enabled: true
  }
}
```

The Bicep file defines two resources:

- Microsoft.Insights/actionGroups
- Microsoft.Insights/activityLogAlerts

Deploy the Bicep file

Deploy the Bicep file using Azure CLI and Azure PowerShell. Replace the sample values for **Resource Group** and **emailAddress** with appropriate values for your environment.

- CLI
- PowerShell

```
az login
az deployment group create --name CreateServiceHealthAlert --resource-group my-resource-group --template-
file CreateServiceHealthAlert.bicep --parameters emailAddress='user@contoso.com'
```

Validate the deployment

Verify that the workspace has been created using one of the following commands. Replace the sample values for **Resource Group** with the value you used above.

- CLI
- PowerShell

```
az monitor activity-log alert show --resource-group my-resource-group --name ServiceHealthActivityLogAlert
```

Clean up resources

If you plan to continue working with subsequent quickstarts and tutorials, you might want to leave these resources in place. When no longer needed, delete the resource group, which deletes the alert rule and the related resources. To delete the resource group by using Azure CLI or Azure PowerShell

- CLI
- PowerShell

```
az group delete --name my-resource-group
```

Next steps

- Learn about best practices for setting up Azure Service Health alerts.
- Learn how to setup mobile push notifications for Azure Service Health.
- Learn how to configure webhook notifications for existing problem management systems.
- Learn about service health notifications.
- Learn about notification rate limiting.
- Review the activity log alert webhook schema.
- Get an overview of activity log alerts, and learn how to receive alerts.
- Learn more about action groups.

Quickstart: Create a Recovery Services vault using Bicep

9/21/2022 • 2 minutes to read • Edit Online

This quickstart describes how to set up a Recovery Services vault using Bicep. The Azure Site Recovery service contributes to your business continuity and disaster recovery (BCDR) strategy so your business applications stay online during planned and unplanned outages. Site Recovery manages disaster recovery of on-premises machines and Azure virtual machines (VM), including replication, failover, and recovery.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Name of the Vault')
param vaultName string

@description('Enable CRR (Works if vault has not registered any backup instance)')
param enableCRR bool = true

@description('Change Vault Storage Type (Works if vault has not registered any backup instance)')
@allowed([
  'LocallyRedundant'
  'GeoRedundant'
])
param vaultStorageType string = 'GeoRedundant'

@description('Location for all resources.')
param location string = resourceGroup().location

var skuName = 'RS0'
var skuTier = 'Standard'

resource recoveryServicesVault 'Microsoft.RecoveryServices/vaults@2022-02-01' = {
  name: vaultName
  location: location
  sku: {
    name: skuName
    tier: skuTier
  }
  properties: {}
}

resource vaultName_vaultstorageconfig 'Microsoft.RecoveryServices/vaults/backupstorageconfig@2022-02-01' = {
  parent: recoveryServicesVault
  name: 'vaultstorageconfig'
  properties: {
    storageModelType: vaultStorageType
    crossRegionRestoreFlag: enableCRR
  }
}
```

Two Azure resources are defined in the Bicep file:

- Microsoft.RecoveryServices vaults: creates the vault.
- Microsoft.RecoveryServices/vaults/backupstorageconfig: configures the vault's backup redundancy settings.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.
   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
vaultName=<vault-name>
```

> **NOTE**
>
> Replace **<vault-name>** with the name of the vault.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use Azure CLI or Azure PowerShell to confirm that the vault was created.

- CLI
- PowerShell

```
az backup vault show --name <vault-name> --resource-group exampleRG
az backup vault backup-properties show --name <vault-name> --resource-group exampleRG
```

> **NOTE**
>
> Replace **<vault-name>** with the name of the vault you created.

Clean up resources

If you plan to use the new resources, no action is needed. Otherwise, you can remove the resource group and vault that was created in this quickstart. To delete the resource group and its resources, use Azure CLI or Azure PowerShell.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you created a Recovery Services vault using Bicep. To learn more about disaster recovery, continue to the next quickstart article.

Set up disaster recovery

Quickstart: Direct web traffic with Azure Application Gateway - Bicep

9/21/2022 • 4 minutes to read • Edit Online

In this quickstart, you use Bicep to create an Azure Application Gateway. Then you test the application gateway to make sure it works correctly.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

- An Azure account with an active subscription. Create an account for free.

Review the Bicep file

This Bicep file creates a simple setup with a public front-end IP address, a basic listener to host a single site on the application gateway, a basic request routing rule, and two virtual machines in the backend pool.

The Bicep file used in this quickstart is from Azure Quickstart Templates

```
@description('Admin username for the backend servers')
param adminUsername string

@description('Password for the admin account on the backend servers')
@secure()
param adminPassword string

@description('Location for all resources.')
param location string = resourceGroup().location

@description('Size of the virtual machine.')
param vmSize string = 'Standard_B2ms'

var virtualMachineName = 'myVM'
var virtualNetworkName = 'myVNet'
var networkInterfaceName = 'net-int'
var ipconfigName = 'ipconfig'
var publicIPAddressName = 'public_ip'
var nsgName = 'vm-nsg'
var applicationGateWayName = 'myAppGateway'
var virtualNetworkPrefix = '10.0.0.0/16'
var subnetPrefix = '10.0.0.0/24'
var backendSubnetPrefix = '10.0.1.0/24'

resource nsg 'Microsoft.Network/networkSecurityGroups@2021-05-01' = [for i in range(0, 2): {
  name: '${nsgName}${i + 1}'
  location: location
  properties: {
    securityRules: [
      {
        name: 'RDP'
        properties: {
          protocol: 'Tcp'
          sourcePortRange: '*'
          destinationPortRange: '3389'
```

```
          sourceAddressPrefix: '*'
          destinationAddressPrefix: '*'
          access: 'Allow'
          priority: 300
          direction: 'Inbound'
        }
      }
    ]
  }
}]

resource publicIPAddress 'Microsoft.Network/publicIPAddresses@2021-05-01' = [for i in range(0, 3): {
  name: '${publicIPAddressName}${i}'
  location: location
  sku: {
    name: 'Standard'
  }
  properties: {
    publicIPAddressVersion: 'IPv4'
    publicIPAllocationMethod: 'Static'
    idleTimeoutInMinutes: 4
  }
}]

resource virtualNetwork 'Microsoft.Network/virtualNetworks@2021-05-01' = {
  name: virtualNetworkName
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        virtualNetworkPrefix
      ]
    }
    subnets: [
      {
        name: 'myAGSubnet'
        properties: {
          addressPrefix: subnetPrefix
          privateEndpointNetworkPolicies: 'Enabled'
          privateLinkServiceNetworkPolicies: 'Enabled'
        }
      }
      {
        name: 'myBackendSubnet'
        properties: {
          addressPrefix: backendSubnetPrefix
          privateEndpointNetworkPolicies: 'Enabled'
          privateLinkServiceNetworkPolicies: 'Enabled'
        }
      }
    ]
    enableDdosProtection: false
    enableVmProtection: false
  }
}

resource virtualMachine 'Microsoft.Compute/virtualMachines@2021-11-01' = [for i in range(0, 2): {
  name: '${virtualMachineName}${i + 1}'
  location: location
  properties: {
    hardwareProfile: {
      vmSize: vmSize
    }
    storageProfile: {
      imageReference: {
        publisher: 'MicrosoftWindowsServer'
        offer: 'WindowsServer'
        sku: '2016-Datacenter'
        version: 'latest'
```

```
      }
      osDisk: {
        osType: 'Windows'
        createOption: 'FromImage'
        caching: 'ReadWrite'
        managedDisk: {
          storageAccountType: 'StandardSSD_LRS'
        }
        diskSizeGB: 127
      }
    }
    osProfile: {
      computerName: '${virtualMachineName}${i + 1}'
      adminUsername: adminUsername
      adminPassword: adminPassword
      windowsConfiguration: {
        provisionVMAgent: true
        enableAutomaticUpdates: true
      }
      allowExtensionOperations: true
    }
    networkProfile: {
      networkInterfaces: [
        {
          id: resourceId('Microsoft.Network/networkInterfaces', '${networkInterfaceName}${i + 1}')
        }
      ]
    }
  }
  dependsOn: [
    networkInterface
  ]
}]

resource virtualMachine_IIS 'Microsoft.Compute/virtualMachines/extensions@2021-11-01' = [for i in range(0,
2): {
  name: '${virtualMachineName}${(i + 1)}/IIS'
  location: location
  properties: {
    autoUpgradeMinorVersion: true
    publisher: 'Microsoft.Compute'
    type: 'CustomScriptExtension'
    typeHandlerVersion: '1.4'
    settings: {
      commandToExecute: 'powershell Add-WindowsFeature Web-Server; powershell Add-Content -Path
"C:\\inetpub\\wwwroot\\Default.htm" -Value $($env:computername)'
    }
  }
  dependsOn: [
    virtualMachine
  ]
}]

resource applicationGateWay 'Microsoft.Network/applicationGateways@2021-05-01' = {
  name: applicationGateWayName
  location: location
  properties: {
    sku: {
      name: 'Standard_v2'
      tier: 'Standard_v2'
    }
    gatewayIPConfigurations: [
      {
        name: 'appGatewayIpConfig'
        properties: {
          subnet: {
            id: resourceId('Microsoft.Network/virtualNetworks/subnets', virtualNetworkName, 'myAGSubnet')
          }
        }
```

```
      }
    ]
    frontendIPConfigurations: [
      {
        name: 'appGwPublicFrontendIp'
        properties: {
          privateIPAllocationMethod: 'Dynamic'
          publicIPAddress: {
            id: resourceId('Microsoft.Network/publicIPAddresses', '${publicIPAddressName}0')
          }
        }
      }
    ]
    frontendPorts: [
      {
        name: 'port_80'
        properties: {
          port: 80
        }
      }
    ]
    backendAddressPools: [
      {
        name: 'myBackendPool'
        properties: {}
      }
    ]
    backendHttpSettingsCollection: [
      {
        name: 'myHTTPSetting'
        properties: {
          port: 80
          protocol: 'Http'
          cookieBasedAffinity: 'Disabled'
          pickHostNameFromBackendAddress: false
          requestTimeout: 20
        }
      }
    ]
    httpListeners: [
      {
        name: 'myListener'
        properties: {
          frontendIPConfiguration: {
            id: resourceId('Microsoft.Network/applicationGateways/frontendIPConfigurations',
applicationGateWayName, 'appGwPublicFrontendIp')
          }
          frontendPort: {
            id: resourceId('Microsoft.Network/applicationGateways/frontendPorts', applicationGateWayName,
'port_80')
          }
          protocol: 'Http'
          requireServerNameIndication: false
        }
      }
    ]
    requestRoutingRules: [
      {
        name: 'myRoutingRule'
        properties: {
          ruleType: 'Basic'
          httpListener: {
            id: resourceId('Microsoft.Network/applicationGateways/httpListeners', applicationGateWayName,
'myListener')
          }
          backendAddressPool: {
            id: resourceId('Microsoft.Network/applicationGateways/backendAddressPools',
applicationGateWayName, 'myBackendPool')
          }
```

```
          backendHttpSettings: {
            id: resourceId('Microsoft.Network/applicationGateways/backendHttpSettingsCollection',
applicationGateWayName, 'myHTTPSetting')
          }
        }
      }
    ]
    enableHttp2: false
    autoscaleConfiguration: {
      minCapacity: 0
      maxCapacity: 10
    }
  }
  dependsOn: [
    virtualNetwork
    publicIPAddress[0]
  ]
}

resource networkInterface 'Microsoft.Network/networkInterfaces@2021-05-01' = [for i in range(0, 2): {
  name: '${networkInterfaceName}${i + 1}'
  location: location
  properties: {
    ipConfigurations: [
      {
        name: '${ipconfigName}${i + 1}'
        properties: {
          privateIPAllocationMethod: 'Dynamic'
          publicIPAddress: {
            id: resourceId('Microsoft.Network/publicIPAddresses', '${publicIPAddressName}${i + 1}')
          }
          subnet: {
            id: resourceId('Microsoft.Network/virtualNetworks/subnets', virtualNetworkName,
'myBackendSubnet')
          }
          primary: true
          privateIPAddressVersion: 'IPv4'
          applicationGatewayBackendAddressPools: [
            {
              id: resourceId('Microsoft.Network/applicationGateways/backendAddressPools',
applicationGateWayName, 'myBackendPool')
            }
          ]
        }
      }
    ]
    enableAcceleratedNetworking: false
    enableIPForwarding: false
    networkSecurityGroup: {
      id: resourceId('Microsoft.Network/networkSecurityGroups', '${nsgName}${i + 1}')
    }
  }
  dependsOn: [
    publicIPAddress
    applicationGateWay
    nsg
  ]
}]
```

Multiple Azure resources are defined in the Bicep file:

- Microsoft.Network/applicationgateways
- Microsoft.Network/publicIPAddresses : one for the application gateway, and two for the virtual machines.
- Microsoft.Network/networkSecurityGroups
- Microsoft.Network/virtualNetworks

- **Microsoft.Compute/virtualMachines** : two virtual machines
- **Microsoft.Network/networkInterfaces** : two for the virtual machines
- **Microsoft.Compute/virtualMachine/extensions** : to configure IIS and the web pages

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name myResourceGroupAG --location eastus
az deployment group create --resource-group myResourceGroupAG --template-file main.bicep --parameters
adminUsername=<admin-username>
```

> **NOTE**
>
> Replace **<admin-username>** with the admin username for the backend servers. You'll also be prompted to enter **adminPassword**.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Validate the deployment

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group myResourceGroupAG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name myResourceGroupAG
```

Next steps

Manage web traffic with an application gateway using the Azure CLI

Quickstart: Create an Azure CDN profile and endpoint - Bicep

9/21/2022 • 2 minutes to read • Edit Online

Get started with Azure Content Delivery Network (CDN) by using a Bicep file. The Bicep file deploys a profile and an endpoint.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

This Bicep file is configured to create a:

- Profile
- Endpoint

```
@description('Name of the CDN Profile')
param profileName string

@description('Name of the CDN Endpoint, must be unique')
param endpointName string

@description('Url of the origin')
param originUrl string

@description('CDN SKU names')
@allowed([
  'Standard_Akamai'
  'Standard_Microsoft'
  'Standard_Verizon'
  'Premium_Verizon'
])
param CDNSku string = 'Standard_Microsoft'

@description('Location for all resources.')
param location string = resourceGroup().location

resource profile 'Microsoft.Cdn/profiles@2021-06-01' = {
  name: profileName
  location: location
  sku: {
    name: CDNSku
  }
}

resource endpoint 'Microsoft.Cdn/profiles/endpoints@2021-06-01' = {
  parent: profile
  name: endpointName
  location: location
```

```
  properties: {
    originHostHeader: originUrl
    isHttpAllowed: true
    isHttpsAllowed: true
    queryStringCachingBehavior: 'IgnoreQueryString'
    contentTypesToCompress: [
      'application/eot'
      'application/font'
      'application/font-sfnt'
      'application/javascript'
      'application/json'
      'application/opentype'
      'application/otf'
      'application/pkcs7-mime'
      'application/truetype'
      'application/ttf'
      'application/vnd.ms-fontobject'
      'application/xhtml+xml'
      'application/xml'
      'application/xml+rss'
      'application/x-font-opentype'
      'application/x-font-truetype'
      'application/x-font-ttf'
      'application/x-httpd-cgi'
      'application/x-javascript'
      'application/x-mpegurl'
      'application/x-opentype'
      'application/x-otf'
      'application/x-perl'
      'application/x-ttf'
      'font/eot'
      'font/ttf'
      'font/otf'
      'font/opentype'
      'image/svg+xml'
      'text/css'
      'text/csv'
      'text/html'
      'text/javascript'
      'text/js'
      'text/plain'
      'text/richtext'
      'text/tab-separated-values'
      'text/xml'
      'text/x-script'
      'text/x-component'
      'text/x-java-source'
    ]
    isCompressionEnabled: true
    origins: [
      {
        name: 'origin1'
        properties: {
          hostName: originUrl
        }
      }
    ]
  }
}
```

One Azure resource is defined in the Bicep file:

- Microsoft.Cdn/profiles

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.

2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
profileName=<profile-name> endpointName=<endpoint-name> originURL=<origin-url>
```

> **NOTE**
>
> Replace **<profile-name>** with the name of the CDN profile. Replace **<endpoint-name>** with a unique CDN Endpoint name. Replace **<origin-url>** with the URL of the origin.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group. Verify that an Endpoint and CDN profile were created in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you created a:

- CDN Profile
- Endpoint

To learn more about Azure CDN, continue to the article below.

Tutorial: Use CDN to serve static content from a web app

Quickstart: Create an Azure DDoS Protection Standard using Bicep

9/21/2022 • 2 minutes to read • Edit Online

This quickstart describes how to use Bicep to create a distributed denial of service (DDoS) protection plan and virtual network (VNet), then enable the protection plan for the VNet. An Azure DDoS Protection Standard plan defines a set of virtual networks that have DDoS protection enabled across subscriptions. You can configure one DDoS protection plan for your organization and link virtual networks from multiple subscriptions to the same plan.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Specify a DDoS protection plan name.')
param ddosProtectionPlanName string

@description('Specify a DDoS virtual network name.')
param virtualNetworkName string

@description('Specify a location for the resources.')
param location string = resourceGroup().location

@description('Specify the virtual network address prefix')
param vnetAddressPrefix string = '172.17.0.0/16'

@description('Specify the virtual network subnet prefix')
param subnetPrefix string = '172.17.0.0/24'

@description('Enable DDoS protection plan.')
param ddosProtectionPlanEnabled bool = true

resource ddosProtectionPlan 'Microsoft.Network/ddosProtectionPlans@2021-05-01' = {
  name: ddosProtectionPlanName
  location: location
}

resource virtualNetwork 'Microsoft.Network/virtualNetworks@2021-05-01' = {
  name: virtualNetworkName
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        vnetAddressPrefix
      ]
    }
    subnets: [
      {
        name: 'default'
        properties: {
          addressPrefix: subnetPrefix
        }
      }
    ]
    enableDdosProtection: ddosProtectionPlanEnabled
    ddosProtectionPlan: {
      id: ddosProtectionPlan.id
    }
  }
}
```

The Bicep file defines two resources:

- Microsoft.Network/ddosProtectionPlans
- Microsoft.Network/virtualNetworks

Deploy the Bicep file

In this example, the Bicep file creates a new resource group, a DDoS protection plan, and a VNet.

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.
   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
ddosProtectionPlanName=<plan-name> virtualNetworkName=<network-name>
```

> **NOTE**
>
> Replace **<plan-name>** with a DDoS protection plan name. Replace **<network-name>** with a DDoS virtual network name.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

To learn how to view and configure telemetry for your DDoS protection plan, continue to the tutorials.

View and configure DDoS protection telemetry

Quickstart: Create an Azure DNS zone and record using Bicep

9/21/2022 • 2 minutes to read • Edit Online

This quickstart describes how to use Bicep to create a DNS zone with an `A` record in it.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

In this quickstart, you'll create a unique DNS zone with a suffix of `azurequickstart.org`. An `A` record pointing to two IP addresses will also be placed in the zone.

```
@description('The name of the DNS zone to be created.  Must have at least 2 segments, e.g. hostname.org')
param zoneName string = '${uniqueString(resourceGroup().id)}.azurequickstart.org'

@description('The name of the DNS record to be created.  The name is relative to the zone, not the FQDN.')
param recordName string = 'www'

resource zone 'Microsoft.Network/dnsZones@2018-05-01' = {
  name: zoneName
  location: 'global'
}

resource record 'Microsoft.Network/dnsZones/A@2018-05-01' = {
  parent: zone
  name: recordName
  properties: {
    TTL: 3600
    ARecords: [
      {
        ipv4Address: '1.2.3.4'
      }
      {
        ipv4Address: '1.2.3.5'
      }
    ]
  }
}

output nameServers array = zone.properties.nameServers
```

Two Azure resources have been defined in the Bicep file:

- **Microsoft.Network/dnsZones**
- **Microsoft.Network/dnsZones/A**: Used to create an `A` record in the zone.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

   ```
   az group create --name exampleRG --location eastus
   az deployment group create --resource-group exampleRG --template-file main.bicep
   ```

   When the deployment finishes, you should see a message indicating the deployment succeeded.

Validate the deployment

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you created a:

- DNS zone
- `A` record

Quickstart: Create an ExpressRoute circuit with private peering using Bicep

9/21/2022 • 4 minutes to read • Edit Online

This quickstart describes how to use Bicep to create an ExpressRoute circuit with private peering.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

In this quickstart, you'll create an ExpressRoute circuit with *Equinix* as the service provider. The circuit will be using a *Premium SKU*, with a bandwidth of *50 Mbps*, and the peering location of *Washington DC*. Private peering will be enabled with a primary and secondary subnet of *192.168.10.16/30* and *192.168.10.20/30* respectively. A virtual network will also be created along with a *HighPerformance ExpressRoute gateway*.

```
@description('Location for all resources deployed in the Bicep file')
param location string = resourceGroup().location

@description('ExpressRoute peering location')
param erpeeringLocation string = 'Washington DC'

@description('Name of the ExpressRoute circuit')
param erCircuitName string = 'er-ckt01'

@description('Name of the ExpressRoute provider')
param serviceProviderName string = 'Equinix'

@description('Tier ExpressRoute circuit')
@allowed([
  'Premium'
  'Standard'
])
param erSKU_Tier string = 'Premium'

@description('Billing model ExpressRoute circuit')
@allowed([
  'MeteredData'
  'UnlimitedData'
])
param erSKU_Family string = 'MeteredData'

@description('Bandwidth ExpressRoute circuit')
@allowed([
  50
  100
  200
  500
  1000
  2000
```

```
  5000
  10000
])
param bandwidthInMbps int = 50

@description('autonomous system number used to create private peering between the customer edge router and
MSEE routers')
param peerASN int = 65001

@description('point-to-point network prefix of primary link between the customer edge router and MSEE
router')
param primaryPeerAddressPrefix string = '192.168.10.16/30'

@description('point-to-point network prefix of secondary link between the customer edge router and MSEE
router')
param secondaryPeerAddressPrefix string = '192.168.10.20/30'

@description('VLAN Id used between the customer edge routers and MSEE routers. primary and secondary link
have the same VLAN Id')
param vlanId int = 100

@description('name of the Virtual Network')
param vnetName string = 'vnet1'

@description('name of the subnet')
param subnet1Name string = 'subnet1'

@description('address space assigned to the Virtual Network')
param vnetAddressSpace string = '10.10.10.0/24'

@description('network prefix assigned to the subnet')
param subnet1Prefix string = '10.10.10.0/25'

@description('network prefixes assigned to the gateway subnet. It has to be a network prefix with mask /27
or larger')
param gatewaySubnetPrefix string = '10.10.10.224/27'

@description('name of the ExpressRoute Gateway')
param gatewayName string = 'er-gw'

@description('ExpressRoute Gateway SKU')
@allowed([
  'Standard'
  'HighPerformance'
  'UltraPerformance'
  'ErGw1AZ'
  'ErGw2AZ'
  'ErGw3AZ'
])
param gatewaySku string = 'HighPerformance'

var erSKU_Name = '${erSKU_Tier}_${erSKU_Family}'
var gatewayPublicIPName = '${gatewayName}-pubIP'
var nsgName = 'nsg'

resource erCircuit 'Microsoft.Network/expressRouteCircuits@2021-05-01' = {
  name: erCircuitName
  location: location
  sku: {
    name: erSKU_Name
    tier: erSKU_Tier
    family: erSKU_Family
  }
  properties: {
    serviceProviderProperties: {
      serviceProviderName: serviceProviderName
      peeringLocation: erpeeringLocation
      bandwidthInMbps: bandwidthInMbps
    }
```

```
    allowClassicOperations: false
  }
}

resource epeering 'Microsoft.Network/expressRouteCircuits/peerings@2021-05-01' = {
  parent: erCircuit
  name: 'AzurePrivatePeering'
  properties: {
    peeringType: 'AzurePrivatePeering'
    peerASN: peerASN
    primaryPeerAddressPrefix: primaryPeerAddressPrefix
    secondaryPeerAddressPrefix: secondaryPeerAddressPrefix
    vlanId: vlanId
  }
}

resource nsg 'Microsoft.Network/networkSecurityGroups@2021-05-01' = {
  name: nsgName
  location: location
  properties: {
    securityRules: [
      {
        name: 'SSH-rule'
        properties: {
          description: 'allow SSH'
          protocol: 'Tcp'
          sourcePortRange: '*'
          destinationPortRange: '22'
          sourceAddressPrefix: '*'
          destinationAddressPrefix: 'VirtualNetwork'
          access: 'Allow'
          priority: 500
          direction: 'Inbound'
        }
      }
      {
        name: 'RDP-rule'
        properties: {
          description: 'allow RDP'
          protocol: 'Tcp'
          sourcePortRange: '*'
          destinationPortRange: '3389'
          sourceAddressPrefix: '*'
          destinationAddressPrefix: 'VirtualNetwork'
          access: 'Allow'
          priority: 600
          direction: 'Inbound'
        }
      }
    ]
  }
}

resource vnet 'Microsoft.Network/virtualNetworks@2021-05-01' = {
  name: vnetName
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        vnetAddressSpace
      ]
    }
    subnets: [
      {
        name: subnet1Name
        properties: {
          addressPrefix: subnet1Prefix
          networkSecurityGroup: {
            id: nsg.id
```

```
          }
        }
      }
      {
        name: 'GatewaySubnet'
        properties: {
          addressPrefix: gatewaySubnetPrefix
        }
      }
    ]
  }
}

resource publicIP 'Microsoft.Network/publicIPAddresses@2021-05-01' = {
  name: gatewayPublicIPName
  location: location
  properties: {
    publicIPAllocationMethod: 'Dynamic'
  }
}

resource gateway 'Microsoft.Network/virtualNetworkGateways@2021-05-01' = {
  name: gatewayName
  location: location
  properties: {
    ipConfigurations: [
      {
        properties: {
          privateIPAllocationMethod: 'Dynamic'
          subnet: {
            id: resourceId('Microsoft.Network/virtualNetworks/subnets', vnetName, 'GatewaySubnet')
          }
          publicIPAddress: {
            id: publicIP.id
          }
        }
        name: 'gwIPconf'
      }
    ]
    gatewayType: 'ExpressRoute'
    sku: {
      name: gatewaySku
      tier: gatewaySku
    }
    vpnType: 'RouteBased'
  }
  dependsOn: [
    vnet
  ]
}

output erCircuitName string = erCircuitName
output gatewayName string = gatewayName
output gatewaySku string = gatewaySku
```

Multiple Azure resources have been defined in the Bicep file:

- Microsoft.Network/expressRouteCircuits
- Microsoft.Network/expressRouteCircuits/peerings (Used to enabled private peering on the circuit)
- Microsoft.Network/networkSecurityGroups (network security group is applied to the subnets in the virtual network)
- Microsoft.Network/virtualNetworks
- Microsoft.Network/publicIPAddresses (Public IP is used by the ExpressRoute gateway)
- Microsoft.Network/virtualNetworkGateways (ExpressRoute gateway is used to link VNet to the circuit)

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

   ```
   az group create --name exampleRG --location eastus
   az deployment group create --resource-group exampleRG --template-file main.bicep
   ```

   When the deployment finishes, you should see a message indicating the deployment succeeded.

Validate the deployment

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

> **NOTE**
> You will need to call the provider to complete the provisioning process before you can link the virtual network to the circuit.

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the VM and all of the resources in the resource group.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you created a:

- ExpressRoute circuit
- Virtual Network
- VPN Gateway
- Public IP
- Network security group

To learn how to link a virtual network to a circuit, continue to the ExpressRoute tutorials.

Quickstart: Deploy Azure Firewall with Availability Zones - Bicep

9/21/2022 • 5 minutes to read • Edit Online

In this quickstart, you use Bicep to deploy an Azure Firewall in three Availability Zones.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

The Bicep file creates a test network environment with a firewall. The network has one virtual network (VNet) with three subnets: *AzureFirewallSubnet*, *ServersSubnet*, and *JumpboxSubnet*. The *ServersSubnet* and *JumpboxSubnet* subnet each have a single, two-core Windows Server virtual machine.

The firewall is in the *AzureFirewallSubnet* subnet, and has an application rule collection with a single rule that allows access to `www.microsoft.com`.

A user-defined route points network traffic from the *ServersSubnet* subnet through the firewall, where the firewall rules are applied.

For more information about Azure Firewall, see Deploy and configure Azure Firewall using the Azure portal.

Prerequisites

- An Azure account with an active subscription. Create an account for free.

Review the Bicep file

This Bicep file creates an Azure Firewall with Availability Zones, along with the necessary resources to support the Azure Firewall.

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('virtual network name')
param virtualNetworkName string = 'test-vnet'

@description('Location for all resources.')
param location string = resourceGroup().location

@description('Username for the Virtual Machine.')
param adminUsername string

@description('Password for the Virtual Machine.')
@secure()
param adminPassword string

@description('Availability zone numbers e.g. 1,2,3.')
param availabilityZones array = [
  '1'
  '2'
  '3'
]

@description('Number of public IP addresses for the Azure Firewall')
@minValue(1)
@maxValue(100)
```

```
param numberOfFirewallPublicIPAddresses int = 1

@description('Size of the virtual machine.')
param jumpBoxSize string = 'Standard_D2s_v3'

@description('Size of the virtual machine.')
param serverSize string = 'Standard_D2s_v3'

var vnetAddressPrefix = '10.0.0.0/16'
var serversSubnetPrefix = '10.0.2.0/24'
var azureFirewallSubnetPrefix = '10.0.1.0/24'
var jumpboxSubnetPrefix = '10.0.0.0/24'
var nextHopIP = '10.0.1.4'
var azureFirewallSubnetName = 'AzureFirewallSubnet'
var jumpBoxSubnetName = 'JumpboxSubnet'
var serversSubnetName = 'ServersSubnet'
var jumpBoxPublicIPAddressName = 'JumpHostPublicIP'
var jumpBoxNsgName = 'JumpHostNSG'
var jumpBoxNicName = 'JumpHostNic'
var jumpBoxSubnetId = resourceId('Microsoft.Network/virtualNetworks/subnets', virtualNetworkName,
jumpBoxSubnetName)
var serverNicName = 'ServerNic'
var serverSubnetId = resourceId('Microsoft.Network/virtualNetworks/subnets', virtualNetworkName,
serversSubnetName)
var storageAccountName = '${uniqueString(resourceGroup().id)}sajumpbox'
var azfwRouteTableName = 'AzfwRouteTable'
var firewallName = 'firewall1'
var publicIPNamePrefix = 'publicIP'
var azureFirewallSubnetId = resourceId('Microsoft.Network/virtualNetworks/subnets', virtualNetworkName,
azureFirewallSubnetName)
var azureFirewallSubnetJSON = json('{"id": "${azureFirewallSubnetId}"}')
var networkSecurityGroupName = '${serversSubnetName}-nsg'
var azureFirewallIpConfigurations = [for i in range(0, numberOfFirewallPublicIPAddresses): {
  name: 'IpConf${i}'
  properties: {
    subnet: ((i == 0) ? azureFirewallSubnetJSON : json('null'))
    publicIPAddress: {
      id: publicIPAddress[i].id
    }
  }
}]

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-08-01' = {
  name: storageAccountName
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'Storage'
  properties: {}
}

resource azfwRouteTable 'Microsoft.Network/routeTables@2021-03-01' = {
  name: azfwRouteTableName
  location: location
  properties: {
    disableBgpRoutePropagation: false
    routes: [
      {
        name: 'AzfwDefaultRoute'
        properties: {
          addressPrefix: '0.0.0.0/0'
          nextHopType: 'VirtualAppliance'
          nextHopIpAddress: nextHopIP
        }
      }
    ]
  }
}
```

```
resource nsg 'Microsoft.Network/networkSecurityGroups@2021-03-01' = {
  name: networkSecurityGroupName
  location: location
  properties: {}
}

resource virtualNetwork 'Microsoft.Network/virtualNetworks@2021-05-01' = {
  name: virtualNetworkName
  location: location
  tags: {
    displayName: virtualNetworkName
  }
  properties: {
    addressSpace: {
      addressPrefixes: [
        vnetAddressPrefix
      ]
    }
    subnets: [
      {
        name: jumpBoxSubnetName
        properties: {
          addressPrefix: jumpboxSubnetPrefix
        }
      }
      {
        name: azureFirewallSubnetName
        properties: {
          addressPrefix: azureFirewallSubnetPrefix
        }
      }
      {
        name: serversSubnetName
        properties: {
          addressPrefix: serversSubnetPrefix
          routeTable: {
            id: azfwRouteTable.id
          }
          networkSecurityGroup: {
            id: nsg.id
          }
        }
      }
    ]
  }
}

resource publicIPAddress 'Microsoft.Network/publicIPAddresses@2021-03-01' = [for i in range(0,
numberOfFirewallPublicIPAddresses): {
  name: '${publicIPNamePrefix}${i+1}'
  location: location
  sku: {
    name: 'Standard'
  }
  properties: {
    publicIPAllocationMethod: 'Static'
    publicIPAddressVersion: 'IPv4'
  }
  zones: availabilityZones
}]

resource jumpBoxPublicIPAddress 'Microsoft.Network/publicIPAddresses@2021-03-01' = {
  name: jumpBoxPublicIPAddressName
  location: location
  properties: {
    publicIPAllocationMethod: 'Dynamic'
  }
}
```

```
resource jumpBoxNsg 'Microsoft.Network/networkSecurityGroups@2021-05-01' = {
  name: jumpBoxNsgName
  location: location
  properties: {
    securityRules: [
      {
        name: 'myNetworkSecurityGroupRuleRDP'
        properties: {
          protocol: 'Tcp'
          sourcePortRange: '*'
          destinationPortRange: '3389'
          sourceAddressPrefix: '*'
          destinationAddressPrefix: '*'
          access: 'Allow'
          priority: 1000
          direction: 'Inbound'
        }
      }
    ]
  }
}

resource JumpBoxNic 'Microsoft.Network/networkInterfaces@2021-05-01' = {
  name: jumpBoxNicName
  location: location
  properties: {
    ipConfigurations: [
      {
        name: 'ipconfig1'
        properties: {
          privateIPAllocationMethod: 'Dynamic'
          publicIPAddress: {
            id: jumpBoxPublicIPAddress.id
          }
          subnet: {
            id: jumpBoxSubnetId
          }
        }
      }
    ]
    networkSecurityGroup: {
      id: jumpBoxNsg.id
    }
  }
  dependsOn: [
    virtualNetwork
  ]
}

resource ServerNic 'Microsoft.Network/networkInterfaces@2021-05-01' = {
  name: serverNicName
  location: location
  properties: {
    ipConfigurations: [
      {
        name: 'ipconfig1'
        properties: {
          privateIPAllocationMethod: 'Dynamic'
          subnet: {
            id: serverSubnetId
          }
        }
      }
    ]
  }
  dependsOn: [
    virtualNetwork
  ]
```

```
  ]
}

resource JumpBoxVM 'Microsoft.Compute/virtualMachines@2021-11-01' = {
  name: 'JumpBox'
  location: location
  properties: {
    hardwareProfile: {
      vmSize: jumpBoxSize
    }
    storageProfile: {
      imageReference: {
        publisher: 'MicrosoftWindowsServer'
        offer: 'WindowsServer'
        sku: '2019-Datacenter'
        version: 'latest'
      }
      osDisk: {
        osType: 'Windows'
        createOption: 'FromImage'
        diskSizeGB: 127
      }
    }
    osProfile: {
      computerName: 'JumpBox'
      adminUsername: adminUsername
      adminPassword: adminPassword
    }
    networkProfile: {
      networkInterfaces: [
        {
          id: JumpBoxNic.id
        }
      ]
    }
    diagnosticsProfile: {
      bootDiagnostics: {
        enabled: true
        storageUri: storageAccount.properties.primaryEndpoints.blob
      }
    }
  }
}

resource ServerVM 'Microsoft.Compute/virtualMachines@2021-11-01' = {
  name: 'Server'
  location: location
  properties: {
    hardwareProfile: {
      vmSize: serverSize
    }
    storageProfile: {
      imageReference: {
        publisher: 'MicrosoftWindowsServer'
        offer: 'WindowsServer'
        sku: '2019-Datacenter'
        version: 'latest'
      }
      osDisk: {
        osType: 'Windows'
        createOption: 'FromImage'
        diskSizeGB: 127
      }
    }
    osProfile: {
      computerName: 'Server'
      adminUsername: adminUsername
      adminPassword: adminPassword
    }
    networkProfile: {
```

```
    networkProfile: {
      networkInterfaces: [
        {
          id: ServerNic.id
        }
      ]
    }
    diagnosticsProfile: {
      bootDiagnostics: {
        enabled: true
        storageUri: storageAccount.properties.primaryEndpoints.blob
      }
    }
  }
}

resource firewall 'Microsoft.Network/azureFirewalls@2021-05-01' = {
  name: firewallName
  location: location
  zones: ((length(availabilityZones) == 0) ? json('null') : availabilityZones)
  properties: {
    ipConfigurations: azureFirewallIpConfigurations
    applicationRuleCollections: [
      {
        name: 'appRc1'
        properties: {
          priority: 101
          action: {
            type: 'Allow'
          }
          rules: [
            {
              name: 'appRule1'
              protocols: [
                {
                  port: 80
                  protocolType: 'Http'
                }
                {
                  port: 443
                  protocolType: 'Https'
                }
              ]
              targetFqdns: [
                'www.microsoft.com'
              ]
              sourceAddresses: [
                '10.0.2.0/24'
              ]
            }
          ]
        }
      }
    ]
    networkRuleCollections: [
      {
        name: 'netRc1'
        properties: {
          priority: 200
          action: {
            type: 'Allow'
          }
          rules: [
            {
              name: 'netRule1'
              protocols: [
                'TCP'
              ]
              sourceAddresses: [
                '10.0.2.0/24'
```

```
                '10.0.2.0/24'
              ]
              destinationAddresses: [
                '*'
              ]
              destinationPorts: [
                '8000-8999'
              ]
            }
          ]
        }
      }
    ]
  }
  dependsOn: [
    virtualNetwork
    publicIPAddress
  ]
}
```

Multiple Azure resources are defined in the Bicep file:

- Microsoft.Storage/storageAccounts
- Microsoft.Network/routeTables
- Microsoft.Network/networkSecurityGroups
- Microsoft.Network/virtualNetworks
- Microsoft.Network/publicIPAddresses
- Microsoft.Network/networkInterfaces
- Microsoft.Compute/virtualMachines
- Microsoft.Network/azureFirewalls

Deploy the Bicep file

1. Save the Bicep file as `main.bicep` to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
adminUsername=<admin-user>
```

> **NOTE**
>
> Replace **<admin-user>** with the administrator login username for the virtual machine. You'll be prompted to enter **adminPassword**.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to validate the deployment and review the deployed resources.

- CLI

- PowerShell

```
az resource list --resource-group exampleRG
```

To learn about the syntax and properties for a firewall in a Bicep file, see Microsoft.Network/azureFirewalls.

Clean up resources

When you no longer need them, use the Azure portal, Azure CLI, or Azure PowerShell to remove the resource group, firewall, and all related resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

Next, you can monitor the Azure Firewall logs.

Tutorial: Monitor Azure Firewall logs

Quickstart: Create an Azure Firewall and IP Groups - Bicep

9/21/2022 • 5 minutes to read • Edit Online

In this quickstart, you use a Bicep file to deploy an Azure Firewall with sample IP Groups used in a network rule and application rule. An IP Group is a top-level resource that allows you to define and group IP addresses, ranges, and subnets into a single object. IP Group is useful for managing IP addresses in Azure Firewall rules. You can either manually enter IP addresses or import them from a file.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

- An Azure account with an active subscription. Create an account for free.

Review the Bicep file

This Bicep file creates an Azure Firewall and IP Groups, along with the necessary resources to support the Azure Firewall.

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('virtual network name')
param virtualNetworkName string = 'vnet${uniqueString(resourceGroup().id)}'
param ipgroups_name1 string = 'ipgroup1${uniqueString(resourceGroup().id)}'
param ipgroups_name2 string = 'ipgroup2${uniqueString(resourceGroup().id)}'

@description('Username for the Virtual Machine.')
param adminUsername string

@description('Location for all resources.')
param location string = resourceGroup().location

@description('Zone numbers e.g. 1,2,3.')
param vmSize string = 'Standard_D2s_v3'

@description('Number of public IP addresses for the Azure Firewall')
@minValue(1)
@maxValue(100)
param numberOfFirewallPublicIPAddresses int = 1

@description('Type of authentication to use on the Virtual Machine. SSH key is recommended.')
@allowed([
  'sshPublicKey'
  'password'
])
param authenticationType string = 'sshPublicKey'

@description('SSH Key or password for the Virtual Machine. SSH key is recommended.')
@secure()
param adminPasswordOrKey string

var vnetAddressPrefix = '10.0.0.0/16'
var serversSubnetPrefix = '10.0.2.0/24'
var azureFirewallSubnetPrefix = '10.0.1.0/24'
```

```
var jumpboxSubnetPrefix = '10.0.0.0/24'
var nextHopIP = '10.0.1.4'
var azureFirewallSubnetName = 'AzureFirewallSubnet'
var jumpBoxSubnetName = 'JumpboxSubnet'
var serversSubnetName = 'ServersSubnet'
var jumpBoxPublicIPAddressName = 'JumpHostPublicIP'
var jumpBoxNsgName = 'JumpHostNSG'
var jumpBoxNicName = 'JumpHostNic'
var jumpBoxSubnetId = resourceId('Microsoft.Network/virtualNetworks/subnets', virtualNetworkName,
jumpBoxSubnetName)
var serverNicName = 'ServerNic'
var serverSubnetId = resourceId('Microsoft.Network/virtualNetworks/subnets', virtualNetworkName,
serversSubnetName)
var storageAccountName = '${uniqueString(resourceGroup().id)}sajumpbox'
var azfwRouteTableName = 'AzfwRouteTable'
var firewallName = 'firewall1'
var publicIPNamePrefix = 'publicIP'
var azureFirewallSubnetId = resourceId('Microsoft.Network/virtualNetworks/subnets', virtualNetworkName,
azureFirewallSubnetName)
var linuxConfiguration = {
  disablePasswordAuthentication: true
  ssh: {
    publicKeys: [
      {
        path: '/home/${adminUsername}/.ssh/authorized_keys'
        keyData: adminPasswordOrKey
      }
    ]
  }
}
var networkSecurityGroupName = '${serversSubnetName}-nsg'
var azureFirewallIpConfigurations = [for i in range(0, numberOfFirewallPublicIPAddresses): {
  name: 'IpConf${i}'
  properties: {
    subnet: {
      id: (i == 0) ? azureFirewallSubnetId : null
    }
    publicIPAddress: {
      id: publicIP[i].id
    }
  }
}]

resource ipgroup1 'Microsoft.Network/ipGroups@2021-08-01' = {
  name: ipgroups_name1
  location: location
  properties: {
    ipAddresses: [
      '13.73.64.64/26'
      '13.73.208.128/25'
      '52.126.194.0/23'
    ]
  }
}

resource ipgroup2 'Microsoft.Network/ipGroups@2021-08-01' = {
  name: ipgroups_name2
  location: location
  properties: {
    ipAddresses: [
      '12.0.0.0/24'
      '13.9.0.0/24'
    ]
  }
}

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-09-01' = {
  name: storageAccountName
  location: location
```

```
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
  properties: {}
}

resource azfwRouteTable 'Microsoft.Network/routeTables@2021-08-01' = {
  name: azfwRouteTableName
  location: location
  properties: {
    disableBgpRoutePropagation: false
    routes: [
      {
        name: 'AzfwDefaultRoute'
        properties: {
          addressPrefix: '0.0.0.0/0'
          nextHopType: 'VirtualAppliance'
          nextHopIpAddress: nextHopIP
        }
      }
    ]
  }
}

resource networkSecurityGroup 'Microsoft.Network/networkSecurityGroups@2021-08-01' = {
  name: networkSecurityGroupName
  location: location
  properties: {}
}

resource virtualNetwork 'Microsoft.Network/virtualNetworks@2021-08-01' = {
  name: virtualNetworkName
  location: location
  tags: {
    displayName: virtualNetworkName
  }
  properties: {
    addressSpace: {
      addressPrefixes: [
        vnetAddressPrefix
      ]
    }
    subnets: [
      {
        name: jumpBoxSubnetName
        properties: {
          addressPrefix: jumpboxSubnetPrefix
        }
      }
      {
        name: azureFirewallSubnetName
        properties: {
          addressPrefix: azureFirewallSubnetPrefix
        }
      }
      {
        name: serversSubnetName
        properties: {
          addressPrefix: serversSubnetPrefix
          routeTable: {
            id: azfwRouteTable.id
          }
          networkSecurityGroup: {
            id: networkSecurityGroup.id
          }
        }
      }
    ]
```

```
    ]
  }
}

resource publicIP 'Microsoft.Network/publicIPAddresses@2021-08-01' = [for i in range(0,
numberOfFirewallPublicIPAddresses): {
  name: '${publicIPNamePrefix}${i + 1}'
  location: location
  sku: {
    name: 'Standard'
  }
  properties: {
    publicIPAllocationMethod: 'Static'
    publicIPAddressVersion: 'IPv4'
  }
}]

resource jumpBoxPublicIPAddress 'Microsoft.Network/publicIPAddresses@2021-08-01' = {
  name: jumpBoxPublicIPAddressName
  location: location
  properties: {
    publicIPAllocationMethod: 'Dynamic'
  }
}

resource jumpBoxNsg 'Microsoft.Network/networkSecurityGroups@2021-08-01' = {
  name: jumpBoxNsgName
  location: location
  properties: {
    securityRules: [
      {
        name: 'myNetworkSecurityGroupRuleSSH'
        properties: {
          protocol: 'Tcp'
          sourcePortRange: '*'
          destinationPortRange: '22'
          sourceAddressPrefix: '*'
          destinationAddressPrefix: '*'
          access: 'Allow'
          priority: 1000
          direction: 'Inbound'
        }
      }
    ]
  }
}

resource JumpBoxNic 'Microsoft.Network/networkInterfaces@2021-08-01' = {
  name: jumpBoxNicName
  location: location
  properties: {
    ipConfigurations: [
      {
        name: 'ipconfig1'
        properties: {
          privateIPAllocationMethod: 'Dynamic'
          publicIPAddress: {
            id: jumpBoxPublicIPAddress.id
          }
          subnet: {
            id: jumpBoxSubnetId
          }
        }
      }
    ]
    networkSecurityGroup: {
      id: jumpBoxNsg.id
    }
  }
```

```
  dependsOn: [
    virtualNetwork
  ]
}

resource ServerNic 'Microsoft.Network/networkInterfaces@2021-08-01' = {
  name: serverNicName
  location: location
  properties: {
    ipConfigurations: [
      {
        name: 'ipconfig1'
        properties: {
          privateIPAllocationMethod: 'Dynamic'
          subnet: {
            id: serverSubnetId
          }
        }
      }
    ]
  }
  dependsOn: [
    virtualNetwork
  ]
}

resource JumpBoxVm 'Microsoft.Compute/virtualMachines@2022-03-01' = {
  name: 'JumpBox'
  location: location
  properties: {
    hardwareProfile: {
      vmSize: vmSize
    }
    storageProfile: {
      imageReference: {
        publisher: 'Canonical'
        offer: 'UbuntuServer'
        sku: '18.04-LTS'
        version: 'latest'
      }
      osDisk: {
        createOption: 'FromImage'
      }
    }
    osProfile: {
      computerName: 'JumpBox'
      adminUsername: adminUsername
      adminPassword: adminPasswordOrKey
      linuxConfiguration: ((authenticationType == 'password') ? json('null') : linuxConfiguration)
    }
    networkProfile: {
      networkInterfaces: [
        {
          id: JumpBoxNic.id
        }
      ]
    }
    diagnosticsProfile: {
      bootDiagnostics: {
        enabled: true
        storageUri: storageAccount.properties.primaryEndpoints.blob
      }
    }
  }
}

resource ServerVm 'Microsoft.Compute/virtualMachines@2022-03-01' = {
  name: 'Server'
  location: location
```

```
  properties: {
    hardwareProfile: {
      vmSize: vmSize
    }
    storageProfile: {
      imageReference: {
        publisher: 'Canonical'
        offer: 'UbuntuServer'
        sku: '18.04-LTS'
        version: 'latest'
      }
      osDisk: {
        createOption: 'FromImage'
      }
    }
    osProfile: {
      computerName: 'Server'
      adminUsername: adminUsername
      adminPassword: adminPasswordOrKey
      linuxConfiguration: ((authenticationType == 'password') ? json('null') : linuxConfiguration)
    }
    networkProfile: {
      networkInterfaces: [
        {
          id: ServerNic.id
        }
      ]
    }
    diagnosticsProfile: {
      bootDiagnostics: {
        enabled: true
        storageUri: storageAccount.properties.primaryEndpoints.blob
      }
    }
  }
}

resource firewall 'Microsoft.Network/azureFirewalls@2021-08-01' = {
  name: firewallName
  location: location
  dependsOn: [
    virtualNetwork
    publicIP
  ]
  properties: {
    ipConfigurations: azureFirewallIpConfigurations
    applicationRuleCollections: [
      {
        name: 'appRc1'
        properties: {
          priority: 101
          action: {
            type: 'Allow'
          }
          rules: [
            {
              name: 'someAppRule'
              protocols: [
                {
                  protocolType: 'Http'
                  port: 8080
                }
              ]
              targetFqdns: [
                '*bing.com'
              ]
              sourceIpGroups: [
                ipgroup1.id
              ]
```

```
            }
            {
              name: 'someOtherAppRule'
              protocols: [
                {
                  protocolType: 'Mssql'
                  port: 1433
                }
              ]
              targetFqdns: [
                'sql1${environment().suffixes.sqlServerHostname}'
              ]
              sourceIpGroups: [
                ipgroup1.id
                ipgroup2.id
              ]
            }
          ]
        }
      }
    ]
    networkRuleCollections: [
      {
        name: 'netRc1'
        properties: {
          priority: 200
          action: {
            type: 'Allow'
          }
          rules: [
            {
              name: 'networkRule'
              description: 'desc1'
              protocols: [
                'UDP'
                'TCP'
                'ICMP'
              ]
              sourceAddresses: [
                '10.0.0.0'
                '111.1.0.0/23'
              ]
              sourceIpGroups: [
                ipgroup1.id
              ]
              destinationIpGroups: [
                ipgroup2.id
              ]
              destinationPorts: [
                '90'
              ]
            }
          ]
        }
      }
    ]
  }
}
```

Multiple Azure resources are defined in the Bicep file:

- Microsoft.Network/ipGroups
- Microsoft.Storage/storageAccounts
- Microsoft.Network/routeTables
- Microsoft.Network/networkSecurityGroups

- Microsoft.Network/virtualNetworks
- Microsoft.Network/publicIPAddresses
- Microsoft.Network/networkInterfaces
- Microsoft.Compute/virtualMachines
- Microsoft.Network/azureFirewalls

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep
```

You'll be prompted to enter the following values:

- **Admin Username**: Type username for the administrator user account
- **Admin Password**: Type an administrator password or key

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to validate the deployment and review the deployed resources.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

To learn about the Bicep syntax and properties for a firewall in a Bicep file, see Microsoft.Network azureFirewalls template reference.

Clean up resources

When you no longer need them, use the Azure portal, Azure CLI, or Azure PowerShell to remove the resource group, firewall, and all related resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

Tutorial: Deploy and configure Azure Firewall in a hybrid network using the Azure portal

Quickstart: Create an Azure Firewall with multiple public IP addresses - Bicep

9/21/2022 • 4 minutes to read • Edit Online

In this quickstart, you use a Bicep file to deploy an Azure Firewall with multiple public IP addresses from a public IP address prefix. The deployed firewall has NAT rule collection rules that allow RDP connections to two Windows Server 2019 virtual machines.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

For more information about Azure Firewall with multiple public IP addresses, see Deploy an Azure Firewall with multiple public IP addresses using Azure PowerShell.

Prerequisites

- An Azure account with an active subscription. Create an account for free.

Review the Bicep file

This Bicep file creates an Azure Firewall with two public IP addresses, along with the necessary resources to support the Azure Firewall.

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Admin username for the backend servers')
param adminUsername string

@description('Password for the admin account on the backend servers')
@secure()
param adminPassword string

@description('Location for all resources.')
param location string = resourceGroup().location

@description('Size of the virtual machine.')
param vmSize string = 'Standard_B2ms'

var virtualMachineName = 'myVM'
var virtualNetworkName = 'myVNet'
var networkInterfaceName = 'net-int'
var ipConfigName = 'ipconfig'
var ipPrefixName = 'public_ip_prefix'
var ipPrefixSize = 31
var publicIpAddressName = 'public_ip'
var nsgName = 'vm-nsg'
var firewallName = 'FW-01'
var vnetPrefix = '10.0.0.0/16'
var fwSubnetPrefix = '10.0.0.0/24'
var backendSubnetPrefix = '10.0.1.0/24'
var azureFirewallSubnetId = subnet.id
var azureFirewallIpConfigurations = [for i in range(0, 2): {
  name: 'IpConf${(i + 1)}'
  properties: {
    subnet: ((i == 0) ? json('{"id": "${azureFirewallSubnetId}"}') : json('null'))
    publicIPAddress: {
```

```
      id: publicIPAddress[i].id
    }
  }
}]

resource nsg 'Microsoft.Network/networkSecurityGroups@2022-01-01' = [for i in range(0, 2): {
  name: '${nsgName}${i + 1}'
  location: location
  properties: {
    securityRules: [
      {
        name: 'RDP'
        properties: {
          protocol: 'Tcp'
          sourcePortRange: '*'
          destinationPortRange: '3389'
          sourceAddressPrefix: '*'
          destinationAddressPrefix: '*'
          access: 'Allow'
          priority: 300
          direction: 'Inbound'
        }
      }
    ]
  }
}]

resource ipprefix 'Microsoft.Network/publicIPPrefixes@2022-01-01' = {
  name: ipPrefixName
  location: location
  properties: {
    prefixLength: ipPrefixSize
    publicIPAddressVersion: 'IPv4'
  }
  sku: {
    name: 'Standard'
  }
}

resource publicIPAddress 'Microsoft.Network/publicIPAddresses@2022-01-01' = [for i in range(0, 2): {
  name: '${publicIpAddressName}${i + 1}'
  location: location
  sku: {
    name: 'Standard'
  }
  properties: {
    publicIPAddressVersion: 'IPv4'
    publicIPAllocationMethod: 'Static'
    publicIPPrefix: {
      id: ipprefix.id
    }
    idleTimeoutInMinutes: 4
  }
}]

resource virtualNetwork 'Microsoft.Network/virtualNetworks@2022-01-01' = {
  name: virtualNetworkName
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        vnetPrefix
      ]
    }
    subnets: [
      {
        name: 'myBackendSubnet'
        properties: {
          addressPrefix: backendSubnetPrefix
```

```
          addressPrefix: backendSubnetPrefix
          routeTable: {
            id: routeTable.id
          }
          privateEndpointNetworkPolicies: 'Enabled'
          privateLinkServiceNetworkPolicies: 'Enabled'
        }
      }
    ]
    enableDdosProtection: false
    enableVmProtection: false
  }
}

resource subnet 'Microsoft.Network/virtualNetworks/subnets@2022-01-01' = {
  parent: virtualNetwork
  name: 'AzureFirewallSubnet'
  properties: {
    addressPrefix: fwSubnetPrefix
    privateEndpointNetworkPolicies: 'Enabled'
    privateLinkServiceNetworkPolicies: 'Enabled'
  }
}

resource virtualMachine 'Microsoft.Compute/virtualMachines@2022-03-01' = [for i in range(0, 2): {
  name: '${virtualMachineName}${i+1}'
  location: location
  properties: {
    hardwareProfile: {
      vmSize: vmSize
    }
    storageProfile: {
      imageReference: {
        publisher: 'MicrosoftWindowsServer'
        offer: 'WindowsServer'
        sku: '2019-Datacenter'
        version: 'latest'
      }
      osDisk: {
        osType: 'Windows'
        createOption: 'FromImage'
        caching: 'ReadWrite'
        managedDisk: {
          storageAccountType: 'StandardSSD_LRS'
        }
        diskSizeGB: 127
      }
    }
    osProfile: {
      computerName: '${virtualMachineName}${i+1}'
      adminUsername: adminUsername
      adminPassword: adminPassword
      windowsConfiguration: {
        provisionVMAgent: true
        enableAutomaticUpdates: true
      }
      allowExtensionOperations: true
    }
    networkProfile: {
      networkInterfaces: [
        {
          id: netInterface[i].id
        }
      ]
    }
  }
}]

resource netInterface 'Microsoft.Network/networkInterfaces@2022-01-01' = [for i in range(0, 2): {
  name: '${networkInterfaceName}${i+1}'
```

```
  name: '${networkInterfaceName}${i + 1}'
  location: location
  properties: {
    ipConfigurations: [
      {
        name: '${ipConfigName}${i + 1}'
        properties: {
          subnet: {
            id: virtualNetwork.properties.subnets[0].id
          }
          primary: true
        }
      }
    ]
    enableAcceleratedNetworking: false
    enableIPForwarding: false
    networkSecurityGroup: {
      id: nsg[i].id
    }
  }
}]

resource firewall 'Microsoft.Network/azureFirewalls@2022-01-01' = {
  name: firewallName
  location: location
  properties: {
    sku: {
      name: 'AZFW_VNet'
      tier: 'Standard'
    }
    threatIntelMode: 'Alert'
    ipConfigurations: azureFirewallIpConfigurations
    applicationRuleCollections: [
      {
        name: 'web'
        properties: {
          priority: 100
          action: {
            type: 'Allow'
          }
          rules: [
            {
              name: 'wan-address'
              protocols: [
                {
                  protocolType: 'Http'
                  port: 80
                }
                {
                  protocolType: 'Https'
                  port: 443
                }
              ]
              targetFqdns: [
                'getmywanip.com'
              ]
              sourceAddresses: [
                '*'
              ]
            }
            {
              name: 'google'
              protocols: [
                {
                  protocolType: 'Http'
                  port: 80
                }
                {
                  protocolType: 'Https'
```

```
                port: 443
              }
            ]
            targetFqdns: [
              'www.google.com'
            ]
            sourceAddresses: [
              '10.0.1.0/24'
            ]
          }
          {
            name: 'wupdate'
            protocols: [
              {
                protocolType: 'Http'
                port: 80
              }
              {
                protocolType: 'Https'
                port: 443
              }
            ]
            fqdnTags: [
              'WindowsUpdate'
            ]
            sourceAddresses: [
              '*'
            ]
          }
        ]
      }
    }
  ]
  natRuleCollections: [
    {
      name: 'Coll-01'
      properties: {
        priority: 100
        action: {
          type: 'Dnat'
        }
        rules: [
          {
            name: 'rdp-01'
            protocols: [
              'TCP'
            ]
            translatedAddress: '10.0.1.4'
            translatedPort: '3389'
            sourceAddresses: [
              '*'
            ]
            destinationAddresses: [
              publicIPAddress[0].properties.ipAddress
            ]
            destinationPorts: [
              '3389'
            ]
          }
          {
            name: 'rdp-02'
            protocols: [
              'TCP'
            ]
            translatedAddress: '10.0.1.5'
            translatedPort: '3389'
            sourceAddresses: [
              '*'
            ]
```

```
                destinationAddresses: [
                  publicIPAddress[1].properties.ipAddress
                ]
                destinationPorts: [
                  '3389'
                ]
              }
            ]
          }
        }
      ]
    }
  }

  resource routeTable 'Microsoft.Network/routeTables@2022-01-01' = {
    name: 'rt-01'
    location: location
    properties: {
      disableBgpRoutePropagation: false
      routes: [
        {
          name: 'fw'
          properties: {
            addressPrefix: '0.0.0.0/0'
            nextHopType: 'VirtualAppliance'
            nextHopIpAddress: '10.0.0.4'
          }
        }
      ]
    }
  }
```

Multiple Azure resources are defined in the template:

- Microsoft.Network/networkSecurityGroups
- Microsoft.Network/publicIPPrefix
- Microsoft.Network/publicIPAddresses
- Microsoft.Network/virtualNetworks
- Microsoft.Compute/virtualMachines
- Microsoft.Storage/storageAccounts
- Microsoft.Network/networkInterfaces
- Microsoft.Network/azureFirewalls
- Microsoft.Network/routeTables

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
adminUsername=<admin-username>
```

> **NOTE**
>
> Replace **<admin-username>** with the admin username for the backend server.

You will be prompt to enter the admin password.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Validate the deployment

In the Azure portal, review the deployed resources. Note the firewall public IP addresses.

Use Remote Desktop Connection to connect to the firewall public IP addresses. Successful connection demonstrates firewall NAT rules that allow the connection to the backend servers.

Clean up resources

When you no longer need the resources that you created with the firewall, delete the resource group. This removes the firewall and all the related resources.

To delete the resource group, call the `Remove-AzResourceGroup` cmdlet:

```
Remove-AzResourceGroup -Name "exampleRG"
```

Next steps

Tutorial: Deploy and configure Azure Firewall in a hybrid network using the Azure portal

Quickstart: Create an Azure Firewall and a firewall policy - Bicep

9/21/2022 • 3 minutes to read • Edit Online

In this quickstart, you use Bicep to create an Azure Firewall and a firewall policy. The firewall policy has an application rule that allows connections to `www.microsoft.com` and a rule that allows connections to Windows Update using the **WindowsUpdate** FQDN tag. A network rule allows UDP connections to a time server at 13.86.101.172.

Also, IP Groups are used in the rules to define the **Source** IP addresses.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

For information about Azure Firewall Manager, see What is Azure Firewall Manager?.

For information about Azure Firewall, see What is Azure Firewall?.

For information about IP Groups, see IP Groups in Azure Firewall.

Prerequisites

- An Azure account with an active subscription. Create an account for free.

Review the Bicep file

This Bicep file creates a hub virtual network, along with the necessary resources to support the scenario.

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Virtual network name')
param virtualNetworkName string = 'vnet${uniqueString(resourceGroup().id)}'

@description('Azure Firewall name')
param firewallName string = 'fw${uniqueString(resourceGroup().id)}'

@description('Number of public IP addresses for the Azure Firewall')
@minValue(1)
@maxValue(100)
param numberOfPublicIPAddresses int = 2

@description('Zone numbers e.g. 1,2,3.')
param availabilityZones array = []

@description('Location for all resources.')
param location string = resourceGroup().location
param infraIpGroupName string = '${location}-infra-ipgroup-${uniqueString(resourceGroup().id)}'
param workloadIpGroupName string = '${location}-workload-ipgroup-${uniqueString(resourceGroup().id)}'
param firewallPolicyName string = '${firewallName}-firewallPolicy'

var vnetAddressPrefix = '10.10.0.0/24'
var azureFirewallSubnetPrefix = '10.10.0.0/25'
var publicIPNamePrefix = 'publicIP'
var azurepublicIpname = publicIPNamePrefix
var azureFirewallSubnetName = 'AzureFirewallSubnet'
var azureFirewallSubnetId = resourceId('Microsoft.Network/virtualNetworks/subnets', virtualNetworkName,
```

```
azureFirewallSubnetName)
var azureFirewallPublicIpId = resourceId('Microsoft.Network/publicIPAddresses', publicIPNamePrefix)
var azureFirewallIpConfigurations = [for i in range(0, numberOfPublicIPAddresses): {
  name: 'IpConf${i}'
  properties: {
    subnet: ((i == 0) ? json('{"id": "${azureFirewallSubnetId}"}') : json('null'))
    publicIPAddress: {
      id: '${azureFirewallPublicIpId}${i + 1}'
    }
  }
}]

resource workloadIpGroup 'Microsoft.Network/ipGroups@2022-01-01' = {
  name: workloadIpGroupName
  location: location
  properties: {
    ipAddresses: [
      '10.20.0.0/24'
      '10.30.0.0/24'
    ]
  }
}

resource infraIpGroup 'Microsoft.Network/ipGroups@2022-01-01' = {
  name: infraIpGroupName
  location: location
  properties: {
    ipAddresses: [
      '10.40.0.0/24'
      '10.50.0.0/24'
    ]
  }
}

resource vnet 'Microsoft.Network/virtualNetworks@2022-01-01' = {
  name: virtualNetworkName
  location: location
  tags: {
    displayName: virtualNetworkName
  }
  properties: {
    addressSpace: {
      addressPrefixes: [
        vnetAddressPrefix
      ]
    }
    subnets: [
      {
        name: azureFirewallSubnetName
        properties: {
          addressPrefix: azureFirewallSubnetPrefix
        }
      }
    ]
    enableDdosProtection: false
  }
}

resource publicIpAddress 'Microsoft.Network/publicIPAddresses@2022-01-01' = [for i in range(0,
numberOfPublicIPAddresses): {
  name: '${azurepublicIpname}${i + 1}'
  location: location
  sku: {
    name: 'Standard'
  }
  properties: {
    publicIPAllocationMethod: 'Static'
    publicIPAddressVersion: 'IPv4'
  }
```

```
}]

resource firewallPolicy 'Microsoft.Network/firewallPolicies@2022-01-01'= {
  name: firewallPolicyName
  location: location
  properties: {
    threatIntelMode: 'Alert'
  }
}

resource networkRuleCollectionGroup 'Microsoft.Network/firewallPolicies/ruleCollectionGroups@2022-01-01' = {
  parent: firewallPolicy
  name: 'DefaultNetworkRuleCollectionGroup'
  properties: {
    priority: 200
    ruleCollections: [
      {
        ruleCollectionType: 'FirewallPolicyFilterRuleCollection'
        action: {
          type: 'Allow'
        }
        name: 'azure-global-services-nrc'
        priority: 1250
        rules: [
          {
            ruleType: 'NetworkRule'
            name: 'time-windows'
            ipProtocols: [
              'UDP'
            ]
            destinationAddresses: [
              '13.86.101.172'
            ]
            sourceIpGroups: [
              workloadIpGroup.id
              infraIpGroup.id
            ]
            destinationPorts: [
              '123'
            ]
          }
        ]
      }
    ]
  }
}

resource applicationRuleCollectionGroup 'Microsoft.Network/firewallPolicies/ruleCollectionGroups@2022-01-01'
= {
  parent: firewallPolicy
  name: 'DefaultApplicationRuleCollectionGroup'
  dependsOn: [
    networkRuleCollectionGroup
  ]
  properties: {
    priority: 300
    ruleCollections: [
      {
        ruleCollectionType: 'FirewallPolicyFilterRuleCollection'
        name: 'global-rule-url-arc'
        priority: 1000
        action: {
          type: 'Allow'
        }
        rules: [
          {
            ruleType: 'ApplicationRule'
            name: 'winupdate-rule-01'
            protocols: [
```

```
            protocols: [
              {
                protocolType: 'Https'
                port: 443
              }
              {
                protocolType: 'Http'
                port: 80
              }
            ]
            fqdnTags: [
              'WindowsUpdate'
            ]
            terminateTLS: false
            sourceIpGroups: [
              workloadIpGroup.id
              infraIpGroup.id
            ]
          }
        ]
      }
      {
        ruleCollectionType: 'FirewallPolicyFilterRuleCollection'
        action: {
          type: 'Allow'
        }
        name: 'Global-rules-arc'
        priority: 1202
        rules: [
          {
            ruleType: 'ApplicationRule'
            name: 'global-rule-01'
            protocols: [
              {
                protocolType: 'Https'
                port: 443
              }
            ]
            targetFqdns: [
              'www.microsoft.com'
            ]
            terminateTLS: false
            sourceIpGroups: [
              workloadIpGroup.id
              infraIpGroup.id
            ]
          }
        ]
      }
    ]
  }
}

resource firewall 'Microsoft.Network/azureFirewalls@2021-03-01' = {
  name: firewallName
  location: location
  zones: ((length(availabilityZones) == 0) ? null : availabilityZones)
  dependsOn: [
    vnet
    publicIpAddress
    workloadIpGroup
    infraIpGroup
    networkRuleCollectionGroup
    applicationRuleCollectionGroup
  ]
  properties: {
    ipConfigurations: azureFirewallIpConfigurations
    firewallPolicy: {
      id: firewallPolicy.id
```

```
    }
  }
}
```

Multiple Azure resources are defined in the Bicep file:

- Microsoft.Network/ipGroups
- Microsoft.Network/firewallPolicies
- Microsoft.Network/firewallPolicies/ruleCollectionGroups
- Microsoft.Network/azureFirewalls
- Microsoft.Network/virtualNetworks
- Microsoft.Network/publicIPAddresses

Deploy the Bicep file

1. Save the Bicep file as `main.bicep` to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
firewallName=<firewall-name>
```

> **NOTE**
>
> Replace **<firewall-name>** with the name of the Azure Firewall.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use Azure CLI or Azure PowerShell to review the deployed resources.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When you no longer need the resources that you created with the firewall, delete the resource group. This removes the firewall and all the related resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

Azure Firewall Manager policy overview

Quickstart: Secure your virtual hub using Azure Firewall Manager - Bicep

9/21/2022 • 5 minutes to read • Edit Online

In this quickstart, you use Bicep to secure your virtual hub using Azure Firewall Manager. The deployed firewall has an application rule that allows connections to `www.microsoft.com` . Two Windows Server 2019 virtual machines are deployed to test the firewall. One jump server is used to connect to the workload server. From the workload server, you can only connect to `www.microsoft.com`.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

For more information about Azure Firewall Manager, see What is Azure Firewall Manager?.

Prerequisites

- An Azure account with an active subscription. Create an account for free.

Review the Bicep file

This Bicep file creates a secured virtual hub using Azure Firewall Manager, along with the necessary resources to support the scenario.

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Admin username for the servers')
param adminUsername string

@description('Password for the admin account on the servers')
@secure()
param adminPassword string

@description('Location for all resources.')
param location string = resourceGroup().location

@description('Size of the virtual machine.')
param vmSize string = 'Standard_D2_v3'

resource virtualWan 'Microsoft.Network/virtualWans@2021-08-01' = {
  name: 'VWan-01'
  location: location
  properties: {
    disableVpnEncryption: false
    allowBranchToBranchTraffic: true
    type: 'Standard'
  }
}

resource virtualHub 'Microsoft.Network/virtualHubs@2021-08-01' = {
  name: 'Hub-01'
  location: location
  properties: {
    addressPrefix: '10.1.0.0/16'
    virtualWan: {
      id: virtualWan.id
    }
```

```
    }
  }
}

resource hubVNetconnection 'Microsoft.Network/virtualHubs/hubVirtualNetworkConnections@2021-08-01' = {
  parent: virtualHub
  name: 'hub-spoke'
  dependsOn: [
    firewall
  ]
  properties: {
    remoteVirtualNetwork: {
      id: virtualNetwork.id
    }
    allowHubToRemoteVnetTransit: true
    allowRemoteVnetToUseHubVnetGateways: false
    enableInternetSecurity: true
    routingConfiguration: {
      associatedRouteTable: {
        id: hubRouteTable.id
      }
      propagatedRouteTables: {
        labels: [
          'VNet'
        ]
        ids: [
          {
            id: hubRouteTable.id
          }
        ]
      }
    }
  }
}

resource policy 'Microsoft.Network/firewallPolicies@2021-08-01' = {
  name: 'Policy-01'
  location: location
  properties: {
    threatIntelMode: 'Alert'
  }
}

resource ruleCollectionGroup 'Microsoft.Network/firewallPolicies/ruleCollectionGroups@2021-08-01' = {
  parent: policy
  name: 'DefaultApplicationRuleCollectionGroup'
  properties: {
    priority: 300
    ruleCollections: [
      {
        ruleCollectionType: 'FirewallPolicyFilterRuleCollection'
        name: 'RC-01'
        priority: 100
        action: {
          type: 'Allow'
        }
        rules: [
          {
            ruleType: 'ApplicationRule'
            name: 'Allow-msft'
            sourceAddresses: [
              '*'
            ]
            protocols: [
              {
                port: 80
                protocolType: 'Http'
              }
              {
                port: 443
```

```
                port: 443
                protocolType: 'Https'
              }
            ]
            targetFqdns: [
              '*.microsoft.com'
            ]
          }
        ]
      }
    ]
  }
}

resource firewall 'Microsoft.Network/azureFirewalls@2021-08-01' = {
  name: 'AzfwTest'
  location: location
  properties: {
    sku: {
      name: 'AZFW_Hub'
      tier: 'Standard'
    }
    hubIPAddresses: {
      publicIPs: {
        count: 1
      }
    }
    virtualHub: {
      id: virtualHub.id
    }
    firewallPolicy: {
      id: policy.id
    }
  }
}

resource virtualNetwork 'Microsoft.Network/virtualNetworks@2021-08-01' = {
  name: 'Spoke-01'
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        '10.0.0.0/16'
      ]
    }
    enableDdosProtection: false
    enableVmProtection: false
  }
}

resource subnet_Workload_SN 'Microsoft.Network/virtualNetworks/subnets@2021-08-01' = {
  parent: virtualNetwork
  name: 'Workload-SN'
  properties: {
    addressPrefix: '10.0.1.0/24'
    privateEndpointNetworkPolicies: 'Enabled'
    privateLinkServiceNetworkPolicies: 'Enabled'
  }
}

resource subnet_Jump_SN 'Microsoft.Network/virtualNetworks/subnets@2021-08-01' = {
  parent: virtualNetwork
  name: 'Jump-SN'
  dependsOn: [
    subnet_Workload_SN
  ]
  properties: {
    addressPrefix: '10.0.2.0/24'
    routeTable: {
```

```
      id: routeTable.id
    }
    privateEndpointNetworkPolicies: 'Enabled'
    privateLinkServiceNetworkPolicies: 'Enabled'
  }
}

resource Jump_Srv 'Microsoft.Compute/virtualMachines@2022-03-01' = {
  name: 'Jump-Srv'
  location: location
  properties: {
    hardwareProfile: {
      vmSize: vmSize
    }
    storageProfile: {
      imageReference: {
        publisher: 'MicrosoftWindowsServer'
        offer: 'WindowsServer'
        sku: '2019-Datacenter'
        version: 'latest'
      }
      osDisk: {
        osType: 'Windows'
        createOption: 'FromImage'
        caching: 'ReadWrite'
        managedDisk: {
          storageAccountType: 'StandardSSD_LRS'
        }
        diskSizeGB: 127
      }
    }
    osProfile: {
      computerName: 'Jump-Srv'
      adminUsername: adminUsername
      adminPassword: adminPassword
      windowsConfiguration: {
        provisionVMAgent: true
        enableAutomaticUpdates: true
      }
      allowExtensionOperations: true
    }
    networkProfile: {
      networkInterfaces: [
        {
          id: netInterface_jump_srv.id
        }
      ]
    }
  }
}

resource Workload_Srv 'Microsoft.Compute/virtualMachines@2022-03-01' = {
  name: 'Workload-Srv'
  location: location
  properties: {
    hardwareProfile: {
      vmSize: vmSize
    }
    storageProfile: {
      imageReference: {
        publisher: 'MicrosoftWindowsServer'
        offer: 'WindowsServer'
        sku: '2019-Datacenter'
        version: 'latest'
      }
      osDisk: {
        osType: 'Windows'
        createOption: 'FromImage'
        caching: 'ReadWrite'
```

```
        managedDisk: {
          storageAccountType: 'StandardSSD_LRS'
        }
        diskSizeGB: 127
      }
    }
    osProfile: {
      computerName: 'Workload-Srv'
      adminUsername: adminUsername
      adminPassword: adminPassword
      windowsConfiguration: {
        provisionVMAgent: true
        enableAutomaticUpdates: true
      }
      allowExtensionOperations: true
    }
    networkProfile: {
      networkInterfaces: [
        {
          id: netInterface_workload_srv.id
        }
      ]
    }
  }
}

resource netInterface_workload_srv 'Microsoft.Network/networkInterfaces@2021-08-01' = {
  name: 'netInterface-workload-srv'
  location: location
  properties: {
    ipConfigurations: [
      {
        name: 'ipconfig1'
        properties: {
          privateIPAllocationMethod: 'Dynamic'
          subnet: {
            id: subnet_Workload_SN.id
          }
          primary: true
          privateIPAddressVersion: 'IPv4'
        }
      }
    ]
    enableAcceleratedNetworking: false
    enableIPForwarding: false
    networkSecurityGroup: {
      id: nsg_workload_srv.id
    }
  }
}

resource netInterface_jump_srv 'Microsoft.Network/networkInterfaces@2021-08-01' = {
  name: 'netInterface-jump-srv'
  location: location
  properties: {
    ipConfigurations: [
      {
        name: 'ipconfig1'
        properties: {
          privateIPAllocationMethod: 'Dynamic'
          publicIPAddress: {
            id: publicIP_jump_srv.id
          }
          subnet: {
            id: subnet_Jump_SN.id
          }
          primary: true
          privateIPAddressVersion: 'IPv4'
        }
```

```
      }
    ]
    enableAcceleratedNetworking: false
    enableIPForwarding: false
    networkSecurityGroup: {
      id: nsg_jump_srv.id
    }
  }
}

resource nsg_jump_srv 'Microsoft.Network/networkSecurityGroups@2021-08-01' = {
  name: 'nsg-jump-srv'
  location: location
  properties: {
    securityRules: [
      {
        name: 'RDP'
        properties: {
          protocol: 'Tcp'
          sourcePortRange: '*'
          destinationPortRange: '3389'
          sourceAddressPrefix: '*'
          destinationAddressPrefix: '*'
          access: 'Allow'
          priority: 300
          direction: 'Inbound'
        }
      }
    ]
  }
}

resource nsg_workload_srv 'Microsoft.Network/networkSecurityGroups@2021-08-01' = {
  name: 'nsg-workload-srv'
  location: location
  properties: {}
}

resource publicIP_jump_srv 'Microsoft.Network/publicIPAddresses@2021-08-01' = {
  name: 'publicIP-jump-srv'
  location: location
  sku: {
    name: 'Standard'
  }
  properties: {
    publicIPAddressVersion: 'IPv4'
    publicIPAllocationMethod: 'Static'
    idleTimeoutInMinutes: 4
  }
}

resource routeTable 'Microsoft.Network/routeTables@2021-08-01' = {
  name: 'RT-01'
  location: location
  properties: {
    disableBgpRoutePropagation: false
    routes: [
      {
        name: 'jump-to-inet'
        properties: {
          addressPrefix: '0.0.0.0/0'
          nextHopType: 'Internet'
        }
      }
    ]
  }
}

resource hubRouteTable 'Microsoft.Network/virtualHubs/hubRouteTables@2021-08-01' = {
```

```
    parent: virtualHub
    name: 'RT_VNet'
    properties: {
      routes: [
        {
          name: 'Workload-SNToFirewall'
          destinationType: 'CIDR'
          destinations: [
            '10.0.1.0/24'
          ]
          nextHopType: 'ResourceId'
          nextHop: firewall.id
        }
        {
          name: 'InternetToFirewall'
          destinationType: 'CIDR'
          destinations: [
            '0.0.0.0/0'
          ]
          nextHopType: 'ResourceId'
          nextHop: firewall.id
        }
      ]
      labels: [
        'VNet'
      ]
    }
  }
```

Multiple Azure resources are defined in the Bicep file:

- Microsoft.Network/virtualWans
- Microsoft.Network/virtualHubs
- Microsoft.Network/firewallPolicies
- Microsoft.Network/azureFirewalls
- Microsoft.Network/virtualNetworks
- Microsoft.Compute/virtualMachines
- Microsoft.Storage/storageAccounts
- Microsoft.Network/networkInterfaces
- Microsoft.Network/networkSecurityGroups
- Microsoft.Network/publicIPAddresses
- Microsoft.Network/routeTables

Deploy the Bicep file

1. Save the Bicep file as `main.bicep` to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
adminUsername=<admin-user>
```

> **NOTE**
>
> Replace **<admin-user>** with the administrator login username for the servers. You'll be prompted to enter **adminPassword**.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Validate the deployment

Use Azure CLI or Azure PowerShell to review the deployed resources.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Now, test the firewall rules to confirm that it works as expected.

1. From the Azure portal, review the network settings for the **Workload-Srv** virtual machine and note the private IP address.
2. Connect a remote desktop to **Jump-Srv** virtual machine, and sign in. From there, open a remote desktop connection to the **Workload-Srv** private IP address.
3. Open Internet Explorer and browse to `www.microsoft.com`.
4. Select **OK** > **Close** on the Internet Explorer security alerts.

   You should see the Microsoft home page.
5. Browse to `www.google.com`.

   You should be blocked by the firewall.

Now you've verified that the firewall rules are working, you can browse to the one allowed FQDN, but not to any others.

Clean up resources

When you no longer need the resources that you created with the firewall, use Azure portal, Azure CLI, or Azure PowerShell to delete the resource group. This removes the firewall and all the related resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

Learn about security partner providers

Quickstart: Create a Front Door using Bicep

9/21/2022 • 2 minutes to read • Edit Online

This quickstart describes how to use Bicep to create a Front Door to set up high availability for a web endpoint.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin.
- IP or FQDN of a website or web application.

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

In this quickstart, you'll create a Front Door configuration with a single backend and a single default path matching `/*`.

```
@description('The name of the frontdoor resource.')
param frontDoorName string

@description('The hostname of the backend. Must be an IP address or FQDN.')
param backendAddress string

var frontEndEndpointName = 'frontEndEndpoint'
var loadBalancingSettingsName = 'loadBalancingSettings'
var healthProbeSettingsName = 'healthProbeSettings'
var routingRuleName = 'routingRule'
var backendPoolName = 'backendPool'

resource frontDoor 'Microsoft.Network/frontDoors@2020-05-01' = {
  name: frontDoorName
  location: 'global'
  properties: {
    enabledState: 'Enabled'

    frontendEndpoints: [
      {
        name: frontEndEndpointName
        properties: {
          hostName: '${frontDoorName}.azurefd.net'
          sessionAffinityEnabledState: 'Disabled'
        }
      }
    ]

    loadBalancingSettings: [
      {
        name: loadBalancingSettingsName
        properties: {
          sampleSize: 4
          successfulSamplesRequired: 2
        }
      }
    ]
```

```
    healthProbeSettings: [
      {
        name: healthProbeSettingsName
        properties: {
          path: '/'
          protocol: 'Http'
          intervalInSeconds: 120
        }
      }
    ]

    backendPools: [
      {
        name: backendPoolName
        properties: {
          backends: [
            {
              address: backendAddress
              backendHostHeader: backendAddress
              httpPort: 80
              httpsPort: 443
              weight: 50
              priority: 1
              enabledState: 'Enabled'
            }
          ]
          loadBalancingSettings: {
            id: resourceId('Microsoft.Network/frontDoors/loadBalancingSettings', frontDoorName,
loadBalancingSettingsName)
          }
          healthProbeSettings: {
            id: resourceId('Microsoft.Network/frontDoors/healthProbeSettings', frontDoorName,
healthProbeSettingsName)
          }
        }
      }
    ]

    routingRules: [
      {
        name: routingRuleName
        properties: {
          frontendEndpoints: [
            {
              id: resourceId('Microsoft.Network/frontDoors/frontEndEndpoints', frontDoorName,
frontEndEndpointName)
            }
          ]
          acceptedProtocols: [
            'Http'
            'Https'
          ]
          patternsToMatch: [
            '/*'
          ]
          routeConfiguration: {
            '@odata.type': '#Microsoft.Azure.FrontDoor.Models.FrontdoorForwardingConfiguration'
            forwardingProtocol: 'MatchRequest'
            backendPool: {
              id: resourceId('Microsoft.Network/frontDoors/backEndPools', frontDoorName, backendPoolName)
            }
          }
          enabledState: 'Enabled'
        }
      }
    ]
  }
}
```

One Azure resource is defined in the Bicep file:

- **Microsoft.Network/frontDoors**

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
frontDoorName=<door-name> backendAddress=<backend-address>
```

> **NOTE**
>
> Replace **<door-name>** with the name of the Front Door resource. Replace **<backend-address>** with the hostname of the backend. It must be an IP address or FQDN.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Validate the deployment

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the Front Door service and the resource group. This removes the Front Door and all the related resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you created a Front Door.

To learn how to add a custom domain to your Front Door, continue to the Front Door tutorials.

Front Door tutorials

Quickstart: Create a Front Door Standard/Premium using Bicep

9/21/2022 • 3 minutes to read • Edit Online

This quickstart describes how to use Bicep to create an Azure Front Door Standard/Premium with a Web App as origin.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin.
- IP or FQDN of a website or web application.

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

In this quickstart, you'll create a Front Door Standard/Premium, an App Service, and configure the App Service to validate that traffic has come through the Front Door origin.

```
@description('The location into which regionally scoped resources should be deployed. Note that Front Door
is a global resource.')
param location string = resourceGroup().location

@description('The name of the App Service application to create. This must be globally unique.')
param appName string = 'myapp-${uniqueString(resourceGroup().id)}'

@description('The name of the SKU to use when creating the App Service plan.')
param appServicePlanSkuName string = 'S1'

@description('The number of worker instances of your App Service plan that should be provisioned.')
param appServicePlanCapacity int = 1

@description('The name of the Front Door endpoint to create. This must be globally unique.')
param frontDoorEndpointName string = 'afd-${uniqueString(resourceGroup().id)}'

@description('The name of the SKU to use when creating the Front Door profile.')
@allowed([
  'Standard_AzureFrontDoor'
  'Premium_AzureFrontDoor'
])
param frontDoorSkuName string = 'Standard_AzureFrontDoor'

var appServicePlanName = 'AppServicePlan'

var frontDoorProfileName = 'MyFrontDoor'
var frontDoorOriginGroupName = 'MyOriginGroup'
var frontDoorOriginName = 'MyAppServiceOrigin'
var frontDoorRouteName = 'MyRoute'

resource frontDoorProfile 'Microsoft.Cdn/profiles@2021-06-01' = {
  name: frontDoorProfileName
  location: 'global'
  sku: {
```

```
    name: frontDoorSkuName
  }
}

resource appServicePlan 'Microsoft.Web/serverFarms@2020-06-01' = {
  name: appServicePlanName
  location: location
  sku: {
    name: appServicePlanSkuName
    capacity: appServicePlanCapacity
  }
  kind: 'app'
}

resource app 'Microsoft.Web/sites@2020-06-01' = {
  name: appName
  location: location
  kind: 'app'
  identity: {
    type: 'SystemAssigned'
  }
  properties: {
    serverFarmId: appServicePlan.id
    httpsOnly: true
    siteConfig: {
      detailedErrorLoggingEnabled: true
      httpLoggingEnabled: true
      requestTracingEnabled: true
      ftpsState: 'Disabled'
      minTlsVersion: '1.2'
      ipSecurityRestrictions: [
        {
          tag: 'ServiceTag'
          ipAddress: 'AzureFrontDoor.Backend'
          action: 'Allow'
          priority: 100
          headers: {
            'x-azure-fdid': [
              frontDoorProfile.properties.frontDoorId
            ]
          }
          name: 'Allow traffic from Front Door'
        }
      ]
    }
  }
}

resource frontDoorEndpoint 'Microsoft.Cdn/profiles/afdEndpoints@2021-06-01' = {
  name: frontDoorEndpointName
  parent: frontDoorProfile
  location: 'global'
  properties: {
    enabledState: 'Enabled'
  }
}

resource frontDoorOriginGroup 'Microsoft.Cdn/profiles/originGroups@2021-06-01' = {
  name: frontDoorOriginGroupName
  parent: frontDoorProfile
  properties: {
    loadBalancingSettings: {
      sampleSize: 4
      successfulSamplesRequired: 3
    }
    healthProbeSettings: {
      probePath: '/'
      probeRequestType: 'HEAD'
      probeProtocol: 'Http'
```

```
      probeIntervalInSeconds: 100
    }
  }
}

resource frontDoorOrigin 'Microsoft.Cdn/profiles/originGroups/origins@2021-06-01' = {
  name: frontDoorOriginName
  parent: frontDoorOriginGroup
  properties: {
    hostName: app.properties.defaultHostName
    httpPort: 80
    httpsPort: 443
    originHostHeader: app.properties.defaultHostName
    priority: 1
    weight: 1000
  }
}

resource frontDoorRoute 'Microsoft.Cdn/profiles/afdEndpoints/routes@2021-06-01' = {
  name: frontDoorRouteName
  parent: frontDoorEndpoint
  dependsOn: [
    frontDoorOrigin // This explicit dependency is required to ensure that the origin group is not empty
when the route is created.
  ]
  properties: {
    originGroup: {
      id: frontDoorOriginGroup.id
    }
    supportedProtocols: [
      'Http'
      'Https'
    ]
    patternsToMatch: [
      '/*'
    ]
    forwardingProtocol: 'HttpsOnly'
    linkToDefaultDomain: 'Enabled'
    httpsRedirect: 'Enabled'
  }
}

output appServiceHostName string = app.properties.defaultHostName
output frontDoorEndpointHostName string = frontDoorEndpoint.properties.hostName
```

Multiple Azure resources are defined in the Bicep file:

- **Microsoft.Network/frontDoors**
- **Microsoft.Web/serverfarms** (App service plan to host web apps)
- **Microsoft.Web/sites** (Web app origin servicing request for Front Door)

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.
   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep
```

When the deployment finishes, the output is similar to:

```
DeploymentName          : main
ResourceGroupName       : exampleRG
ProvisioningState       : Succeeded
Timestamp               : 7/8/2022 6:39:08 PM
Mode                    : Incremental
TemplateLink            :
Parameters              :
                          Name                    Type                    Value
                          ======================  ======================  ==========
                          location                String                  "eastus"
                          appName                 String                  "myapp-ftq6ssulg2cts"
                          appServicePlanSkuName   String                  "S1"
                          appServicePlanCapacity  Int                     1
                          frontDoorEndpointName   String                  "afd-ftq6ssulg2cts"
                          frontDoorSkuName        String                  "Standard_AzureFrontDoor"

Outputs                 :
                          Name                      Type                    Value
                          ========================  ======================  ==========
                          appServiceHostName        String                  "myapp-ftq6ssulg2cts.azurewebsites.net"
                          frontDoorEndpointHostName String
                          "afd-ftq6ssulg2cts-a9gmeqa7avfrb0f0.z01.azurefd.net"
```

Validate the deployment

Use Azure CLI or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

You can also use the Azure portal to validate the deployment.

1. Sign in to the Azure portal.
2. Select **Resource groups** from the left pane.
3. Select the resource group that you created in the previous section.
4. Select the Front Door you created and you'll be able to see the endpoint hostname. Copy the hostname and paste it on to the address bar of a browser. Press enter and your request will automatically get routed to the web app.

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the Front Door service and the resource group. This removes the Front Door and all the related resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you created a:

- Front Door
- App Service plan
- Web App

To learn how to add a custom domain to your Front Door, continue to the Front Door tutorials.

Front Door tutorials

Quickstart: Create an internal load balancer to load balance VMs by using Bicep

9/21/2022 • 3 minutes to read • Edit Online

This quickstart describes how to use Bicep to create an internal Azure load balancer.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from the Azure Quickstart Templates.

```
@description('Admin username')
param adminUsername string

@description('Admin password')
@secure()
param adminPassword string

@description('Prefix to use for VM names')
param vmNamePrefix string = 'BackendVM'

@description('Location for all resources.')
param location string = resourceGroup().location

@description('Size of the virtual machines')
param vmSize string = 'Standard_D2s_v3'

var availabilitySetName = 'AvSet'
var storageAccountType = 'Standard_LRS'
var storageAccountName = uniqueString(resourceGroup().id)
var virtualNetworkName = 'vNet'
var subnetName = 'backendSubnet'
var loadBalancerName = 'ilb'
var networkInterfaceName = 'nic'
var subnetRef = resourceId('Microsoft.Network/virtualNetworks/subnets', virtualNetworkName, subnetName)
var numberOfInstances = 2

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-08-01' = {
  name: storageAccountName
  location: location
  sku: {
    name: storageAccountType
  }
  kind: 'StorageV2'
}

resource availabilitySet 'Microsoft.Compute/availabilitySets@2021-11-01' = {
  name: availabilitySetName
  location: location
  sku: {
    name: 'Aligned'
```

```
  }
  properties: {
    platformUpdateDomainCount: 2
    platformFaultDomainCount: 2
  }
}

resource virtualNetwork 'Microsoft.Network/virtualNetworks@2021-05-01' = {
  name: virtualNetworkName
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        '10.0.0.0/16'
      ]
    }
    subnets: [
      {
        name: subnetName
        properties: {
          addressPrefix: '10.0.2.0/24'
        }
      }
    ]
  }
}

resource networkInterface 'Microsoft.Network/networkInterfaces@2021-05-01' = [for i in range(0,
numberOfInstances): {
  name: '${networkInterfaceName}${i}'
  location: location
  properties: {
    ipConfigurations: [
      {
        name: 'ipconfig1'
        properties: {
          privateIPAllocationMethod: 'Dynamic'
          subnet: {
            id: subnetRef
          }
          loadBalancerBackendAddressPools: [
            {
              id: resourceId('Microsoft.Network/loadBalancers/backendAddressPools', loadBalancerName,
'BackendPool1')
            }
          ]
        }
      }
    ]
  }
  dependsOn: [
    virtualNetwork
    loadBalancer
  ]
}]

resource loadBalancer 'Microsoft.Network/loadBalancers@2021-05-01' = {
  name: loadBalancerName
  location: location
  sku: {
    name: 'Standard'
  }
  properties: {
    frontendIPConfigurations: [
      {
        properties: {
          subnet: {
            id: subnetRef
          }
```

```
          privateIPAddress: '10.0.2.6'
          privateIPAllocationMethod: 'Static'
        }
        name: 'LoadBalancerFrontend'
      }
    ]
    backendAddressPools: [
      {
        name: 'BackendPool1'
      }
    ]
    loadBalancingRules: [
      {
        properties: {
          frontendIPConfiguration: {
            id: resourceId('Microsoft.Network/loadBalancers/frontendIpConfigurations', loadBalancerName,
'LoadBalancerFrontend')
          }
          backendAddressPool: {
            id: resourceId('Microsoft.Network/loadBalancers/backendAddressPools', loadBalancerName,
'BackendPool1')
          }
          probe: {
            id: resourceId('Microsoft.Network/loadBalancers/probes', loadBalancerName, 'lbprobe')
          }
          protocol: 'Tcp'
          frontendPort: 80
          backendPort: 80
          idleTimeoutInMinutes: 15
        }
        name: 'lbrule'
      }
    ]
    probes: [
      {
        properties: {
          protocol: 'Tcp'
          port: 80
          intervalInSeconds: 15
          numberOfProbes: 2
        }
        name: 'lbprobe'
      }
    ]
  }
  dependsOn: [
    virtualNetwork
  ]
}

resource vm 'Microsoft.Compute/virtualMachines@2021-11-01' = [for i in range(0, numberOfInstances): {
  name: '${vmNamePrefix}${i}'
  location: location
  properties: {
    availabilitySet: {
      id: availabilitySet.id
    }
    hardwareProfile: {
      vmSize: vmSize
    }
    osProfile: {
      computerName: '${vmNamePrefix}${i}'
      adminUsername: adminUsername
      adminPassword: adminPassword
    }
    storageProfile: {
      imageReference: {
        publisher: 'MicrosoftWindowsServer'
        offer: 'WindowsServer'
```

```
        sku: '2019-Datacenter'
        version: 'latest'
      }
      osDisk: {
        createOption: 'FromImage'
      }
    }
    networkProfile: {
      networkInterfaces: [
        {
          id: networkInterface[i].id
        }
      ]
    }
    diagnosticsProfile: {
      bootDiagnostics: {
        enabled: true
        storageUri: storageAccount.properties.primaryEndpoints.blob
      }
    }
  }
}]
```

Multiple Azure resources have been defined in the Bicep file:

- **Microsoft.Storage/storageAccounts**: Virtual machine storage accounts for boot diagnostics.
- **Microsoft.Compute/availabilitySets**: Availability set for virtual machines.
- **Microsoft.Network/virtualNetworks**: Virtual network for load balancer and virtual machines.
- **Microsoft.Network/networkInterfaces**: Network interfaces for virtual machines.
- **Microsoft.Network/loadBalancers**: Internal load balancer.
- **Microsoft.Compute/virtualMachines**: Virtual machines.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

   ```
   az group create --name exampleRG --location eastus
   az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
   adminUsername=<admin-user>
   ```

   > **NOTE**
   >
   > Replace **<admin-user>** with the admin username. You'll also be prompted to enter **adminPassword**.

   When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

For a step-by-step tutorial that guides you through the process of creating a Bicep file, see:

Quickstart: Create Bicep files with Visual Studio Code

Quickstart: Create a public load balancer to load balance VMs by using a Bicep file

9/21/2022 • 6 minutes to read • Edit Online

Load balancing provides a higher level of availability and scale by spreading incoming requests across multiple virtual machines (VMs).

This quickstart shows you how to deploy a standard load balancer to load balance virtual machines.

Using a Bicep file takes fewer steps comparing to other deployment methods.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

Load balancer and public IP SKUs must match. When you create a standard load balancer, you must also create a new standard public IP address that is configured as the frontend for the standard load balancer. If you want to create a basic load balancer, use this template. Microsoft recommends using standard SKU for production workloads.

```
@description('Specifies a project name that is used for generating resource names.')
param projectName string

@description('Specifies the location for all of the resources created by this template.')
param location string = resourceGroup().location

@description('Specifies the virtual machine administrator username.')
param adminUsername string

@description('Specifies the virtual machine administrator password.')
@secure()
param adminPassword string

@description('Size of the virtual machine')
param vmSize string = 'Standard_D2s_v3'

var lbName = '${projectName}-lb'
var lbSkuName = 'Standard'
var lbPublicIpAddressName = '${projectName}-lbPublicIP'
var lbPublicIPAddressNameOutbound = '${projectName}-lbPublicIPOutbound'
var lbFrontEndName = 'LoadBalancerFrontEnd'
var lbFrontEndNameOutbound = 'LoadBalancerFrontEndOutbound'
var lbBackendPoolName = 'LoadBalancerBackEndPool'
var lbBackendPoolNameOutbound = 'LoadBalancerBackEndPoolOutbound'
var lbProbeName = 'loadBalancerHealthProbe'
var nsgName = '${projectName}-nsg'
var vNetName = '${projectName}-vnet'
var vNetAddressPrefix = '10.0.0.0/16'
var vNetSubnetName = 'BackendSubnet'
```

```
var vNetSubnetAddressPrefix = '10.0.0.0/24'
var bastionName = '${projectName}-bastion'
var bastionSubnetName = 'AzureBastionSubnet'
var vNetBastionSubnetAddressPrefix = '10.0.1.0/24'
var bastionPublicIPAddressName = '${projectName}-bastionPublicIP'
var vmStorageAccountType = 'Premium_LRS'

resource project_vm_1_networkInterface 'Microsoft.Network/networkInterfaces@2021-08-01' = [for i in range(0,
3): {
  name: '${projectName}-vm${(i + 1)}-networkInterface'
  location: location
  properties: {
    ipConfigurations: [
      {
        name: 'ipconfig1'
        properties: {
          privateIPAllocationMethod: 'Dynamic'
          subnet: {
            id: vNetName_vNetSubnetName.id
          }
          loadBalancerBackendAddressPools: [
            {
              id: resourceId('Microsoft.Network/loadBalancers/backendAddressPools', lbName,
lbBackendPoolName)
            }
            {
              id: resourceId('Microsoft.Network/loadBalancers/backendAddressPools', lbName,
lbBackendPoolNameOutbound)
            }
          ]
        }
      }
    ]
    networkSecurityGroup: {
      id: nsg.id
    }
  }
  dependsOn: [
    vNet
    lb
  ]
}]

resource project_vm_1_InstallWebServer 'Microsoft.Compute/virtualMachines/extensions@2021-11-01' = [for i in
range(0, 3): {
  name: '${projectName}-vm${(i + 1)}/InstallWebServer'
  location: location
  properties: {
    publisher: 'Microsoft.Compute'
    type: 'CustomScriptExtension'
    typeHandlerVersion: '1.7'
    autoUpgradeMinorVersion: true
    settings: {
      commandToExecute: 'powershell.exe Install-WindowsFeature -name Web-Server -IncludeManagementTools &&
powershell.exe remove-item \'C:\\inetpub\\wwwroot\\iisstart.htm\' && powershell.exe Add-Content -Path
\'C:\\inetpub\\wwwroot\\iisstart.htm\' -Value $(\'Hello World from \' + $env:computername)'
    }
  }
  dependsOn: [
    project_vm_1
  ]
}]

resource project_vm_1 'Microsoft.Compute/virtualMachines@2021-11-01' = [for i in range(1, 3): {
  name: '${projectName}-vm${i}'
  location: location
  zones: [
    string(i)
  ]
```

```
  ]
  properties: {
    hardwareProfile: {
      vmSize: vmSize
    }
    storageProfile: {
      imageReference: {
        publisher: 'MicrosoftWindowsServer'
        offer: 'WindowsServer'
        sku: '2019-Datacenter'
        version: 'latest'
      }
      osDisk: {
        createOption: 'FromImage'
        managedDisk: {
          storageAccountType: vmStorageAccountType
        }
      }
    }
    networkProfile: {
      networkInterfaces: [
        {
          id: resourceId('Microsoft.Network/networkInterfaces', '${projectName}-vm${i}-networkInterface')
        }
      ]
    }
    osProfile: {
      computerName: '${projectName}-vm${i}'
      adminUsername: adminUsername
      adminPassword: adminPassword
      windowsConfiguration: {
        enableAutomaticUpdates: true
        provisionVMAgent: true
      }
    }
  }
  dependsOn: [
    project_vm_1_networkInterface
  ]
}]

resource vNetName_bastionSubnet 'Microsoft.Network/virtualNetworks/subnets@2021-08-01' = {
  parent: vNet
  name: bastionSubnetName
  properties: {
    addressPrefix: vNetBastionSubnetAddressPrefix
  }
  dependsOn: [
    vNetName_vNetSubnetName
  ]
}

resource vNetName_vNetSubnetName 'Microsoft.Network/virtualNetworks/subnets@2021-08-01' = {
  parent: vNet
  name: vNetSubnetName
  properties: {
    addressPrefix: vNetSubnetAddressPrefix
  }
}

resource bastion 'Microsoft.Network/bastionHosts@2021-08-01' = {
  name: bastionName
  location: location
  properties: {
    ipConfigurations: [
      {
        name: 'IpConf'
        properties: {
          privateIPAllocationMethod: 'Dynamic'
```

```
          publicIPAddress: {
            id: bastionPublicIPAddress.id
          }
          subnet: {
            id: vNetName_bastionSubnet.id
          }
        }
      }
    ]
  }
}

resource bastionPublicIPAddress 'Microsoft.Network/publicIPAddresses@2021-08-01' = {
  name: bastionPublicIPAddressName
  location: location
  sku: {
    name: lbSkuName
  }
  properties: {
    publicIPAddressVersion: 'IPv4'
    publicIPAllocationMethod: 'Static'
  }
}

resource lb 'Microsoft.Network/loadBalancers@2021-08-01' = {
  name: lbName
  location: location
  sku: {
    name: lbSkuName
  }
  properties: {
    frontendIPConfigurations: [
      {
        name: lbFrontEndName
        properties: {
          publicIPAddress: {
            id: lbPublicIPAddress.id
          }
        }
      }
      {
        name: lbFrontEndNameOutbound
        properties: {
          publicIPAddress: {
            id: lbPublicIPAddressOutbound.id
          }
        }
      }
    ]
    backendAddressPools: [
      {
        name: lbBackendPoolName
      }
      {
        name: lbBackendPoolNameOutbound
      }
    ]
    loadBalancingRules: [
      {
        name: 'myHTTPRule'
        properties: {
          frontendIPConfiguration: {
            id: resourceId('Microsoft.Network/loadBalancers/frontendIPConfigurations', lbName,
lbFrontEndName)
          }
          backendAddressPool: {
            id: resourceId('Microsoft.Network/loadBalancers/backendAddressPools', lbName, lbBackendPoolName)
          }
          frontendPort: 80
```

```
          backendPort: 80
          enableFloatingIP: false
          idleTimeoutInMinutes: 15
          protocol: 'Tcp'
          enableTcpReset: true
          loadDistribution: 'Default'
          disableOutboundSnat: true
          probe: {
            id: resourceId('Microsoft.Network/loadBalancers/probes', lbName, lbProbeName)
          }
        }
      }
    ]
    probes: [
      {
        name: lbProbeName
        properties: {
          protocol: 'Tcp'
          port: 80
          intervalInSeconds: 5
          numberOfProbes: 2
        }
      }
    ]
    outboundRules: [
      {
        name: 'myOutboundRule'
        properties: {
          allocatedOutboundPorts: 10000
          protocol: 'All'
          enableTcpReset: false
          idleTimeoutInMinutes: 15
          backendAddressPool: {
            id: resourceId('Microsoft.Network/loadBalancers/backendAddressPools', lbName,
lbBackendPoolNameOutbound)
          }
          frontendIPConfigurations: [
            {
              id: resourceId('Microsoft.Network/loadBalancers/frontendIPConfigurations', lbName,
lbFrontEndNameOutbound)
            }
          ]
        }
      }
    ]
  }
}

resource lbPublicIPAddress 'Microsoft.Network/publicIPAddresses@2021-08-01' = {
  name: lbPublicIpAddressName
  location: location
  sku: {
    name: lbSkuName
  }
  properties: {
    publicIPAddressVersion: 'IPv4'
    publicIPAllocationMethod: 'Static'
  }
}

resource lbPublicIPAddressOutbound 'Microsoft.Network/publicIPAddresses@2021-08-01' = {
  name: lbPublicIPAddressNameOutbound
  location: location
  sku: {
    name: lbSkuName
  }
  properties: {
    publicIPAddressVersion: 'IPv4'
    publicIPAllocationMethod: 'Static'
```

```
    }
  }

  resource nsg 'Microsoft.Network/networkSecurityGroups@2021-08-01' = {
    name: nsgName
    location: location
    properties: {
      securityRules: [
        {
          name: 'AllowHTTPInbound'
          properties: {
            protocol: '*'
            sourcePortRange: '*'
            destinationPortRange: '80'
            sourceAddressPrefix: 'Internet'
            destinationAddressPrefix: '*'
            access: 'Allow'
            priority: 100
            direction: 'Inbound'
          }
        }
      ]
    }
  }

  resource vNet 'Microsoft.Network/virtualNetworks@2021-08-01' = {
    name: vNetName
    location: location
    properties: {
      addressSpace: {
        addressPrefixes: [
          vNetAddressPrefix
        ]
      }
    }
  }
```

Multiple Azure resources have been defined in the bicep file:

- **Microsoft.Network/loadBalancers**
- **Microsoft.Network/publicIPAddresses**: for the load balancer, bastion host, and for each of the three virtual machines.
- **Microsoft.Network/bastionHosts**
- **Microsoft.Network/networkSecurityGroups**
- **Microsoft.Network/virtualNetworks**
- **Microsoft.Compute/virtualMachines** (3).
- **Microsoft.Network/networkInterfaces** (3).
- **Microsoft.Compute/virtualMachine/extensions** (3): use to configure the Internet Information Server (IIS), and the web pages.

To find more Bicep files or ARM templates that are related to Azure Load Balancer, see Azure Quickstart Templates.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.

2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location centralus
az deployment group create --resource-group exampleRG --template-file main.bicep
```

> **NOTE**
>
> The Bicep file deployment creates three availability zones. Availability zones are supported only in certain regions. Use one of the supported regions. If you aren't sure, enter **centralus**.

You will be prompted to enter the following values:

- **projectName**: used for generating resource names.
- **adminUsername**: virtual machine administrator username.
- **adminPassword**: virtual machine administrator password.

It takes about 10 minutes to deploy the Bicep file.

Review deployed resources

1. Sign in to the Azure portal.
2. Select **Resource groups** from the left pane.
3. Select the resource group that you created in the previous section. The default resource group name is **exampleRG**.
4. Select the load balancer. Its default name is the project name with **-lb** appended.
5. Copy only the IP address part of the public IP address, and then paste it into the address bar of your browser.

The browser displays the default page of the Internet Information Services (IIS) web server.

To see the load balancer distribute traffic across all three VMs, you can force a refresh of your web browser from the client machine.

Clean up resources

When you no longer need them, delete the:

- Resource group
- Load balancer
- Related resources

Go to the Azure portal, select the resource group that contains the load balancer, and then select **Delete resource group**.

Next steps

In this quickstart, you:

- Created a virtual network for the load balancer and virtual machines.
- Created an Azure Bastion host for management.
- Created a standard load balancer and attached VMs to it.
- Configured the load-balancer traffic rule, and the health probe.
- Tested the load balancer.

To learn more, continue to the tutorials for Azure Load Balancer.

Azure Load Balancer tutorials

Quickstart: Create a NAT gateway - Bicep

9/21/2022 • 4 minutes to read • Edit Online

Get started with Virtual Network NAT using Bicep. This Bicep file deploys a virtual network, a NAT gateway resource, and Ubuntu virtual machine. The Ubuntu virtual machine is deployed to a subnet that is associated with the NAT gateway resource.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

This Bicep file is configured to create a:

- Virtual network
- NAT gateway resource
- Ubuntu virtual machine

The Ubuntu VM is deployed to a subnet that's associated with the NAT gateway resource.

```
@description('Name of the virtual machine')
param vmname string = 'myVM'

@description('Size of the virtual machine')
param vmsize string = 'Standard_D2s_v3'

@description('Name of the virtual network')
param vnetname string = 'myVnet'

@description('Name of the subnet for virtual network')
param subnetname string = 'mySubnet'

@description('Address space for virtual network')
param vnetaddressspace string = '192.168.0.0/16'

@description('Subnet prefix for virtual network')
param vnetsubnetprefix string = '192.168.0.0/24'

@description('Name of the NAT gateway')
param natgatewayname string = 'myNATgateway'

@description('Name of the virtual machine nic')
param networkinterfacename string = 'myvmNIC'

@description('Name of the NAT gateway public IP')
param publicipname string = 'myPublicIP'

@description('Name of the virtual machine NSG')
param nsgname string = 'myVMnsg'

@description('Name of the virtual machine public IP')
```

```
@description('Name of the virtual machine public IP')
param publicipvmname string = 'myPublicIPVM'

@description('Name of the NAT gateway public IP')
param publicipprefixname string = 'myPublicIPPrefix'

@description('Administrator username for virtual machine')
param adminusername string

@description('Administrator password for virtual machine')
@secure()
param adminpassword string

@description('Name of resource group')
param location string = resourceGroup().location

resource nsg 'Microsoft.Network/networkSecurityGroups@2021-05-01' = {
  name: nsgname
  location: location
  properties: {
    securityRules: [
      {
        name: 'SSH'
        properties: {
          protocol: 'Tcp'
          sourcePortRange: '*'
          destinationPortRange: '22'
          sourceAddressPrefix: '*'
          destinationAddressPrefix: '*'
          access: 'Allow'
          priority: 300
          direction: 'Inbound'
        }
      }
    ]
  }
}

resource publicip 'Microsoft.Network/publicIPAddresses@2021-05-01' = {
  name: publicipname
  location: location
  sku: {
    name: 'Standard'
  }
  properties: {
    publicIPAddressVersion: 'IPv4'
    publicIPAllocationMethod: 'Static'
    idleTimeoutInMinutes: 4
  }
}

resource publicipvm 'Microsoft.Network/publicIPAddresses@2021-05-01' = {
  name: publicipvmname
  location: location
  sku: {
    name: 'Standard'
  }
  properties: {
    publicIPAddressVersion: 'IPv4'
    publicIPAllocationMethod: 'Static'
    idleTimeoutInMinutes: 4
  }
}

resource publicipprefix 'Microsoft.Network/publicIPPrefixes@2021-05-01' = {
  name: publicipprefixname
  location: location
  sku: {
    name: 'Standard'
  }
```

```
    }
    properties: {
      prefixLength: 31
      publicIPAddressVersion: 'IPv4'
    }
  }

  resource vm 'Microsoft.Compute/virtualMachines@2021-11-01' = {
    name: vmname
    location: location
    properties: {
      hardwareProfile: {
        vmSize: vmsize
      }
      storageProfile: {
        imageReference: {
          publisher: 'Canonical'
          offer: 'UbuntuServer'
          sku: '18.04-LTS'
          version: 'latest'
        }
        osDisk: {
          osType: 'Linux'
          name: '${vmname}_disk1'
          createOption: 'FromImage'
          caching: 'ReadWrite'
          managedDisk: {
            storageAccountType: 'Premium_LRS'
          }
          diskSizeGB: 30
        }
      }
      osProfile: {
        computerName: vmname
        adminUsername: adminusername
        adminPassword: adminpassword
        linuxConfiguration: {
          disablePasswordAuthentication: false
          provisionVMAgent: true
        }
        allowExtensionOperations: true
      }
      networkProfile: {
        networkInterfaces: [
          {
            id: networkinterface.id
          }
        ]
      }
    }
  }

  resource vnet 'Microsoft.Network/virtualNetworks@2021-05-01' = {
    name: vnetname
    location: location
    properties: {
      addressSpace: {
        addressPrefixes: [
          vnetaddressspace
        ]
      }
      subnets: [
        {
          name: subnetname
          properties: {
            addressPrefix: vnetsubnetprefix
            natGateway: {
              id: natgateway.id
            }
```

```
          privateEndpointNetworkPolicies: 'Enabled'
          privateLinkServiceNetworkPolicies: 'Enabled'
        }
      }
    ]
    enableDdosProtection: false
    enableVmProtection: false
  }
}

resource natgateway 'Microsoft.Network/natGateways@2021-05-01' = {
  name: natgatewayname
  location: location
  sku: {
    name: 'Standard'
  }
  properties: {
    idleTimeoutInMinutes: 4
    publicIpAddresses: [
      {
        id: publicip.id
      }
    ]
    publicIpPrefixes: [
      {
        id: publicipprefix.id
      }
    ]
  }
}

resource mySubnet 'Microsoft.Network/virtualNetworks/subnets@2021-05-01' = {
  parent: vnet
  name: 'mySubnet'
  properties: {
    addressPrefix: vnetsubnetprefix
    natGateway: {
      id: natgateway.id
    }
    privateEndpointNetworkPolicies: 'Enabled'
    privateLinkServiceNetworkPolicies: 'Enabled'
  }
}

resource networkinterface 'Microsoft.Network/networkInterfaces@2021-05-01' = {
  name: networkinterfacename
  location: location
  properties: {
    ipConfigurations: [
      {
        name: 'ipconfig1'
        properties: {
          privateIPAddress: '192.168.0.4'
          privateIPAllocationMethod: 'Dynamic'
          publicIPAddress: {
            id: publicipvm.id
          }
          subnet: {
            id: mySubnet.id
          }
          primary: true
          privateIPAddressVersion: 'IPv4'
        }
      }
    ]
    enableAcceleratedNetworking: false
    enableIPForwarding: false
    networkSecurityGroup: {
      id: nsg.id
```

```
    }
  }
}
```

Nine Azure resources are defined in the Bicep file:

- **Microsoft.Network/networkSecurityGroups**: Creates a network security group.
- **Microsoft.Network/networkSecurityGroups/securityRules**: Creates a security rule.
- **Microsoft.Network/publicIPAddresses**: Creates a public IP address.
- **Microsoft.Network/publicIPPrefixes**: Creates a public IP prefix.
- **Microsoft.Compute/virtualMachines**: Creates a virtual machine.
- **Microsoft.Network/virtualNetworks**: Creates a virtual network.
- **Microsoft.Network/natGateways**: Creates a NAT gateway resource.
- **Microsoft.Network/virtualNetworks/subnets**: Creates a virtual network subnet.
- **Microsoft.Network/networkinterfaces**: Creates a network interface.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

   ```
   az group create --name exampleRG --location eastus
   az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
   adminusername=<admin-name>
   ```

   > **NOTE**
   > Replace **<admin-name>** with the administrator username for the virtual machine. You'll also be prompted to enter **adminpassword**.

   When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI

- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you created a:

- NAT gateway resource
- Virtual network
- Ubuntu virtual machine

The virtual machine is deployed to a virtual network subnet associated with the NAT gateway.

To learn more about Virtual Network NAT and Bicep, continue to the articles below.

- Read an Overview of Virtual Network NAT
- Read about the NAT Gateway resource
- Learn more about Bicep

Quickstart: Create a private endpoint using Bicep

9/21/2022 • 5 minutes to read • Edit Online

In this quickstart, you'll use Bicep to create a private endpoint.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

You can also create a private endpoint by using the Azure portal, Azure PowerShell, the Azure CLI, or an Azure Resource Manager Template.

Prerequisites

You need an Azure account with an active subscription. If you don't already have an Azure account, create an account for free.

Review the Bicep file

This Bicep file creates a private endpoint for an instance of Azure SQL Database.

The Bicep file that this quickstart uses is from Azure Quickstart Templates.

```
@description('The administrator username of the SQL logical server')
param sqlAdministratorLogin string

@description('The administrator password of the SQL logical server.')
@secure()
param sqlAdministratorLoginPassword string

@description('Username for the Virtual Machine.')
param vmAdminUsername string

@description('Password for the Virtual Machine. The password must be at least 12 characters long and have
lower case, upper characters, digit and a special character (Regex match)')
@secure()
param vmAdminPassword string

@description('The size of the VM')
param VmSize string = 'Standard_D2_v3'

@description('Location for all resources.')
param location string = resourceGroup().location

var vnetName = 'myVirtualNetwork'
var vnetAddressPrefix = '10.0.0.0/16'
var subnet1Prefix = '10.0.0.0/24'
var subnet1Name = 'mySubnet'
var sqlServerName = 'sqlserver${uniqueString(resourceGroup().id)}'
var databaseName = '${sqlServerName}/sample-db'
var privateEndpointName = 'myPrivateEndpoint'
var privateDnsZoneName = 'privatelink${environment().suffixes.sqlServerHostname}'
var pvtEndpointDnsGroupName = '${privateEndpointName}/mydnsgroupname'
var vmName = take('myVm${uniqueString(resourceGroup().id)}', 15)
var publicIpAddressName = '${vmName}PublicIP'
var networkInterfaceName = '${vmName}NetInt'
var osDiskType = 'StandardSSD_LRS'

resource sqlServer 'Microsoft.Sql/servers@2021-11-01-preview' = {
```

```
  name: sqlServerName
  location: location
  tags: {
    displayName: sqlServerName
  }
  properties: {
    administratorLogin: sqlAdministratorLogin
    administratorLoginPassword: sqlAdministratorLoginPassword
    version: '12.0'
    publicNetworkAccess: 'Disabled'
  }
}

resource database 'Microsoft.Sql/servers/databases@2021-11-01-preview' = {
  name: databaseName
  location: location
  sku: {
    name: 'Basic'
    tier: 'Basic'
    capacity: 5
  }
  tags: {
    displayName: databaseName
  }
  properties: {
    collation: 'SQL_Latin1_General_CP1_CI_AS'
    maxSizeBytes: 104857600
    sampleName: 'AdventureWorksLT'
  }
  dependsOn: [
    sqlServer
  ]
}

resource vnet 'Microsoft.Network/virtualNetworks@2021-05-01' = {
  name: vnetName
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        vnetAddressPrefix
      ]
    }
  }
}

resource subnet 'Microsoft.Network/virtualNetworks/subnets@2021-05-01' = {
  parent: vnet
  name: subnet1Name
  properties: {
    addressPrefix: subnet1Prefix
    privateEndpointNetworkPolicies: 'Disabled'
  }
}

resource privateEndpoint 'Microsoft.Network/privateEndpoints@2021-05-01' = {
  name: privateEndpointName
  location: location
  properties: {
    subnet: {
      id: subnet.id
    }
    privateLinkServiceConnections: [
      {
        name: privateEndpointName
        properties: {
          privateLinkServiceId: sqlServer.id
          groupIds: [
            'sqlServer'
```

```
            sqlServer
          ]
        }
      }
    ]
  }
  dependsOn: [
    vnet
  ]
}

resource privateDnsZone 'Microsoft.Network/privateDnsZones@2020-06-01' = {
  name: privateDnsZoneName
  location: 'global'
  properties: {}
  dependsOn: [
    vnet
  ]
}

resource privateDnsZoneLink 'Microsoft.Network/privateDnsZones/virtualNetworkLinks@2020-06-01' = {
  parent: privateDnsZone
  name: '${privateDnsZoneName}-link'
  location: 'global'
  properties: {
    registrationEnabled: false
    virtualNetwork: {
      id: vnet.id
    }
  }
}

resource pvtEndpointDnsGroup 'Microsoft.Network/privateEndpoints/privateDnsZoneGroups@2021-05-01' = {
  name: pvtEndpointDnsGroupName
  properties: {
    privateDnsZoneConfigs: [
      {
        name: 'config1'
        properties: {
          privateDnsZoneId: privateDnsZone.id
        }
      }
    ]
  }
  dependsOn: [
    privateEndpoint
  ]
}

resource publicIpAddress 'Microsoft.Network/publicIPAddresses@2021-05-01' = {
  name: publicIpAddressName
  location: location
  tags: {
    displayName: publicIpAddressName
  }
  properties: {
    publicIPAllocationMethod: 'Dynamic'
  }
}

resource networkInterface 'Microsoft.Network/networkInterfaces@2021-05-01' = {
  name: networkInterfaceName
  location: location
  tags: {
    displayName: networkInterfaceName
  }
  properties: {
    ipConfigurations: [
      {
        name: 'ipConfig1'
```

```
        name: 'ipconfig1'
        properties: {
          privateIPAllocationMethod: 'Dynamic'
          publicIPAddress: {
            id: publicIpAddress.id
          }
          subnet: {
            id: subnet.id
          }
        }
      }
    ]
  }
  dependsOn: [
    vnet
  ]
}

resource vm 'Microsoft.Compute/virtualMachines@2021-11-01' = {
  name: vmName
  location: location
  tags: {
    displayName: vmName
  }
  properties: {
    hardwareProfile: {
      vmSize: VmSize
    }
    osProfile: {
      computerName: vmName
      adminUsername: vmAdminUsername
      adminPassword: vmAdminPassword
    }
    storageProfile: {
      imageReference: {
        publisher: 'MicrosoftWindowsServer'
        offer: 'WindowsServer'
        sku: '2019-Datacenter'
        version: 'latest'
      }
      osDisk: {
        name: '${vmName}OsDisk'
        caching: 'ReadWrite'
        createOption: 'FromImage'
        managedDisk: {
          storageAccountType: osDiskType
        }
        diskSizeGB: 128
      }
    }
    networkProfile: {
      networkInterfaces: [
        {
          id: networkInterface.id
        }
      ]
    }
  }
}
```

The Bicep file defines multiple Azure resources:

- **Microsoft.Sql/servers**: The instance of SQL Database with the sample database.
- **Microsoft.Sql/servers/databases**: The sample database.
- **Microsoft.Network/virtualNetworks**: The virtual network where the private endpoint is deployed.
- **Microsoft.Network/privateEndpoints**: The private endpoint that you use to access the instance of SQL

Database.

- **Microsoft.Network/privateDnsZones**: The zone that you use to resolve the private endpoint IP address.
- **Microsoft.Network/privateDnsZones/virtualNetworkLinks**
- **Microsoft.Network/privateEndpoints/privateDnsZoneGroups**: The zone group that you use to associate the private endpoint with a private DNS zone.
- **Microsoft.Network/publicIpAddresses**: The public IP address that you use to access the virtual machine.
- **Microsoft.Network/networkInterfaces**: The network interface for the virtual machine.
- **Microsoft.Compute/virtualMachines**: The virtual machine that you use to test the connection of the private endpoint to the instance of SQL Database.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.

2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
sqlAdministratorLogin=<admin-login> vmAdminUsername=<vm-login>
```

> **NOTE**
> Replace **<admin-login>** with the username for the SQL logical server. Replace **<vm-login>** with the username for the virtual machine. You'll be prompted to enter **sqlAdministratorLoginPassword**. You'll also be prompted to enter **vmAdminPassword**, which must be at least 12 characters long and contain at least one lowercase and uppercase character and one special character.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Validate the deployment

> **NOTE**
> The Bicep file generates a unique name for the virtual machine myVm**{uniqueid}** resource, and for the SQL Database sqlserver**{uniqueid}** resource. Substitute your generated value for **{uniqueid}**.

Connect to a VM from the internet

Connect to the VM *myVm{uniqueid}* from the internet by doing the following:

1. In the Azure portal search bar, enter *myVm{uniqueid}*.

2. Select **Connect**. **Connect to virtual machine** opens.

3. Select **Download RDP File**. Azure creates a Remote Desktop Protocol (RDP) file and downloads it to your computer.

4. Open the downloaded RDP file.

   a. If you're prompted, select **Connect**.
   b. Enter the username and password that you specified when you created the VM.

> **NOTE**
> You might need to select **More choices** > **Use a different account** to specify the credentials you entered when you created the VM.

5. Select **OK**.

   You might receive a certificate warning during the sign-in process. If you do, select **Yes** or **Continue**.

6. After the VM desktop appears, minimize it to go back to your local desktop.

Access the SQL Database server privately from the VM

To connect to the SQL Database server from the VM by using the private endpoint, do the following:

1. On the Remote Desktop of *myVM{uniqueid}*, open PowerShell.

2. Run the following command:

   ```
   nslookup sqlserver{uniqueid}.database.windows.net
   ```

   You'll receive a message that's similar to this one:

   ```
   Server:  UnKnown
   Address:  168.63.129.16
   Non-authoritative answer:
   Name:    sqlserver.privatelink.database.windows.net
   Address:  10.0.0.5
   Aliases:  sqlserver.database.windows.net
   ```

3. Install SQL Server Management Studio.

4. Onthe **Connect to server** pane, do the following:

   - For **Server type**, select **Database Engine**.
   - For **Server name**, select **sqlserver{uniqueid}.database.windows.net**.
   - For **Username**, enter the username that was provided earlier.
   - For **Password**, enter the password that was provided earlier.
   - For **Remember password**, select**Yes**.

5. Select **Connect**.

6. On the left pane, select **Databases**. Optionally, you can create or query information from *sample-db*.

7. Close the Remote Desktop connection to *myVm{uniqueid}*.

Clean up resources

When you no longer need the resources that you created with the private link service, delete the resource group. This removes the private link service and all the related resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

For more information about the services that support private endpoints, see:

What is Azure Private Link?

Quickstart: Create a private link service using Bicep

9/21/2022 • 6 minutes to read • Edit Online

In this quickstart, you use Bicep to create a private link service.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

You need an Azure account with an active subscription. Create an account for free.

Review the Bicep file

This Bicep file creates a private link service.

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Username for the Virtual Machine.')
param vmAdminUsername string

@description('Password for the Virtual Machine. The password must be at least 12 characters long and have
lower case, upper characters, digit and a special character (Regex match)')
@secure()
param vmAdminPassword string

@description('The size of the VM')
param vmSize string = 'Standard_D2_v3'

@description('Location for all resources.')
param location string = resourceGroup().location

var vnetName = 'myVirtualNetwork'
var vnetConsumerName = 'myPEVnet'
var vnetAddressPrefix = '10.0.0.0/16'
var frontendSubnetPrefix = '10.0.1.0/24'
var frontendSubnetName = 'frontendSubnet'
var backendSubnetPrefix = '10.0.2.0/24'
var backendSubnetName = 'backendSubnet'
var consumerSubnetPrefix = '10.0.0.0/24'
var consumerSubnetName = 'myPESubnet'
var loadbalancerName = 'myILB'
var backendPoolName = 'myBackEndPool'
var loadBalancerFrontEndIpConfigurationName = 'myFrontEnd'
var healthProbeName = 'myHealthProbe'
var privateEndpointName = 'myPrivateEndpoint'
var vmName = take('myVm${uniqueString(resourceGroup().id)}', 15)
var networkInterfaceName = '${vmName}NetInt'
var vmConsumerName = take('myConsumerVm${uniqueString(resourceGroup().id)}', 15)
var publicIpAddressConsumerName = '${vmConsumerName}PublicIP'
var networkInterfaceConsumerName = '${vmConsumerName}NetInt'
var osDiskType = 'StandardSSD_LRS'
var privatelinkServiceName = 'myPLS'
var loadbalancerId = loadbalancer.id

resource vnet 'Microsoft.Network/virtualNetworks@2021-05-01' = {
  name: vnetName
  location: location
```

```
  properties: {
    addressSpace: {
      addressPrefixes: [
        vnetAddressPrefix
      ]
    }
    subnets: [
      {
        name: frontendSubnetName
        properties: {
          addressPrefix: frontendSubnetPrefix
          privateLinkServiceNetworkPolicies: 'Disabled'
        }
      }
      {
        name: backendSubnetName
        properties: {
          addressPrefix: backendSubnetPrefix
        }
      }
    ]
  }
}

resource loadbalancer 'Microsoft.Network/loadBalancers@2021-05-01' = {
  name: loadbalancerName
  location: location
  sku: {
    name: 'Standard'
  }
  properties: {
    frontendIPConfigurations: [
      {
        name: loadBalancerFrontEndIpConfigurationName
        properties: {
          privateIPAllocationMethod: 'Dynamic'
          subnet: {
            id: resourceId('Microsoft.Network/virtualNetworks/subnets', vnetName, frontendSubnetName)
          }
        }
      }
    ]
    backendAddressPools: [
      {
        name: backendPoolName
      }
    ]
    inboundNatRules: [
      {
        name: 'RDP-VM0'
        properties: {
          frontendIPConfiguration: {
            id: resourceId('Microsoft.Network/loadBalancers/frontendIpConfigurations', loadbalancerName,
loadBalancerFrontEndIpConfigurationName)
          }
          protocol: 'Tcp'
          frontendPort: 3389
          backendPort: 3389
          enableFloatingIP: false
        }
      }
    ]
    loadBalancingRules: [
      {
        name: 'myHTTPRule'
        properties: {
          frontendIPConfiguration: {
            id: resourceId('Microsoft.Network/loadBalancers/frontendIpConfigurations', loadbalancerName,
loadBalancerFrontEndIpConfigurationName)
```

```
          }
          backendAddressPool: {
            id: resourceId('Microsoft.Network/loadBalancers/backendAddressPools', loadbalancerName,
backendPoolName)
          }
          probe: {
            id: resourceId('Microsoft.Network/loadBalancers/probes', loadbalancerName, healthProbeName)
          }
          protocol: 'Tcp'
          frontendPort: 80
          backendPort: 80
          idleTimeoutInMinutes: 15
        }
      }
    ]
    probes: [
      {
        properties: {
          protocol: 'Tcp'
          port: 80
          intervalInSeconds: 15
          numberOfProbes: 2
        }
        name: healthProbeName
      }
    ]
  }
  dependsOn: [
    vnet
  ]
}

resource networkInterface 'Microsoft.Network/networkInterfaces@2021-05-01' = {
  name: networkInterfaceName
  location: location
  tags: {
    displayName: networkInterfaceName
  }
  properties: {
    ipConfigurations: [
      {
        name: 'ipConfig1'
        properties: {
          privateIPAllocationMethod: 'Dynamic'
          subnet: {
            id: resourceId('Microsoft.Network/virtualNetworks/subnets', vnetName, backendSubnetName)
          }
          loadBalancerBackendAddressPools: [
            {
              id: resourceId('Microsoft.Network/loadBalancers/backendAddressPools', loadbalancerName,
backendPoolName)
            }
          ]
          loadBalancerInboundNatRules: [
            {
              id: resourceId('Microsoft.Network/loadBalancers/inboundNatRules/', loadbalancerName, 'RDP-
VM0')
            }
          ]
        }
      }
    ]
  }
  dependsOn: [
    loadbalancer
  ]
}

resource vm 'Microsoft.Compute/virtualMachines@2021-11-01' = {
```

```
  name: vmName
  location: location
  tags: {
    displayName: vmName
  }
  properties: {
    hardwareProfile: {
      vmSize: vmSize
    }
    osProfile: {
      computerName: vmName
      adminUsername: vmAdminUsername
      adminPassword: vmAdminPassword
    }
    storageProfile: {
      imageReference: {
        publisher: 'MicrosoftWindowsServer'
        offer: 'WindowsServer'
        sku: '2019-Datacenter'
        version: 'latest'
      }
      osDisk: {
        name: '${vmName}OsDisk'
        caching: 'ReadWrite'
        createOption: 'FromImage'
        managedDisk: {
          storageAccountType: osDiskType
        }
        diskSizeGB: 128
      }
    }
    networkProfile: {
      networkInterfaces: [
        {
          id: networkInterface.id
        }
      ]
    }
  }
}

resource vmExtension 'Microsoft.Compute/virtualMachines/extensions@2021-11-01' = {
  parent: vm
  name: 'installcustomscript'
  location: location
  tags: {
    displayName: 'install software for Windows VM'
  }
  properties: {
    publisher: 'Microsoft.Compute'
    type: 'CustomScriptExtension'
    typeHandlerVersion: '1.9'
    autoUpgradeMinorVersion: true
    protectedSettings: {
      commandToExecute: 'powershell -ExecutionPolicy Unrestricted Install-WindowsFeature -Name Web-Server'
    }
  }
}

resource privatelinkService 'Microsoft.Network/privateLinkServices@2021-05-01' = {
  name: privatelinkServiceName
  location: location
  properties: {
    enableProxyProtocol: false
    loadBalancerFrontendIpConfigurations: [
      {
        id: resourceId('Microsoft.Network/loadBalancers/frontendIpConfigurations', loadbalancerName,
loadBalancerFrontEndIpConfigurationName)
      }
```

```
    ]
    ipConfigurations: [
      {
        name: 'snet-provider-default-1'
        properties: {
          privateIPAllocationMethod: 'Dynamic'
          privateIPAddressVersion: 'IPv4'
          subnet: {
            id: reference(loadbalancerId, '2019-06-01').frontendIPConfigurations[0].properties.subnet.id
          }
          primary: false
        }
      }
    ]
  }
}

resource vnetConsumer 'Microsoft.Network/virtualNetworks@2021-05-01' = {
  name: vnetConsumerName
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        vnetAddressPrefix
      ]
    }
    subnets: [
      {
        name: consumerSubnetName
        properties: {
          addressPrefix: consumerSubnetPrefix
          privateEndpointNetworkPolicies: 'Disabled'
        }
      }
      {
        name: backendSubnetName
        properties: {
          addressPrefix: backendSubnetPrefix
        }
      }
    ]
  }
}

resource publicIpAddressConsumer 'Microsoft.Network/publicIPAddresses@2021-05-01' = {
  name: publicIpAddressConsumerName
  location: location
  tags: {
    displayName: publicIpAddressConsumerName
  }
  properties: {
    publicIPAllocationMethod: 'Dynamic'
    dnsSettings: {
      domainNameLabel: toLower(vmConsumerName)
    }
  }
}

resource networkInterfaceConsumer 'Microsoft.Network/networkInterfaces@2021-05-01' = {
  name: networkInterfaceConsumerName
  location: location
  tags: {
    displayName: networkInterfaceConsumerName
  }
  properties: {
    ipConfigurations: [
      {
        name: 'ipConfig1'
        properties: {
```

```
          privateIPAllocationMethod: 'Dynamic'
          publicIPAddress: {
            id: publicIpAddressConsumer.id
          }
          subnet: {
            id: resourceId('Microsoft.Network/virtualNetworks/subnets', vnetConsumerName,
consumerSubnetName)
          }
        }
      }
    ]
  }
  dependsOn: [
    vnetConsumer
  ]
}

resource vmConsumer 'Microsoft.Compute/virtualMachines@2021-11-01' = {
  name: vmConsumerName
  location: location
  tags: {
    displayName: vmConsumerName
  }
  properties: {
    hardwareProfile: {
      vmSize: vmSize
    }
    osProfile: {
      computerName: vmConsumerName
      adminUsername: vmAdminUsername
      adminPassword: vmAdminPassword
    }
    storageProfile: {
      imageReference: {
        publisher: 'MicrosoftWindowsServer'
        offer: 'WindowsServer'
        sku: '2019-Datacenter'
        version: 'latest'
      }
      osDisk: {
        name: '${vmConsumerName}OsDisk'
        caching: 'ReadWrite'
        createOption: 'FromImage'
        managedDisk: {
          storageAccountType: osDiskType
        }
        diskSizeGB: 128
      }
    }
    networkProfile: {
      networkInterfaces: [
        {
          id: networkInterfaceConsumer.id
        }
      ]
    }
  }
}

resource privateEndpoint 'Microsoft.Network/privateEndpoints@2021-05-01' = {
  name: privateEndpointName
  location: location
  properties: {
    subnet: {
      id: resourceId('Microsoft.Network/virtualNetworks/subnets', vnetConsumerName, consumerSubnetName)
    }
    privateLinkServiceConnections: [
      {
        name: privateEndpointName
```

```
        name: privateEndpointName
        properties: {
          privateLinkServiceId: privatelinkService.id
        }
      }
    ]
  }
  dependsOn: [
    vnetConsumer
  ]
}
```

Multiple Azure resources are defined in the Bicep file:

- **Microsoft.Network/virtualNetworks**: There's one virtual network for each virtual machine.
- **Microsoft.Network/loadBalancers**: The load balancer that exposes the virtual machines that host the service.
- **Microsoft.Network/networkInterfaces**: There are two network interfaces, one for each virtual machine.
- **Microsoft.Compute/virtualMachines**: There are two virtual machines, one that hosts the service and one that tests the connection to the private endpoint.
- **Microsoft.Compute/virtualMachines/extensions**: The extension that installs a web server.
- **Microsoft.Network/privateLinkServices**: The private link service to expose the service.
- **Microsoft.Network/publicIpAddresses**: There is a public IP address for the test virtual machine.
- **Microsoft.Network/privateendpoints**: The private endpoint to access the service.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

   ```
   az group create --name exampleRG --location eastus
   az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
   vmAdminUsername=<admin-user>
   ```

   > **NOTE**
   > Replace **<admin-user>** with the username for the virtual machine. You'll also be prompted to enter **vmAdminPassword**. The password must be at least 12 characters long and have uppercase and lowercase characters, a digit, and a special character.

   When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Validate the deployment

> **NOTE**
> The Bicep file generates a unique name for the virtual machine myConsumerVm**{uniqueid}** resource. Substitute your generated value for **{uniqueid}**.

Connect to a VM from the internet

Connect to the VM *myConsumerVm{uniqueid}* from the internet as follows:

1. In the Azure portal search bar, enter *myConsumerVm{uniqueid}*.
2. Select **Connect**. **Connect to virtual machine** opens.
3. Select **Download RDP File**. Azure creates a Remote Desktop Protocol (*.rdp*) file and downloads it to your computer.
4. Open the downloaded .rdp file.

   a. If prompted, select **Connect**.

   b. Enter the username and password you specified when you created the VM.

   > **NOTE**
   > You might need to select **More choices** > **Use a different account**, to specify the credentials you entered when you created the VM.

5. Select **OK**.
6. You might receive a certificate warning during the sign-in process. If you receive a certificate warning, select **Yes** or **Continue**.
7. After the VM desktop appears, minimize it to go back to your local desktop.

Access the http service privately from the VM

Here's how to connect to the http service from the VM by using the private endpoint.

1. Go to the Remote Desktop of *myConsumerVm{uniqueid}*.
2. Open a browser, and enter the private endpoint address: `http://10.0.0.5/`.
3. The default IIS page appears.

Clean up resources

When you no longer need the resources that you created with the private link service, delete the resource group. This removes the private link service and all the related resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

For more information on the services that support a private endpoint, see:

Private Link availability

Quickstart: Create a Traffic Manager profile using Bicep

9/21/2022 • 3 minutes to read • Edit Online

This quickstart describes how to use Bicep to create a Traffic Manager profile with external endpoints using the performance routing method.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Relative DNS name for the traffic manager profile, must be globally unique.')
param uniqueDnsName string

resource ExternalEndpointExample 'Microsoft.Network/trafficmanagerprofiles@2018-08-01' = {
  name: 'ExternalEndpointExample'
  location: 'global'
  properties: {
    profileStatus: 'Enabled'
    trafficRoutingMethod: 'Performance'
    dnsConfig: {
      relativeName: uniqueDnsName
      ttl: 30
    }
    monitorConfig: {
      protocol: 'HTTPS'
      port: 443
      path: '/'
      expectedStatusCodeRanges: [
        {
          min: 200
          max: 202
        }
        {
          min: 301
          max: 302
        }
      ]
    }
    endpoints: [
      {
        type: 'Microsoft.Network/TrafficManagerProfiles/ExternalEndpoints'
        name: 'endpoint1'
        properties: {
          target: 'www.microsoft.com'
          endpointStatus: 'Enabled'
          endpointLocation: 'northeurope'
        }
      }
      {
        type: 'Microsoft.Network/TrafficManagerProfiles/ExternalEndpoints'
        name: 'endpoint2'
        properties: {
          target: 'docs.microsoft.com'
          endpointStatus: 'Enabled'
          endpointLocation: 'southcentralus'
        }
      }
    ]
  }
}
```

One Azure resource is defined in the Bicep file:

- **Microsoft.Network/trafficManagerProfiles**

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.
   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
uniqueDnsName=<dns-name>
```

The Bicep file deployment creates a profile with two external endpoints. **Endpoint1** uses a target endpoint of `www.microsoft.com` with the location in **North Europe**. **Endpoint2** uses a target endpoint of `learn.microsoft.com` with the location in **South Central US**.

> **NOTE**
>
> **uniqueDNSname** needs to be a globally unique name in order for the Bicep file to deploy successfully.

When the deployment finishes, you'll see a message indicating the deployment succeeded.

Validate the deployment

Use Azure CLI or Azure PowerShell to validate the deployment.

1. Determine the DNS name of the Traffic Manager profile.

   - CLI
   - PowerShell

   ```
   az network traffic-manager profile show --name ExternalEndpointExample --resource-group exampleRG
   ```

   From the output, copy the **fqdn** value. It'll be in the following format: `<relativeDnsName>.trafficmanager.net`. This value is also the DNS name of your Traffic Manager profile.

2. Run the following command by replacing the **{relativeDnsName}** variable with `<relativeDnsName>.trafficmanager.net`.

   - CLI
   - PowerShell

   ```
   nslookup -type=cname {relativeDnsName}
   ```

   You should get a canonical name of either `www.microsoft.com` or `learn.microsoft.com` depending on which region is closer to you.

3. To check if you can resolve to the other endpoint, disable the endpoint for the target you got in the last step. Replace the **{endpointName}** with either **endpoint1** or **endpoint2** to disable the target for `www.microsoft.com` or `learn.microsoft.com` respectively.

   - CLI
   - PowerShell

   ```
   az network traffic-manager endpoint update --name {endpointName} --type externalEndpoints --profile-
   name ExternalEndpointExample --resource-group exampleRG --endpoint-status "Disabled"
   ```

4. Run the command from Step 2 again in Azure CLI or Azure PowerShell. This time, you should get the other canonical name/NameHost for the other endpoint.

Clean up resources

When you no longer need the Traffic Manager profile, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group. This removes the Traffic Manager profile and all the related resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you created a Traffic Manager profile using Bicep.

To learn more about routing traffic, continue to the Traffic Manager tutorials.

Traffic Manager tutorials

Quickstart: Create a virtual network - Bicep

9/21/2022 • 2 minutes to read • Edit Online

In this quickstart, you learn how to create a virtual network with two subnets using Bicep. A virtual network is the fundamental building block for your private network in Azure. It enables Azure resources, like VMs, to securely communicate with each other and with the internet.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates

```
@description('VNet name')
param vnetName string = 'VNet1'

@description('Address prefix')
param vnetAddressPrefix string = '10.0.0.0/16'

@description('Subnet 1 Prefix')
param subnet1Prefix string = '10.0.0.0/24'

@description('Subnet 1 Name')
param subnet1Name string = 'Subnet1'

@description('Subnet 2 Prefix')
param subnet2Prefix string = '10.0.1.0/24'

@description('Subnet 2 Name')
param subnet2Name string = 'Subnet2'

@description('Location for all resources.')
param location string = resourceGroup().location

resource vnet 'Microsoft.Network/virtualNetworks@2021-08-01' = {
  name: vnetName
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        vnetAddressPrefix
      ]
    }
    subnets: [
      {
        name: subnet1Name
        properties: {
          addressPrefix: subnet1Prefix
        }
      }
      {
        name: subnet2Name
        properties: {
          addressPrefix: subnet2Prefix
        }
      }
    ]
  }
}
```

The following Azure resources have been defined in the Bicep file:

- **Microsoft.Network/virtualNetworks**: Create an Azure virtual network.
- **Microsoft.Network/virtualNetworks/subnets**: Create a subnet.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.

2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep
```

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use Azure CLI or Azure PowerShell to review the deployed resources.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

You can use the Azure portal to explore the resources by browsing the settings blades for **VNet1**.

1. On the **Overview** tab, you will see the defined address space of **10.0.0.0/16**.
2. On the **Subnets** tab, you will see the deployed subnets of **Subnet1** and **Subnet2** with the appropriate values from the Bicep file.

Clean up resources

When you no longer need the resources that you created with the virtual network, use Azure portal, Azure CLI, or Azure PowerShell to delete the resource group. This removes the virtual network and all the related resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you deployed an Azure virtual network with two subnets. To learn more about Azure virtual networks, continue to the tutorial for virtual networks.

Filter network traffic

Quickstart: Create an Azure WAF v2 on Application Gateway using Bicep

9/21/2022 • 6 minutes to read • Edit Online

In this quickstart, you use Bicep to create an Azure Web Application Firewall v2 on Application Gateway.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

> **NOTE**
>
> To interact with Azure, the Azure Az PowerShell module is recommended. See Install Azure PowerShell to get started. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

Prerequisites

- An Azure account with an active subscription. Create an account for free.

Review the Bicep file

This Bicep file creates a simple Web Application Firewall v2 on Azure Application Gateway. This includes a public IP frontend IP address, HTTP settings, a rule with a basic listener on port 80, and a backend pool. The file also creates a WAF policy with a custom rule to block traffic to the backend pool based on an IP address match type.

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Admin username for the backend servers')
param adminUsername string

@description('Password for the admin account on the backend servers')
@secure()
param adminPassword string

@description('Location for all resources.')
param location string = resourceGroup().location

@description('Size of the virtual machine.')
param vmSize string = 'Standard_B2ms'

var virtualMachines_myVM_name = 'myVM'
var virtualNetworks_myVNet_name = 'myVNet'
var myNic_name = 'net-int'
var ipconfig_name = 'ipconfig'
var publicIPAddress_name = 'public_ip'
var nsg_name = 'vm-nsg'
var applicationGateways_myAppGateway_name = 'myAppGateway'
var vnet_prefix = '10.0.0.0/16'
var ag_subnet_prefix = '10.0.0.0/24'
var backend_subnet_prefix = '10.0.1.0/24'
var AppGW_AppFW_Pol_name = 'WafPol01'

resource nsg 'Microsoft.Network/networkSecurityGroups@2021-08-01' = [for i in range(0, 2): {
  name: '${nsg_name}${(i + 1)}'
  location: location
```

```
  properties: {
    securityRules: [
      {
        name: 'RDP'
        properties: {
          protocol: 'Tcp'
          sourcePortRange: '*'
          destinationPortRange: '3389'
          sourceAddressPrefix: '*'
          destinationAddressPrefix: '*'
          access: 'Allow'
          priority: 300
          direction: 'Inbound'
        }
      }
    ]
  }
}]

resource publicIPAddress 'Microsoft.Network/publicIPAddresses@2021-08-01' = [for i in range(0, 3): {
  name: '${publicIPAddress_name}${i}'
  location: location
  sku: {
    name: 'Standard'
  }
  properties: {
    publicIPAddressVersion: 'IPv4'
    publicIPAllocationMethod: 'Static'
    idleTimeoutInMinutes: 4
  }
}]

resource myVNet 'Microsoft.Network/virtualNetworks@2021-08-01' = {
  name: virtualNetworks_myVNet_name
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        vnet_prefix
      ]
    }
    subnets: [
      {
        name: 'myAGSubnet'
        properties: {
          addressPrefix: ag_subnet_prefix
          privateEndpointNetworkPolicies: 'Enabled'
          privateLinkServiceNetworkPolicies: 'Enabled'
        }
      }
      {
        name: 'myBackendSubnet'
        properties: {
          addressPrefix: backend_subnet_prefix
          privateEndpointNetworkPolicies: 'Enabled'
          privateLinkServiceNetworkPolicies: 'Enabled'
        }
      }
    ]
    enableDdosProtection: false
    enableVmProtection: false
  }
}

resource myVM 'Microsoft.Compute/virtualMachines@2021-11-01' = [for i in range(0, 2): {
  name: '${virtualMachines_myVM_name}${(i + 1)}'
  location: location
  properties: {
    hardwareProfile: {
```

```
      vmSize: vmSize
    }
    storageProfile: {
      imageReference: {
        publisher: 'MicrosoftWindowsServer'
        offer: 'WindowsServer'
        sku: '2019-Datacenter'
        version: 'latest'
      }
      osDisk: {
        osType: 'Windows'
        createOption: 'FromImage'
        caching: 'ReadWrite'
        managedDisk: {
          storageAccountType: 'StandardSSD_LRS'
        }
        diskSizeGB: 127
      }
    }
    osProfile: {
      computerName: '${virtualMachines_myVM_name}${(i + 1)}'
      adminUsername: adminUsername
      adminPassword: adminPassword
      windowsConfiguration: {
        provisionVMAgent: true
        enableAutomaticUpdates: true
      }
      allowExtensionOperations: true
    }
    networkProfile: {
      networkInterfaces: [
        {
          id: resourceId('Microsoft.Network/networkInterfaces', '${myNic_name}${(i + 1)}')
        }
      ]
    }
  }
  dependsOn: [
    myNic
  ]
}]

resource myVM_IIS 'Microsoft.Compute/virtualMachines/extensions@2021-11-01' = [for i in range(0, 2): {
  name: '${virtualMachines_myVM_name}${(i + 1)}/IIS'
  location: location
  properties: {
    autoUpgradeMinorVersion: true
    publisher: 'Microsoft.Compute'
    type: 'CustomScriptExtension'
    typeHandlerVersion: '1.4'
    settings: {
      commandToExecute: 'powershell Add-WindowsFeature Web-Server; powershell Add-Content -Path
"C:\\inetpub\\wwwroot\\Default.htm" -Value $($env:computername)'
    }
  }
  dependsOn: [
    myVM
  ]
}]

resource myAppGateway 'Microsoft.Network/applicationGateways@2021-08-01' = {
  name: applicationGateways_myAppGateway_name
  location: location
  properties: {
    sku: {
      name: 'WAF_v2'
      tier: 'WAF_v2'
      capacity: 2
    }
```

```
    gatewayIPConfigurations: [
      {
        name: 'appGatewayIpConfig'
        properties: {
          subnet: {
            id: resourceId('Microsoft.Network/virtualNetworks/subnets', virtualNetworks_myVNet_name,
'myAGSubnet')
          }
        }
      }
    ]
    frontendIPConfigurations: [
      {
        name: 'appGwPublicFrontendIp'
        properties: {
          privateIPAllocationMethod: 'Dynamic'
          publicIPAddress: {
            id: resourceId('Microsoft.Network/publicIPAddresses', '${publicIPAddress_name}0')
          }
        }
      }
    ]
    frontendPorts: [
      {
        name: 'port_80'
        properties: {
          port: 80
        }
      }
    ]
    backendAddressPools: [
      {
        name: 'myBackendPool'
        properties: {}
      }
    ]
    backendHttpSettingsCollection: [
      {
        name: 'myHTTPSetting'
        properties: {
          port: 80
          protocol: 'Http'
          cookieBasedAffinity: 'Disabled'
          pickHostNameFromBackendAddress: false
          requestTimeout: 20
        }
      }
    ]
    httpListeners: [
      {
        name: 'myListener'
        properties: {
          firewallPolicy: {
            id: AppGW_AppFW_Pol.id
          }
          frontendIPConfiguration: {
            id: resourceId('Microsoft.Network/applicationGateways/frontendIPConfigurations',
applicationGateways_myAppGateway_name, 'appGwPublicFrontendIp')
          }
          frontendPort: {
            id: resourceId('Microsoft.Network/applicationGateways/frontendPorts',
applicationGateways_myAppGateway_name, 'port_80')
          }
          protocol: 'Http'
          requireServerNameIndication: false
        }
      }
    ]
    requestRoutingRules: [
```

```
      {
        name: 'myRoutingRule'
        properties: {
          ruleType: 'Basic'
          priority: 10
          httpListener: {
            id: resourceId('Microsoft.Network/applicationGateways/httpListeners',
applicationGateways_myAppGateway_name, 'myListener')
          }
          backendAddressPool: {
            id: resourceId('Microsoft.Network/applicationGateways/backendAddressPools',
applicationGateways_myAppGateway_name, 'myBackendPool')
          }
          backendHttpSettings: {
            id: resourceId('Microsoft.Network/applicationGateways/backendHttpSettingsCollection',
applicationGateways_myAppGateway_name, 'myHTTPSetting')
          }
        }
      }
    ]
    enableHttp2: false
    webApplicationFirewallConfiguration: {
      enabled: true
      firewallMode: 'Prevention'
      ruleSetType: 'OWASP'
      ruleSetVersion: '3.1'
      requestBodyCheck: true
      maxRequestBodySizeInKb: 128
      fileUploadLimitInMb: 100
    }
    firewallPolicy: {
      id: AppGW_AppFW_Pol.id
    }
  }
  dependsOn: [
    myVNet
    publicIPAddress
  ]
}

resource AppGW_AppFW_Pol 'Microsoft.Network/ApplicationGatewayWebApplicationFirewallPolicies@2021-08-01' = {
  name: AppGW_AppFW_Pol_name
  location: location
  properties: {
    customRules: [
      {
        name: 'CustRule01'
        priority: 100
        ruleType: 'MatchRule'
        action: 'Block'
        matchConditions: [
          {
            matchVariables: [
              {
                variableName: 'RemoteAddr'
              }
            ]
            operator: 'IPMatch'
            negationConditon: true
            matchValues: [
              '10.10.10.0/24'
            ]
          }
        ]
      }
    ]
    policySettings: {
      requestBodyCheck: true
      maxRequestBodySizeInKb: 128
```

```
      fileUploadLimitInMb: 100
      state: 'Enabled'
      mode: 'Prevention'
    }
    managedRules: {
      managedRuleSets: [
        {
          ruleSetType: 'OWASP'
          ruleSetVersion: '3.1'
        }
      ]
    }
  }
}

resource myNic 'Microsoft.Network/networkInterfaces@2021-08-01' = [for i in range(0, 2): {
  name: '${myNic_name}${(i + 1)}'
  location: location
  properties: {
    ipConfigurations: [
      {
        name: '${ipconfig_name}${(i + 1)}'
        properties: {
          privateIPAllocationMethod: 'Dynamic'
          publicIPAddress: {
            id: resourceId('Microsoft.Network/publicIPAddresses', '${publicIPAddress_name}${(i + 1)}')
          }
          subnet: {
            id: resourceId('Microsoft.Network/virtualNetworks/subnets', virtualNetworks_myVNet_name,
'myBackendSubnet')
          }
          primary: true
          privateIPAddressVersion: 'IPv4'
          applicationGatewayBackendAddressPools: [
            {
              id: resourceId('Microsoft.Network/applicationGateways/backendAddressPools',
applicationGateways_myAppGateway_name, 'myBackendPool')
            }
          ]
        }
      }
    ]
    enableAcceleratedNetworking: false
    enableIPForwarding: false
    networkSecurityGroup: {
      id: resourceId('Microsoft.Network/networkSecurityGroups', '${nsg_name}${(i + 1)}')
    }
  }
  dependsOn: [
    publicIPAddress
    myVNet
    myAppGateway
    nsg
  ]
}]
```

Multiple Azure resources are defined in the Bicep file:

- Microsoft.Network/applicationgateways
- Microsoft.Network/ApplicationGatewayWebApplicationFirewallPolicies
- Microsoft.Network/publicIPAddresses : one for the application gateway, and two for the virtual machines.
- Microsoft.Network/networkSecurityGroups
- Microsoft.Network/virtualNetworks

- `Microsoft.Compute/virtualMachines` : two virtual machines
- `Microsoft.Network/networkInterfaces` : two for the virtual machines
- `Microsoft.Compute/virtualMachine/extensions` : to configure IIS and the web pages

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.

2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
adminUsername=<admin-user>
```

> **NOTE**
>
> You'll be prompted to enter **adminPassword**, which is the password for the admin account on the backend servers. The password must be between 8-123 characters long and must contain at least three of the following: an uppercase character, a lowercase character, a numeric digit, or a special character.

When the deployment finishes, you should see a message indicating the deployment succeeded. The deployment can take 10 minutes or longer to complete.

Validate the deployment

Although IIS isn't required to create the application gateway, it's installed on the backend servers to verify if Azure successfully created a WAF v2 on the application gateway.

Use IIS to test the application gateway:

1. Find the public IP address for the application gateway on its **Overview** page.

2. Copy the public IP address, and then paste it into the address bar of your browser to browse that IP address.

3. Check the response. A **403 Forbidden** response verifies that the WAF was successfully created and is blocking connections to the backend pool.

4. Change the custom rule to **Allow traffic** using Azure PowerShell.

```
$rgName = "exampleRG"
$appGWName = "myAppGateway"
$fwPolicyName = "WafPol01"

# Pull the existing Azure resources

$appGW = Get-AzApplicationGateway -Name $appGWName -ResourceGroupName $rgName
$pol = Get-AzApplicationGatewayFirewallPolicy -Name $fwPolicyName -ResourceGroupName $rgName

# Update the resources

$pol[0].CustomRules[0].Action = "allow"
$appGW.FirewallPolicy = $pol

# Push your changes to Azure

Set-AzApplicationGatewayFirewallPolicy -Name $fwPolicyName -ResourceGroupName $rgName -CustomRule
$pol.CustomRules
Set-AzApplicationGateway -ApplicationGateway $appGW
```

Refresh your browser multiple times and you should see connections to both myVM1 and myVM2.

Clean up resources

When you no longer need the resources that you created with the application gateway, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group. This removes the application gateway and all the related resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

Tutorial: Create an application gateway with a Web Application Firewall using the Azure portal

Quickstart: Create an Azure Attestation provider with a Bicep file

9/21/2022 • 2 minutes to read • Edit Online

Microsoft Azure Attestation is a solution for attesting Trusted Execution Environments (TEEs). This quickstart focuses on the process of deploying a Bicep file to create a Microsoft Azure Attestation policy.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Name of the Attestation provider. Must be between 3 and 24 characters in length and use
numbers and lower-case letters only.')
param attestationProviderName string = uniqueString(resourceGroup().name)

@description('Location for all resources.')
param location string = resourceGroup().location

param policySigningCertificates string = ''

var PolicySigningCertificates = {
  PolicySigningCertificates: {
    keys: [
      {
        kty: 'RSA'
        use: 'sig'
        x5c: [
          policySigningCertificates
        ]
      }
    ]
  }
}

resource attestationProvider 'Microsoft.Attestation/attestationProviders@2021-06-01-preview' = {
  name: attestationProviderName
  location: location
  properties: (empty(policySigningCertificates) ? json('{}') : PolicySigningCertificates)
}

output attestationName string = attestationProviderName
```

Azure resources defined in the Bicep file:

- Microsoft.Attestation/attestationProviders

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

   ```
   az group create --name exampleRG --location eastus

   az deployment group create --resource-group exampleRG --template-file main.bicep
   ```

   When the deployment finishes, you should see a message indicating the deployment succeeded.

Validate the deployment

Use the Azure portal, Azure CLI, or Azure PowerShell to verify the resource group and server resource were created.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

Other Azure Attestation build upon this quickstart. If you plan to continue on to work with subsequent quickstarts and tutorials, you may wish to leave these resources in place.

When no longer needed, delete the resource group, which deletes the Attestation resource. To delete the resource group by using Azure CLI or Azure PowerShell:

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you created an attestation resource using a Bicep file, and validated the deployment. To learn more about Azure Attestation, see Overview of Azure Attestation.

Quickstart: Create an automatic response to a specific security alert using an ARM template or Bicep

9/21/2022 • 6 minutes to read • Edit Online

This quickstart describes how to use an Azure Resource Manager template (ARM template) or a Bicep file to create a workflow automation that triggers a logic app when specific security alerts are received by Microsoft Defender for Cloud.

Prerequisites

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

For a list of the roles and permissions required to work with Microsoft Defender for Cloud's workflow automation feature, see workflow automation.

The examples in this quickstart assume you have an existing Logic App. To deploy the example, you pass in parameters that contain the logic app name and resource group. For information about deploying a logic app, see Quickstart: Create and deploy a Consumption logic app workflow in multi-tenant Azure Logic Apps with Bicep or Quickstart: Create and deploy a Consumption logic app workflow in multi-tenant Azure Logic Apps with an ARM template.

ARM template tutorial

An ARM template is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project. The template uses declarative syntax. In declarative syntax, you describe your intended deployment without writing the sequence of programming commands to create the deployment.

If your environment meets the prerequisites and you're familiar with using ARM templates, select the **Deploy to Azure** button. The template will open in the Azure portal.

Deploy to Azure

Review the template

The template used in this quickstart is from Azure Quickstart Templates.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "metadata": {
    "_generator": {
      "name": "bicep",
      "version": "0.5.6.12127",
      "templateHash": "5191074894407113732"
    }
  },
  "parameters": {
    "automationName": {
      "type": "string",
      "maxLength": 24,
      "minLength": 3
    },
    "location": {
```

```
      "location": {
        "type": "string",
        "defaultValue": "[resourceGroup().location]",
        "metadata": {
          "description": "Location for the automation"
        }
      },
      "logicAppName": {
        "type": "string",
        "minLength": 3
      },
      "logicAppResourceGroupName": {
        "type": "string",
        "minLength": 3
      },
      "subscriptionId": {
        "type": "string",
        "defaultValue": "[subscription().subscriptionId]",
        "metadata": {
          "description": "The Azure resource GUID id of the subscription"
        }
      },
      "alertSettings": {
        "type": "object",
        "metadata": {
          "description": "The alert settings object used for deploying the automation"
        }
      }
    },
    "variables": {
      "automationDescription": "automation description for subscription {0}",
      "scopeDescription": "automation scope for subscription {0}"
    },
    "resources": [
      {
        "type": "Microsoft.Security/automations",
        "apiVersion": "2019-01-01-preview",
        "name": "[parameters('automationName')]",
        "location": "[parameters('location')]",
        "properties": {
          "description": "[format(variables('automationDescription'), parameters('subscriptionId'))]",
          "isEnabled": true,
          "actions": [
            {
              "actionType": "LogicApp",
              "logicAppResourceId": "[resourceId('Microsoft.Logic/workflows', parameters('logicAppName'))]",
              "uri": "[listCallbackURL(resourceId(parameters('subscriptionId'),
parameters('logicAppResourceGroupName'), 'Microsoft.Logic/workflows/triggers', parameters('logicAppName'),
'manual'), '2019-05-01').value]"
            }
          ],
          "scopes": [
            {
              "description": "[format(variables('scopeDescription'), parameters('subscriptionId'))]",
              "scopePath": "[subscription().id]"
            }
          ],
          "sources": [
            {
              "copy": [
                {
                  "name": "ruleSets",
                  "count": "[length(range(0, length(parameters('alertSettings').alertSeverityMapping)))]",
                  "input": {
                    "rules": [
                      {
                        "propertyJPath": "[parameters('alertSettings').alertSeverityMapping[range(0,
length(parameters('alertSettings').alertSeverityMapping))[copyIndex('ruleSets')]].jpath]",
                        "propertyType": "String",
                        "expectedValue": "[parameters('alertSettings').alertSeverityMapping[range(0
```

```
                      "expectedValue": "[parameters('alertSettings').alertSeverityMapping[range(0,
length(parameters('alertSettings').alertSeverityMapping))[copyIndex('ruleSets')]].expectedValue]",
                      "operator": "[parameters('alertSettings').alertSeverityMapping[range(0,
length(parameters('alertSettings').alertSeverityMapping))[copyIndex('ruleSets')]].operator]"
                    },
                    {
                      "propertyJPath": "Severity",
                      "propertyType": "String",
                      "expectedValue": "[parameters('alertSettings').alertSeverityMapping[range(0,
length(parameters('alertSettings').alertSeverityMapping))[copyIndex('ruleSets')]].severity]",
                      "operator": "Equals"
                    }
                  ]
                }
              }
            ],
            "eventSource": "Alerts"
          }
        ]
      }
    }
  ]
}
```

Relevant resources

- Microsoft.Security/automations: The automation that will trigger the logic app, upon receiving a Microsoft Defender for Cloud alert that contains a specific string.
- Microsoft.Logic/workflows: An empty triggerable Logic App.

For other Defender for Cloud quickstart templates, see these community contributed templates.

Deploy the template

- PowerShell:

```
New-AzResourceGroup -Name <resource-group-name> -Location <resource-group-location> #use this command
when you need to create a new resource group for your deployment
New-AzResourceGroupDeployment -ResourceGroupName <resource-group-name> -TemplateUri
https://raw.githubusercontent.com/Azure/azure-quickstart-
templates/master/quickstarts/microsoft.security/securitycenter-create-automation-for-
alertnamecontains/azuredeploy.json
```

- CLI:

```
az group create --name <resource-group-name> --location <resource-group-location> #use this command
when you need to create a new resource group for your deployment
az deployment group create --resource-group <my-resource-group> --template-uri
https://raw.githubusercontent.com/Azure/azure-quickstart-
templates/master/quickstarts/microsoft.security/securitycenter-create-automation-for-
alertnamecontains/azuredeploy.json
```

- Portal:

  Deploy to Azure

  To find more information about this deployment option, see Use a deployment button to deploy templates from GitHub repository.

Review deployed resources

Use the Azure portal to check the workflow automation has been deployed.

1. From the Azure portal, open **Microsoft Defender for Cloud**.
2. From the top menu bar, select the filter icon, and select the specific subscription on which you deployed the new workflow automation.
3. From Microsoft Defender for Cloud's menu, open **workflow automation** and check for your new automation.

> **TIP**
> If you have many workflow automations on your subscription, use the **filter by name** option.

Clean up resources

When no longer needed, delete the workflow automation using the Azure portal.

1. From the Azure portal, open **Microsoft Defender for Cloud**.
2. From the top menu bar, select the filter icon, and select the specific subscription on which you deployed the new workflow automation.
3. From Microsoft Defender for Cloud's menu, open **workflow automation** and find the automation to be deleted.

4. Select the checkbox for the item to be deleted.
5. From the toolbar, select **Delete**.

Bicep tutorial

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@minLength(3)
@maxLength(24)
param automationName string

@description('Location for the automation')
param location string = resourceGroup().location

@minLength(3)
param logicAppName string

@minLength(3)
param logicAppResourceGroupName string

@description('The Azure resource GUID id of the subscription')
param subscriptionId string = subscription().subscriptionId

@description('The alert settings object used for deploying the automation')
param alertSettings object

var automationDescription = 'automation description for subscription {0}'
var scopeDescription = 'automation scope for subscription {0}'

resource automation 'Microsoft.Security/automations@2019-01-01-preview' = {
  name: automationName
  location: location
  properties: {
    description: format(automationDescription, subscriptionId)
    isEnabled: true
    actions: [
      {
        actionType: 'LogicApp'
        logicAppResourceId: resourceId('Microsoft.Logic/workflows', logicAppName)
        uri: listCallbackURL(resourceId(subscriptionId, logicAppResourceGroupName,
'Microsoft.Logic/workflows/triggers', logicAppName, 'manual'), '2019-05-01').value
      }
    ]
    scopes: [
      {
        description: format(scopeDescription, subscriptionId)
        scopePath: subscription().id
      }
    ]
    sources: [
      {
        eventSource: 'Alerts'
        ruleSets: [for j in range(0, length(alertSettings.alertSeverityMapping)): {
          rules: [
            {
              propertyJPath: alertSettings.alertSeverityMapping[j].jpath
              propertyType: 'String'
              expectedValue: alertSettings.alertSeverityMapping[j].expectedValue
              operator: alertSettings.alertSeverityMapping[j].operator
            }
            {
              propertyJPath: 'Severity'
              propertyType: 'String'
              expectedValue: alertSettings.alertSeverityMapping[j].severity
              operator: 'Equals'
            }
          ]
        }]
      }
    ]
  }
}
```

Relevant resources

- **Microsoft.Security/automations**: The automation that will trigger the logic app, upon receiving a Microsoft Defender for Cloud alert that contains a specific string.
- **Microsoft.Logic/workflows**: An empty triggerable Logic App.

For other Defender for Cloud quickstart templates, see these community contributed templates.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
automationName=<automation-name> logicAppName=<logic-name> logicAppResourceGroupName=<group-name>
alertSettings={alert-settings}
```

   You're required to enter the following parameters:

   - **automationName**: Replace **<automation-name>** with the name of the automation. It has a minimum length of 3 characters and a maximum length of 24 characters.
   - **logicAppName**: Replace **<logic-name>** with the name of the logic app. It has a minimum length of 3 characters.
   - **logicAppResourceGroupName**: Replace **<group-name>** with the name of the resource group in which the resources are located. It has a minimum length of 3 characters.
   - **alertSettings**: Replace **{alert-settings}** with the alert settings object used for deploying the automation.

   > **NOTE**
   > When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and all of its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

For step-by-step tutorials that guide you through the process of creating an ARM template or a Bicep file, see:

Tutorial: Create and deploy your first ARM template Quickstart: Create Bicep files with Visual Studio Code

Quickstart: Create an Azure key vault and a key by using Bicep

9/21/2022 • 4 minutes to read • Edit Online

Azure Key Vault is a cloud service that provides a secure store for secrets, such as keys, passwords, and certificate. This quickstart focuses on the process of deploying a Bicep file to create a key vault and a key.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

To complete this article:

- If you don't have an Azure subscription, create a free account before you begin.
- User would need to have an Azure built-in role assigned, recommended role **contributor**. Learn more here

Review the Bicep file

```
@description('The name of the key vault to be created.')
param vaultName string

@description('The name of the key to be created.')
param keyName string

@description('The location of the resources')
param location string = resourceGroup().location

@description('The SKU of the vault to be created.')
@allowed([
  'standard'
  'premium'
])
param skuName string = 'standard'

@description('The JsonWebKeyType of the key to be created.')
@allowed([
  'EC'
  'EC-HSM'
  'RSA'
  'RSA-HSM'
])
param keyType string = 'RSA'

@description('The permitted JSON web key operations of the key to be created.')
param keyOps array = []

@description('The size in bits of the key to be created.')
param keySize int = 2048

@description('The JsonWebKeyCurveName of the key to be created.')
@allowed([
  ''
  'P-256'
  'P-256K'
  'P-384'
```

```
  'P-521'
])
param curveName string = ''

resource vault 'Microsoft.KeyVault/vaults@2021-11-01-preview' = {
  name: vaultName
  location: location
  properties: {
    accessPolicies:[]
    enableRbacAuthorization: false
    enableSoftDelete: false
    enabledForDeployment: false
    enabledForDiskEncryption: false
    enabledForTemplateDeployment: false
    tenantId: subscription().tenantId
    sku: {
      name: skuName
      family: 'A'
    }
    networkAcls: {
      defaultAction: 'Allow'
      bypass: 'AzureServices'
    }
  }
}

resource key 'Microsoft.KeyVault/vaults/keys@2021-11-01-preview' = {
  parent: vault
  name: keyName
  properties: {
    kty: keyType
    keyOps: keyOps
    keySize: keySize
    curveName: curveName
  }
}

output proxyKey object = key.properties
```

Two resources are defined in the Bicep file:

- Microsoft.KeyVault/vaults
- Microsoft.KeyVault/vaults/keys

More Azure Key Vault template samples can be found in Azure Quickstart Templates.

Parameters and definitions

PARAMETER	DEFINITION
keyOps	Specifies operations that can be performed by using the key. If you don't specify this parameter, all operations can be performed. The acceptable values for this parameter are a comma-separated list of key operations as defined by the JSON Web Key (JWK) specification: `["sign", "verify", "encrypt", "decrypt", " wrapKey", "unwrapKey"]`
CurveName	Elliptic curve (EC) name for EC key type. See JsonWebKeyCurveName
Kty	The type of key to create. For valid values, see JsonWebKeyType

PARAMETER	DEFINITION
Tags	Application-specific metadata in the form of key-value pairs.
nbf	Specifies the time, as a DateTime object, before which the key can't be used. The format would be Unix time stamp (the number of seconds after Unix Epoch on January 1st, 1970 at UTC).
exp	Specifies the expiration time, as a DateTime object. The format would be Unix time stamp (the number of seconds after Unix Epoch on January 1st, 1970 at UTC).

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
vaultName=<vault-name> keyName=<key-name>
```

> **NOTE**
> Replace **<vault-name>** with the name of the key vault. Replace **<vault-name>** with the name of the key vault, and replace **<key-name>** with the name of the key.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

You can use the Azure portal to check the key vault and the key. Alternatively, use the following Azure CLI or Azure PowerShell script to list the key created.

- CLI
- PowerShell

```
echo "Enter your key vault name:" &&
read keyVaultName &&
az keyvault key list --vault-name $keyVaultName &&
echo "Press [ENTER] to continue ..."
```

Creating key using ARM template is different from creating key via data plane

Creating a key via ARM

- It's only possible to create *new* keys. It isn't possible to update existing keys, nor create new versions of

existing keys. If the key already exists, then the existing key is retrieved from storage and used (no write operations will occur).

- To be authorized to use this API, the caller needs to have the **"Microsoft.KeyVault/vaults/keys/write"** role-based access control (RBAC) Action. The built-in "Key Vault Contributor" role is sufficient, since it authorizes all RBAC Actions that match the pattern "Microsoft.KeyVault/*".

Existing API (creating key via data plane)

- It's possible to create new keys, update existing keys, and create new versions of existing keys.
- The caller must be authorized to use this API. If the vault uses access policies, the caller must have "create" key permission; if the vault is enabled for RBAC, the caller must have "Microsoft.KeyVault/vaults/keys/create/action" RBAC DataAction.

Clean up resources

Other Key Vault quickstarts and tutorials build upon this quickstart. If you plan to continue on to work with subsequent quickstarts and tutorials, you may wish to leave these resources in place. When no longer needed, delete the resource group, which deletes the Key Vault and related resources. To delete the resource group by using Azure CLI or Azure PowerShell:

- CLI
- PowerShell

```
echo "Enter the Resource Group name:" &&
read resourceGroupName &&
az group delete --name $resourceGroupName &&
echo "Press [ENTER] to continue ..."
```

Next steps

In this quickstart, you created a key vault and a key using a Bicep file, and validated the deployment. To learn more about Key Vault and Azure Resource Manager, see these articles.

- Read an Overview of Azure Key Vault
- Learn more about Azure Resource Manager
- Review the Key Vault security overview

Quickstart: Set and retrieve a secret from Azure Key Vault using Bicep

9/21/2022 • 4 minutes to read • Edit Online

Azure Key Vault is a cloud service that provides a secure store for secrets, such as keys, passwords, certificates, and other secrets. This quickstart focuses on the process of deploying a Bicep file to create a key vault and a secret.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin.
- Your Azure AD user object ID is needed by the template to configure permissions. The following procedure gets the object ID (GUID).
  1. Run the following Azure PowerShell or Azure CLI command by select **Try it**, and then paste the script into the shell pane. To paste the script, right-click the shell, and then select **Paste**.
     - CLI
     - PowerShell

     ```
     echo "Enter your email address that is used to sign in to Azure:" &&
     read upn &&
     az ad user show --id $upn --query "objectId" &&
     echo "Press [ENTER] to continue ..."
     ```

  2. Write down the object ID. You need it in the next section of this quickstart.

Review the Bicep file

The template used in this quickstart is from Azure Quickstart Templates.

```
@description('Specifies the name of the key vault.')
param keyVaultName string

@description('Specifies the Azure location where the key vault should be created.')
param location string = resourceGroup().location

@description('Specifies whether Azure Virtual Machines are permitted to retrieve certificates stored as
secrets from the key vault.')
param enabledForDeployment bool = false

@description('Specifies whether Azure Disk Encryption is permitted to retrieve secrets from the vault and
unwrap keys.')
param enabledForDiskEncryption bool = false

@description('Specifies whether Azure Resource Manager is permitted to retrieve secrets from the key
vault.')
param enabledForTemplateDeployment bool = false
```

```
@description('Specifies the Azure Active Directory tenant ID that should be used for authenticating requests
to the key vault. Get it by using Get-AzSubscription cmdlet.')
param tenantId string = subscription().tenantId

@description('Specifies the object ID of a user, service principal or security group in the Azure Active
Directory tenant for the vault. The object ID must be unique for the list of access policies. Get it by
using Get-AzADUser or Get-AzADServicePrincipal cmdlets.')
param objectId string

@description('Specifies the permissions to keys in the vault. Valid values are: all, encrypt, decrypt,
wrapKey, unwrapKey, sign, verify, get, list, create, update, import, delete, backup, restore, recover, and
purge.')
param keysPermissions array = [
  'list'
]

@description('Specifies the permissions to secrets in the vault. Valid values are: all, get, list, set,
delete, backup, restore, recover, and purge.')
param secretsPermissions array = [
  'list'
]

@description('Specifies whether the key vault is a standard vault or a premium vault.')
@allowed([
  'standard'
  'premium'
])
param skuName string = 'standard'

@description('Specifies the name of the secret that you want to create.')
param secretName string

@description('Specifies the value of the secret that you want to create.')
@secure()
param secretValue string

resource kv 'Microsoft.KeyVault/vaults@2021-11-01-preview' = {
  name: keyVaultName
  location: location
  properties: {
    enabledForDeployment: enabledForDeployment
    enabledForDiskEncryption: enabledForDiskEncryption
    enabledForTemplateDeployment: enabledForTemplateDeployment
    tenantId: tenantId
    accessPolicies: [
      {
        objectId: objectId
        tenantId: tenantId
        permissions: {
          keys: keysPermissions
          secrets: secretsPermissions
        }
      }
    ]
    sku: {
      name: skuName
      family: 'A'
    }
    networkAcls: {
      defaultAction: 'Allow'
      bypass: 'AzureServices'
    }
  }
}

resource secret 'Microsoft.KeyVault/vaults/secrets@2021-11-01-preview' = {
  parent: kv
  name: secretName
```

```
  properties: {
    value: secretValue
  }
}
```

Two Azure resources are defined in the Bicep file:

- **Microsoft.KeyVault/vaults**: create an Azure key vault.
- **Microsoft.KeyVault/vaults/secrets**: create a key vault secret.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

   ```
   az group create --name exampleRG --location eastus
   az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
   keyVaultName=<vault-name> objectID=<object-id>
   ```

   > **NOTE**
   > Replace **<vault-name>** with the name of the key vault. Replace **<object-id>** with the object ID of a user, service principal, or security group in the Azure Active Directory tenant for the vault. The object ID must be unique for the list of access policies. Get it by using Get-AzADUser or Get-AzADServicePrincipal cmdlets.

   When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

You can either use the Azure portal to check the key vault and the secret, or use the following Azure CLI or Azure PowerShell script to list the secret created.

- CLI
- PowerShell

```
echo "Enter your key vault name:" &&
read keyVaultName &&
az keyvault secret list --vault-name $keyVaultName &&
echo "Press [ENTER] to continue ..."
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you created a key vault and a secret using Bicep and then validated the deployment. To learn more about Key Vault and Bicep, continue on to the articles below.

- Read an Overview of Azure Key Vault
- Learn more about Bicep
- Review the Key Vault security overview

Quickstart: Share data using Azure Data Share and Bicep

9/21/2022 • 2 minutes to read • Edit Online

Learn how to set up a new Azure Data Share from an Azure storage account using Bicep, and start sharing your data with customers and partners outside of your Azure organization. For a list of the supported data stores, see Supported data stores in Azure Data Share.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Specify a project name that is used to generate resource names.')
param projectName string

@description('Specify the location for the resources.')
param location string = resourceGroup().location

@description('Specify an email address for receiving data share invitations.')
param invitationEmail string

@description('Specify snapshot schedule recurrence.')
@allowed([
  'Day'
  'Hour'
])
param syncInterval string = 'Day'

@description('Specify snapshot schedule start time.')
param syncTime string = utcNow('yyyy-MM-ddTHH:mm:ssZ')

var storageAccountName = '${projectName}store'
var containerName = '${projectName}container'
var dataShareAccountName = '${projectName}shareaccount'
var dataShareName = '${projectName}share'
var roleAssignmentName = guid(sa.id, storageBlobDataReaderRoleDefinitionId, dataShareAccount.id)
var inviteName = '${dataShareName}invite'
var storageBlobDataReaderRoleDefinitionId = resourceId('Microsoft.Authorization/roleDefinitions', '2a2b9908-
6ea1-4ae2-8e65-a410df84e7d1')

resource sa 'Microsoft.Storage/storageAccounts@2021-04-01' = {
  name: storageAccountName
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
  properties: {
    accessTier: 'Hot'
```

```
  }
}

resource container 'Microsoft.Storage/storageAccounts/blobServices/containers@2021-04-01' = {
  name: '${sa.name}/default/${containerName}'
}

resource dataShareAccount 'Microsoft.DataShare/accounts@2021-08-01' = {
  name: dataShareAccountName
  location: location
  identity: {
    type: 'SystemAssigned'
  }
  properties: {}
}

resource dataShare 'Microsoft.DataShare/accounts/shares@2021-08-01' = {
  parent: dataShareAccount
  name: dataShareName
  properties: {
    shareKind: 'CopyBased'
  }
}

resource roleAssignment 'Microsoft.Authorization/roleAssignments@2020-04-01-preview' = {
  scope: sa
  name: roleAssignmentName
  properties: {
    roleDefinitionId: storageBlobDataReaderRoleDefinitionId
    principalId: dataShareAccount.identity.principalId
    principalType: 'ServicePrincipal'
  }
}

resource dataSet 'Microsoft.DataShare/accounts/shares/dataSets@2021-08-01' = {
  parent: dataShare
  name: containerName
  kind: 'Container'
  dependsOn: [ // this is used to delay this resource until the roleAssignment replicates
    container
    invitation
    synchronizationSetting
  ]
  properties: {
    subscriptionId: subscription().subscriptionId
    resourceGroup: resourceGroup().name
    storageAccountName: sa.name
    containerName: containerName
  }
}

resource invitation 'Microsoft.DataShare/accounts/shares/invitations@2021-08-01' = {
  parent: dataShare
  name: inviteName
  properties: {
    targetEmail: invitationEmail
  }
}

resource synchronizationSetting 'Microsoft.DataShare/accounts/shares/synchronizationSettings@2021-08-01' = {
  parent: dataShare
  name: '${dataShareName}_synchronizationSetting'
  kind: 'ScheduleBased'
  properties: {
    recurrenceInterval: syncInterval
    synchronizationTime: syncTime
  }
}
```

The following resources are defined in the Bicep file:

- Microsoft.Storage/storageAccounts:
- Microsoft.Storage/storageAccounts/blobServices/containers
- Microsoft.DataShare/accounts
- Microsoft.DataShare/accounts/shares
- Microsoft.Storage/storageAccounts/providers/roleAssignments
- Microsoft.DataShare/accounts/shares/dataSets
- Microsoft.DataShare/accounts/shares/invitations
- Microsoft.DataShare/accounts/shares/synchronizationSettings

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
projectName=<project-name> invitationEmail=<invitation-email>
```

> **NOTE**
>
> Replace **<project-name>** with a project name. The project name will be used to generate resource names.
> Replace **<invitation-email>** with an email address for receiving data share invitations.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you learned how to create an Azure data share and invite recipients. To learn more about how a data consumer can accept and receive a data share, continue to the accept and receive data tutorial.

Create a storage account

9/21/2022 • 22 minutes to read • Edit Online

An Azure storage account contains all of your Azure Storage data objects: blobs, files, queues, and tables. The storage account provides a unique namespace for your Azure Storage data that is accessible from anywhere in the world over HTTP or HTTPS. For more information about Azure storage accounts, see Storage account overview.

In this how-to article, you learn to create a storage account using the Azure portal, Azure PowerShell, Azure CLI, or an Azure Resource Manager template.

Prerequisites

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

- Portal
- PowerShell
- Azure CLI
- Bicep
- Template

None.

Next, sign in to Azure.

- Portal
- PowerShell
- Azure CLI
- Bicep
- Template

Sign in to the Azure portal.

Create a storage account

A storage account is an Azure Resource Manager resource. Resource Manager is the deployment and management service for Azure. For more information, see Azure Resource Manager overview.

Every Resource Manager resource, including an Azure storage account, must belong to an Azure resource group. A resource group is a logical container for grouping your Azure services. When you create a storage account, you have the option to either create a new resource group, or use an existing resource group. This how-to shows how to create a new resource group.

- Portal
- PowerShell
- Azure CLI
- Bicep
- Template

To create an Azure storage account with the Azure portal, follow these steps:

1. From the left portal menu, select **Storage accounts** to display a list of your storage accounts. If the portal menu isn't visible, click the menu button to toggle it on.

2. On the **Storage accounts** page, select **Create**.

Options for your new storage account are organized into tabs in the **Create a storage account** page. The following sections describe each of the tabs and their options.

Basics tab

On the **Basics** tab, provide the essential information for your storage account. After you complete the **Basics** tab, you can choose to further customize your new storage account by setting options on the other tabs, or you can select **Review + create** to accept the default options and proceed to validate and create the account.

The following table describes the fields on the **Basics** tab.

SECTION	FIELD	REQUIRED OR OPTIONAL	DESCRIPTION
Project details	Subscription	Required	Select the subscription for the new storage account.

SECTION	FIELD	REQUIRED OR OPTIONAL	DESCRIPTION
Project details	Resource group	Required	Create a new resource group for this storage account, or select an existing one. For more information, see Resource groups.
Instance details	Storage account name	Required	Choose a unique name for your storage account. Storage account names must be between 3 and 24 characters in length and may contain numbers and lowercase letters only.
Instance details	Region	Required	Select the appropriate region for your storage account. For more information, see Regions and Availability Zones in Azure. Not all regions are supported for all types of storage accounts or redundancy configurations. For more information, see Azure Storage redundancy. The choice of region can have a billing impact. For more information, see Storage account billing.

SECTION	FIELD	REQUIRED OR OPTIONAL	DESCRIPTION
Instance details	Performance	Required	Select **Standard** performance for general-purpose v2 storage accounts (default). This type of account is recommended by Microsoft for most scenarios. For more information, see Types of storage accounts. Select **Premium** for scenarios requiring low latency. After selecting **Premium**, select the type of premium storage account to create. The following types of premium storage accounts are available: • Block blobs • File shares • Page blobs Microsoft recommends creating a general-purpose v2, premium block blob, or premium file share account for most scenarios. To select a legacy account type, use the link provided beneath **Instance details**. For more information about legacy account types, see Legacy storage account types.
Instance details	Redundancy	Required	Select your desired redundancy configuration. Not all redundancy options are available for all types of storage accounts in all regions. For more information about redundancy configurations, see Azure Storage redundancy. If you select a geo-redundant configuration (GRS or GZRS), your data is replicated to a data center in a different region. For read access to data in the secondary region, select **Make read access to data available in the event of regional unavailability**.

The following image shows a standard configuration of the basic properties for a new storage account.

Advanced tab

On the **Advanced** tab, you can configure additional options and modify default settings for your new storage account. Some of these options can also be configured after the storage account is created, while others must be configured at the time of creation.

The following table describes the fields on the **Advanced** tab.

SECTION	FIELD	REQUIRED OR OPTIONAL	DESCRIPTION
Security	Require secure transfer for REST API operations	Optional	Require secure transfer to ensure that incoming requests to this storage account are made only via HTTPS (default). Recommended for optimal security. For more information, see Require secure transfer to ensure secure connections.

SECTION	FIELD	REQUIRED OR OPTIONAL	DESCRIPTION
Security	Enable blob public access	Optional	When enabled, this setting allows a user with the appropriate permissions to enable anonymous public access to a container in the storage account (default). Disabling this setting prevents all anonymous public access to the storage account. For more information, see Prevent anonymous public read access to containers and blobs. Enabling blob public access does not make blob data available for public access unless the user takes the additional step to explicitly configure the container's public access setting.
Security	Enable storage account key access	Optional	When enabled, this setting allows clients to authorize requests to the storage account using either the account access keys or an Azure Active Directory (Azure AD) account (default). Disabling this setting prevents authorization with the account access keys. For more information, see Prevent Shared Key authorization for an Azure Storage account.
Security	Default to Azure Active Directory authorization in the Azure portal	Optional	When enabled, the Azure portal authorizes data operations with the user's Azure AD credentials by default. If the user does not have the appropriate permissions assigned via Azure role-based access control (Azure RBAC) to perform data operations, then the portal will use the account access keys for data access instead. The user can also choose to switch to using the account access keys. For more information, see Default to Azure AD authorization in the Azure portal.

SECTION	FIELD	REQUIRED OR OPTIONAL	DESCRIPTION
Security	Minimum TLS version	Required	Select the minimum version of Transport Layer Security (TLS) for incoming requests to the storage account. The default value is TLS version 1.2. When set to the default value, incoming requests made using TLS 1.0 or TLS 1.1 are rejected. For more information, see Enforce a minimum required version of Transport Layer Security (TLS) for requests to a storage account.
Data Lake Storage Gen2	Enable hierarchical namespace	Optional	To use this storage account for Azure Data Lake Storage Gen2 workloads, configure a hierarchical namespace. For more information, see Introduction to Azure Data Lake Storage Gen2.
Blob storage	Enable SFTP	Optional	Enable the use of Secure File Transfer Protocol (SFTP) to securely transfer of data over the internet. For more information, see Secure File Transfer (SFTP) protocol support in Azure Blob Storage.
Blob storage	Enable network file share (NFS) v3	Optional	NFS v3 provides Linux file system compatibility at object storage scale enables Linux clients to mount a container in Blob storage from an Azure Virtual Machine (VM) or a computer on-premises. For more information, see Network File System (NFS) 3.0 protocol support in Azure Blob Storage.
Blob storage	Allow cross-tenant replication	Required	By default, users with appropriate permissions can configure object replication across Azure AD tenants. To prevent replication across tenants, deselect this option. For more information, see Prevent replication across Azure AD tenants.

SECTION	FIELD	REQUIRED OR OPTIONAL	DESCRIPTION
Blob storage	Access tier	Required	Blob access tiers enable you to store blob data in the most cost-effective manner, based on usage. Select the hot tier (default) for frequently accessed data. Select the cool tier for infrequently accessed data. For more information, see Hot, Cool, and Archive access tiers for blob data.
Azure Files	Enable large file shares	Optional	Available only for standard file shares with the LRS or ZRS redundancies.

The following image shows a standard configuration of the advanced properties for a new storage account.

Networking tab

On the **Networking** tab, you can configure network connectivity and routing preference settings for your new

storage account. These options can also be configured after the storage account is created.

The following table describes the fields on the **Networking** tab.

SECTION	FIELD	REQUIRED OR OPTIONAL	DESCRIPTION
Network connectivity	Connectivity method	Required	By default, incoming network traffic is routed to the public endpoint for your storage account. You can specify that traffic must be routed to the public endpoint through an Azure virtual network. You can also configure private endpoints for your storage account. For more information, see Use private endpoints for Azure Storage.
Network connectivity	Endpoint type	Required	Azure Storage supports two types of endpoints: standard endpoints (the default) and Azure DNS zone endpoints (preview). Within a given subscription, you can create up to 250 accounts with standard endpoints per region, and up to 5000 accounts with Azure DNS zone endpoints per region. To learn how to view the service endpoints for an existing storage account, see Get service endpoints for the storage account.
Network routing	Routing preference	Required	The network routing preference specifies how network traffic is routed to the public endpoint of your storage account from clients over the internet. By default, a new storage account uses Microsoft network routing. You can also choose to route network traffic through the POP closest to the storage account, which may lower networking costs. For more information, see Network routing preference for Azure Storage.

The following image shows a standard configuration of the networking properties for a new storage account.

> **IMPORTANT**
>
> Azure DNS zone endpoints are currently in PREVIEW. See the Supplemental Terms of Use for Microsoft Azure Previews for legal terms that apply to Azure features that are in beta, preview, or otherwise not yet released into general availability.

Data protection tab

On the **Data protection** tab, you can configure data protection options for blob data in your new storage account. These options can also be configured after the storage account is created. For an overview of data protection options in Azure Storage, see Data protection overview.

The following table describes the fields on the **Data protection** tab.

SECTION	FIELD	REQUIRED OR OPTIONAL	DESCRIPTION
Recovery	Enable point-in-time restore for containers	Optional	Point-in-time restore provides protection against accidental deletion or corruption by enabling you to restore block blob data to an earlier state. For more information, see Point-in-time restore for block blobs. Enabling point-in-time restore also enables blob versioning, blob soft delete, and blob change feed. These prerequisite features may have a cost impact. For more information, see Pricing and billing for point-in-time restore.

SECTION	FIELD	REQUIRED OR OPTIONAL	DESCRIPTION
Recovery	Enable soft delete for blobs	Optional	Blob soft delete protects an individual blob, snapshot, or version from accidental deletes or overwrites by maintaining the deleted data in the system for a specified retention period. During the retention period, you can restore a soft-deleted object to its state at the time it was deleted. For more information, see Soft delete for blobs. Microsoft recommends enabling blob soft delete for your storage accounts and setting a minimum retention period of seven days.
Recovery	Enable soft delete for containers	Optional	Container soft delete protects a container and its contents from accidental deletes by maintaining the deleted data in the system for a specified retention period. During the retention period, you can restore a soft-deleted container to its state at the time it was deleted. For more information, see Soft delete for containers (preview). Microsoft recommends enabling container soft delete for your storage accounts and setting a minimum retention period of seven days.

SECTION	FIELD	REQUIRED OR OPTIONAL	DESCRIPTION
Recovery	Enable soft delete for file shares	Optional	Soft delete for file shares protects a file share and its contents from accidental deletes by maintaining the deleted data in the system for a specified retention period. During the retention period, you can restore a soft-deleted file share to its state at the time it was deleted. For more information, see Prevent accidental deletion of Azure file shares. Microsoft recommends enabling soft delete for file shares for Azure Files workloads and setting a minimum retention period of seven days.
Tracking	Enable versioning for blobs	Optional	Blob versioning automatically saves the state of a blob in a previous version when the blob is overwritten. For more information, see Blob versioning. Microsoft recommends enabling blob versioning for optimal data protection for the storage account.
Tracking	Enable blob change feed	Optional	The blob change feed provides transaction logs of all changes to all blobs in your storage account, as well as to their metadata. For more information, see Change feed support in Azure Blob Storage.

SECTION	FIELD	REQUIRED OR OPTIONAL	DESCRIPTION
Access control	Enable version-level immutability support	Optional	Enable support for immutability policies that are scoped to the blob version. If this option is selected, then after you create the storage account, you can configure a default time-based retention policy for the account or for the container, which blob versions within the account or container will inherit by default. For more information, see Enable version-level immutability support on a storage account.

The following image shows a standard configuration of the data protection properties for a new storage account.

Encryption tab

On the **Encryption** tab, you can configure options that relate to how your data is encrypted when it is persisted to the cloud. Some of these options can be configured only when you create the storage account.

FIELD	REQUIRED OR OPTIONAL	DESCRIPTION
Encryption type	Required	By default, data in the storage account is encrypted by using Microsoft-managed keys. You can rely on Microsoft-managed keys for the encryption of your data, or you can manage encryption with your own keys. For more information, see Azure Storage encryption for data at rest.
Enable support for customer-managed keys	Required	By default, customer managed keys can be used to encrypt only blobs and files. Set this option to **All service types (blobs, files, tables, and queues)** to enable support for customer-managed keys for all services. You are not required to use customer-managed keys if you choose this option. For more information, see Customer-managed keys for Azure Storage encryption.
Encryption key	Required if **Encryption type** field is set to **Customer-managed keys**.	If you choose **Select a key vault and key**, you are presented with the option to navigate to the key vault and key that you wish to use. If you choose **Enter key from URI**, then you are presented with a field to enter the key URI and the subscription.
User-assigned identity	Required if **Encryption type** field is set to **Customer-managed keys**.	If you are configuring customer-managed keys at create time for the storage account, you must provide a user-assigned identity to use for authorizing access to the key vault.
Enable infrastructure encryption	Optional	By default, infrastructure encryption is not enabled. Enable infrastructure encryption to encrypt your data at both the service level and the infrastructure level. For more information, see Create a storage account with infrastructure encryption enabled for double encryption of data.

The following image shows a standard configuration of the encryption properties for a new storage account.

Tags tab

On the **Tags** tab, you can specify Resource Manager tags to help organize your Azure resources. For more information, see Tag resources, resource groups, and subscriptions for logical organization.

The following image shows a standard configuration of the index tag properties for a new storage account.

Review + create tab

When you navigate to the **Review + create** tab, Azure runs validation on the storage account settings that you have chosen. If validation passes, you can proceed to create the storage account.

If validation fails, then the portal indicates which settings need to be modified.

The following image shows the **Review** tab data prior to the creation of a new storage account.

Delete a storage account

Deleting a storage account deletes the entire account, including all data in the account. Be sure to back up any data you want to save before you delete the account.

Under certain circumstances, a deleted storage account may be recovered, but recovery is not guaranteed. For more information, see Recover a deleted storage account.

If you try to delete a storage account associated with an Azure virtual machine, you may get an error about the storage account still being in use. For help troubleshooting this error, see Troubleshoot errors when you delete storage accounts.

- Portal
- PowerShell
- Azure CLI
- Bicep
- Template

1. Navigate to the storage account in the Azure portal.
2. Click **Delete**.

Alternately, you can delete the resource group, which deletes the storage account and any other resources in that resource group. For more information about deleting a resource group, see Delete resource group and resources.

Next steps

- Storage account overview
- Upgrade to a general-purpose v2 storage account
- Move a storage account to another region
- Recover a deleted storage account

Quickstart: Create a new Azure API Management service instance using Bicep

9/21/2022 • 2 minutes to read • Edit Online

This quickstart describes how to use a Bicep file to create an Azure API Management (APIM) service instance. APIM helps organizations publish APIs to external, partner, and internal developers to unlock the potential of their data and services. API Management provides the core competencies to ensure a successful API program through developer engagement, business insights, analytics, security, and protection. APIM enables you to create and manage modern API gateways for existing backend services hosted anywhere. For more information, see the Overview.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('The name of the API Management service instance')
param apiManagementServiceName string = 'apiservice${uniqueString(resourceGroup().id)}'

@description('The email address of the owner of the service')
@minLength(1)
param publisherEmail string

@description('The name of the owner of the service')
@minLength(1)
param publisherName string

@description('The pricing tier of this API Management service')
@allowed([
  'Developer'
  'Standard'
  'Premium'
])
param sku string = 'Developer'

@description('The instance size of this API Management service.')
@allowed([
  1
  2
])
param skuCount int = 1

@description('Location for all resources.')
param location string = resourceGroup().location

resource apiManagementService 'Microsoft.ApiManagement/service@2021-08-01' = {
  name: apiManagementServiceName
  location: location
  sku: {
    name: sku
    capacity: skuCount
  }
  properties: {
    publisherEmail: publisherEmail
    publisherName: publisherName
  }
}
```

The following resource is defined in the Bicep file:

- **Microsoft.ApiManagement/service**

In this example, the Bicep file configures the API Management instance in the Developer tier, an economical option to evaluate Azure API Management. This tier isn't for production use.

More Azure API Management Bicep samples can be found in Azure Quickstart Templates.

Deploy the Bicep file

You can use Azure CLI or Azure PowerShell to deploy the Bicep file. For more information about deploying Bicep files, see Deploy.

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.
   - CLI
   - PowerShell

```
az group create --name exampleRG --location eastus

az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
publisherEmail=<publisher-email> publisherName=<publisher-name>
```

Replace **<publisher-name>** and **<publisher-email>** with the name of the API publisher's organization and the email address to receive notifications.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI or Azure PowerShell to list the deployed App Configuration resource in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

When your API Management service instance is online, you're ready to use it. Start with the tutorial to import and publish your first API.

Clean up resources

If you plan to continue working with subsequent tutorials, you might want to leave the API Management instance in place. When no longer needed, delete the resource group, which deletes the resources in the resource group.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

Tutorial: Import and publish your first API

Quickstart: Create a notification hub using Bicep

9/21/2022 • 2 minutes to read • Edit Online

Azure Notification Hubs provides an easy-to-use and scaled-out push engine that enables you to send notifications to any platform (iOS, Android, Windows, Kindle, etc.) from any backend (cloud or on-premises). For more information about the service, see What is Azure Notification Hubs.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

This quickstart uses Bicep to create an Azure Notification Hubs namespace, and a notification hub named **MyHub** within that namespace.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('The name of the Notification Hubs namespace.')
param namespaceName string

@description('The location in which the Notification Hubs resources should be deployed.')
param location string = resourceGroup().location

var hubName = 'MyHub'

resource namespace 'Microsoft.NotificationHubs/namespaces@2017-04-01' = {
  name: namespaceName
  location: location
  sku: {
    name: 'Free'
  }
}

resource notificationHub 'Microsoft.NotificationHubs/namespaces/notificationHubs@2017-04-01' = {
  name: hubName
  location: location
  parent: namespace
  properties: {
  }
}
```

The Bicep file creates the two Azure resources:

- Microsoft.NotificationHubs/namespaces
- Microsoft.NotificationHubs/namespaces/notificationHubs

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

- CLI
- PowerShell

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep --parameters
namespaceName=<namespace-name>
```

> **NOTE**
>
> Replace **<namespace-name>** with the name of the Notifications Hub namespace.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When you no longer need the logic app, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

For a step-by-step tutorial that guides you through the process of creating a Bicep file, see:

Quickstart: Create Bicep files with Visual Studio Code

Create a Web App plus Azure Cache for Redis using Bicep

9/21/2022 • 2 minutes to read • Edit Online

In this article, you use Bicep to deploy an Azure Web App that uses Azure Cache for Redis, as well as an App Service plan.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

You can use this Bicep file for your own deployments. The Bicep file provides unique names for the Azure Web App, the App Service plan, and the Azure Cache for Redis. If you'd like, you can customize the Bicep file after you save it to your local device to meet your requirements.

For more information about creating Bicep files, see Quickstart: Create Bicep files with Visual Studio Code. To learn about Bicep syntax, see Understand the structure and syntax of Bicep files.

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('Describes plan\'s pricing tier and instance size. Check details at
https://azure.microsoft.com/en-us/pricing/details/app-service/')
@allowed([
  'F1'
  'D1'
  'B1'
  'B2'
  'B3'
  'S1'
  'S2'
  'S3'
  'P1'
  'P2'
  'P3'
  'P4'
])
param skuName string = 'F1'

@description('Describes plan\'s instance count')
@minValue(1)
@maxValue(7)
param skuCapacity int = 1

@description('The pricing tier of the new Azure Redis Cache.')
@allowed([
  'Basic'
  'Standard'
])
param cacheSKUName string = 'Basic'

@description('The family for the sku.')
@allowed([
  'C'
])
param cacheSKUFamily string = 'C'
```

```
@description('The size of the new Azure Redis Cache instance. ')
@minValue(0)
@maxValue(6)
param cacheSKUCapacity int = 0

@description('Location for all resources.')
param location string = resourceGroup().location

var hostingPlanName = 'hostingplan${uniqueString(resourceGroup().id)}'
var webSiteName = 'webSite${uniqueString(resourceGroup().id)}'
var cacheName = 'cache${uniqueString(resourceGroup().id)}'

resource hostingPlan 'Microsoft.Web/serverfarms@2021-03-01' = {
  name: hostingPlanName
  location: location
  tags: {
    displayName: 'HostingPlan'
  }
  sku: {
    name: skuName
    capacity: skuCapacity
  }
  properties: {
  }
}

resource webSite 'Microsoft.Web/sites@2021-03-01' = {
  name: webSiteName
  location: location
  tags: {
    'hidden-related:${hostingPlan.id}': 'empty'
    displayName: 'Website'
  }
  identity: {
    type: 'SystemAssigned'
  }
  properties: {
    serverFarmId: hostingPlan.id
    httpsOnly: true
  }
  dependsOn: [
    cache
  ]
}

resource appsettings 'Microsoft.Web/sites/config@2021-03-01' = {
  parent: webSite
  name: 'appsettings'
  properties: {
    CacheConnection:
'${cacheName}.redis.cache.windows.net,abortConnect=false,ssl=true,password=${cache.listKeys().primaryKey}'
    minTlsVersion: '1.2'
    ftpsState: 'FtpsOnly'
  }
}

resource cache 'Microsoft.Cache/Redis@2021-06-01' = {
  name: cacheName
  location: location
  tags: {
    displayName: 'cache'
  }
  properties: {
    sku: {
      name: cacheSKUName
      family: cacheSKUFamily
      capacity: cacheSKUCapacity
    }
  }
```

```
}
```

With this Bicep file, you deploy:

- **Microsoft.Cache/Redis**
- **Microsoft.Web/sites**
- **Microsoft.Web/serverfarms**

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

   ```
   az group create --name exampleRG --location eastus
   az deployment group create --resource-group exampleRG --template-file main.bicep
   ```

   When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

To learn more about Bicep, continue to the following article:

- Bicep overview

Quickstart: Use Bicep to deploy Azure SignalR Service

9/21/2022 • 2 minutes to read • Edit Online

This quickstart describes how to use Bicep to create an Azure SignalR Service using Azure CLI or PowerShell.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

An Azure account with an active subscription. Create one for free.

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

```
@description('The globally unique name of the SignalR resource to create.')
param name string = uniqueString(resourceGroup().id)

@description('Location for the SignalR resource.')
param location string = resourceGroup().location

@description('The pricing tier of the SignalR resource.')
@allowed([
  'Free_F1'
  'Standard_S1'
  'Premium_P1'
])
param pricingTier string = 'Standard_S1'

@description('The number of SignalR Unit.')
@allowed([
  1
  2
  5
  10
  20
  50
  100
])
param capacity int = 1

@description('Visit https://github.com/Azure/azure-signalr/blob/dev/docs/faq.md#service-mode to understand
SignalR Service Mode.')
@allowed([
  'Default'
  'Serverless'
  'Classic'
])
param serviceMode string = 'Default'

param enableConnectivityLogs bool = true

param enableMessagingLogs bool = true

param enableLiveTrace bool = true
```

```
@description('Set the list of origins that should be allowed to make cross-origin calls.')
param allowedOrigins array = [
  '*'
]

resource signalR 'Microsoft.SignalRService/signalR@2022-02-01' = {
  name: name
  location: location
  sku: {
    capacity: capacity
    name: pricingTier
  }
  kind: 'SignalR'
  identity: {
    type: 'SystemAssigned'
  }
  properties: {
    tls: {
      clientCertEnabled: false
    }
    features: [
      {
        flag: 'ServiceMode'
        value: serviceMode
      }
      {
        flag: 'EnableConnectivityLogs'
        value: string(enableConnectivityLogs)
      }
      {
        flag: 'EnableMessagingLogs'
        value: string(enableMessagingLogs)
      }
      {
        flag: 'EnableLiveTrace'
        value: string(enableLiveTrace)
      }
    ]
    cors: {
      allowedOrigins: allowedOrigins
    }
    networkACLs: {
      defaultAction: 'Deny'
      publicNetwork: {
        allow: [
          'ClientConnection'
        ]
      }
      privateEndpoints: [
        {
          name: 'mySignalRService.1fa229cd-bf3f-47f0-8c49-afb36723997e'
          allow: [
            'ServerConnection'
          ]
        }
      ]
    }
    upstream: {
      templates: [
        {
          categoryPattern: '*'
          eventPattern: 'connect,disconnect'
          hubPattern: '*'
          urlTemplate: 'https://example.com/chat/api/connect'
        }
      ]
    }
  }
```

```
}
```

The Bicep file defines one Azure resource:

- **Microsoft.SignalRService/SignalR**

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   - CLI
   - PowerShell

   ```
   az group create --name exampleRG --location eastus
   az deployment group create --resource-group exampleRG --template-file main.bicep
   ```

   When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

```
az resource list --resource-group exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

- CLI
- PowerShell

```
az group delete --name exampleRG
```

Next steps

For a step-by-step tutorial that guides you through the process of creating a Bicep file using Visual Studio Code, see:

Quickstart: Create Bicep files with Visual Studio Code

Understand the structure and syntax of Bicep files

9/21/2022 • 7 minutes to read • Edit Online

This article describes the structure and syntax of a Bicep file. It presents the different sections of the file and the properties that are available in those sections.

For a step-by-step tutorial that guides you through the process of creating a Bicep file, see Quickstart: Create Bicep files with Visual Studio Code.

Bicep format

Bicep is a declarative language, which means the elements can appear in any order. Unlike imperative languages, the order of elements doesn't affect how deployment is processed.

A Bicep file has the following elements.

```
targetScope = '<scope>'

@<decorator>(<argument>)
param <parameter-name> <parameter-data-type> = <default-value>

var <variable-name> = <variable-value>

resource <resource-symbolic-name> '<resource-type>@<api-version>' = {
  <resource-properties>
}

module <module-symbolic-name> '<path-to-file>' = {
  name: '<linked-deployment-name>'
  params: {
    <parameter-names-and-values>
  }
}

output <output-name> <output-data-type> = <output-value>
```

The following example shows an implementation of these elements.

```
@minLength(3)
@maxLength(11)
param storagePrefix string

param storageSKU string = 'Standard_LRS'
param location string = resourceGroup().location

var uniqueStorageName = '${storagePrefix}${uniqueString(resourceGroup().id)}'

resource stg 'Microsoft.Storage/storageAccounts@2019-04-01' = {
  name: uniqueStorageName
  location: location
  sku: {
    name: storageSKU
  }
  kind: 'StorageV2'
  properties: {
    supportsHttpsTrafficOnly: true
  }
}

module webModule './webApp.bicep' = {
  name: 'webDeploy'
  params: {
    skuName: 'S1'
    location: location
  }
}

output storageEndpoint object = stg.properties.primaryEndpoints
```

Target scope

By default, the target scope is set to `resourceGroup`. If you're deploying at the resource group level, you don't need to set the target scope in your Bicep file.

The allowed values are:

- **resourceGroup** - default value, used for resource group deployments.
- **subscription** - used for subscription deployments.
- **managementGroup** - used for management group deployments.
- **tenant** - used for tenant deployments.

In a module, you can specify a scope that is different than the scope for the rest of the Bicep file. For more information, see Configure module scope

Parameters

Use parameters for values that need to vary for different deployments. You can define a default value for the parameter that is used if no value is provided during deployment.

For example, you can add a SKU parameter to specify different sizes for a resource. You might pass in different values depending on whether you're deploying to test or production.

```
param storageSKU string = 'Standard_LRS'
```

The parameter is available for use in your Bicep file.

```
sku: {
  name: storageSKU
}
```

For more information, see Parameters in Bicep.

Parameter decorators

You can add one or more decorators for each parameter. These decorators describe the parameter and define constraints for the values that are passed in. The following example shows one decorator but there are many others that are available.

```
@allowed([
  'Standard_LRS'
  'Standard_GRS'
  'Standard_ZRS'
  'Premium_LRS'
])
param storageSKU string = 'Standard_LRS'
```

For more information, including descriptions of all available decorators, see Decorators.

Variables

You can make your Bicep file more readable by encapsulating complex expressions in a variable. For example, you might add a variable for a resource name that is constructed by concatenating several values together.

```
var uniqueStorageName = '${storagePrefix}${uniqueString(resourceGroup().id)}'
```

Apply this variable wherever you need the complex expression.

```
resource stg 'Microsoft.Storage/storageAccounts@2019-04-01' = {
  name: uniqueStorageName
```

For more information, see Variables in Bicep.

Resources

Use the `resource` keyword to define a resource to deploy. Your resource declaration includes a symbolic name for the resource. You'll use this symbolic name in other parts of the Bicep file to get a value from the resource.

The resource declaration includes the resource type and API version. Within the body of the resource declaration, include properties that are specific to the resource type.

```
resource stg 'Microsoft.Storage/storageAccounts@2019-06-01' = {
  name: uniqueStorageName
  location: location
  sku: {
    name: storageSKU
  }
  kind: 'StorageV2'
  properties: {
    supportsHttpsTrafficOnly: true
  }
}
```

For more information, see Resource declaration in Bicep.

Some resources have a parent/child relationship. You can define a child resource either inside the parent resource or outside of it.

The following example shows how to define a child resource within a parent resource. It contains a storage account with a child resource (file service) that is defined within the storage account. The file service also has a child resource (share) that is defined within it.

```
resource storage 'Microsoft.Storage/storageAccounts@2021-02-01' = {
  name: 'examplestorage'
  location: resourceGroup().location
  kind: 'StorageV2'
  sku: {
    name: 'Standard_LRS'
  }

  resource service 'fileServices' = {
    name: 'default'

    resource share 'shares' = {
      name: 'exampleshare'
    }
  }
}
```

The next example shows how to define a child resource outside of the parent resource. You use the parent property to identify a parent/child relationship. The same three resources are defined.

```
resource storage 'Microsoft.Storage/storageAccounts@2021-02-01' = {
  name: 'examplestorage'
  location: resourceGroup().location
  kind: 'StorageV2'
  sku: {
    name: 'Standard_LRS'
  }
}

resource service 'Microsoft.Storage/storageAccounts/fileServices@2021-02-01' = {
  name: 'default'
  parent: storage
}

resource share 'Microsoft.Storage/storageAccounts/fileServices/shares@2021-02-01' = {
  name: 'exampleshare'
  parent: service
}
```

For more information, see Set name and type for child resources in Bicep.

Modules

Modules enable you to reuse code from a Bicep file in other Bicep files. In the module declaration, you link to the file to reuse. When you deploy the Bicep file, the resources in the module are also deployed.

```
module webModule './webApp.bicep' = {
  name: 'webDeploy'
  params: {
    skuName: 'S1'
    location: location
  }
}
```

The symbolic name enables you to reference the module from somewhere else in the file. For example, you can get an output value from a module by using the symbolic name and the name of the output value.

For more information, see Use Bicep modules.

Resource and module decorators

You can add a decorator to a resource or module definition. The only supported decorator is `batchSize(int)`. You can only apply it to a resource or module definition that uses a `for` expression.

By default, resources are deployed in parallel. When you add the `batchSize` decorator, you deploy instances serially.

```
@batchSize(3)
resource storageAccountResources 'Microsoft.Storage/storageAccounts@2019-06-01' = [for storageName in
storageAccounts: {
  ...
}]
```

For more information, see Deploy in batches.

Outputs

Use outputs to return values from the deployment. Typically, you return a value from a deployed resource when you need to reuse that value for another operation.

```
output storageEndpoint object = stg.properties.primaryEndpoints
```

For more information, see Outputs in Bicep.

Loops

You can add iterative loops to your Bicep file to define multiple copies of a:

- resource
- module
- variable
- property
- output

Use the `for` expression to define a loop.

```
param moduleCount int = 2

module stgModule './example.bicep' = [for i in range(0, moduleCount): {
  name: '${i}deployModule'
  params: {
  }
}]
```

You can iterate over an array, object, or integer index.

For more information, see Iterative loops in Bicep.

Conditional deployment

You can add a resource or module to your Bicep file that is conditionally deployed. During deployment, the condition is evaluated and the result determines whether the resource or module is deployed. Use the `if` expression to define a conditional deployment.

```
param deployZone bool

resource dnsZone 'Microsoft.Network/dnszones@2018-05-01' = if (deployZone) {
  name: 'myZone'
  location: 'global'
}
```

For more information, see Conditional deployment in Bicep.

Whitespace

Spaces and tabs are ignored when authoring Bicep files.

Bicep is newline sensitive. For example:

```
resource sa 'Microsoft.Storage/storageAccounts@2019-06-01' = if (newOrExisting == 'new') {
  ...
}
```

Can't be written as:

```
resource sa 'Microsoft.Storage/storageAccounts@2019-06-01' =
    if (newOrExisting == 'new') {
      ...
    }
```

Define objects and arrays in multiple lines.

Comments

Use `//` for single-line comments or `/* ... */` for multi-line comments

The following example shows a single-line comment.

```
// This is your primary NIC.
resource nic1 'Microsoft.Network/networkInterfaces@2020-06-01' = {
   ...
}
```

The following example shows a multi-line comment.

```
/*
  This Bicep file assumes the key vault already exists and
  is in same subscription and resource group as the deployment.
*/
param existingKeyVaultName string
```

Multi-line strings

You can break a string into multiple lines. Use three single quote characters `'''` to start and end the multi-line string.

Characters within the multi-line string are handled as-is. Escape characters are unnecessary. You can't include `'''` in the multi-line string. String interpolation isn't currently supported.

You can either start your string right after the opening `'''` or include a new line. In either case, the resulting string doesn't include a new line. Depending on the line endings in your Bicep file, new lines are interpreted as `\r\n` or `\n`.

The following example shows a multi-line string.

```
var stringVar = '''
this is multi-line
  string with formatting
  preserved.
'''
```

The preceding example is equivalent to the following JSON.

```
"variables": {
  "stringVar": "this is multi-line\r\n  string with formatting\r\n  preserved.\r\n"
}
```

Multiple-line declarations

You can now use multiple lines in function, array and object declarations. This feature requires **Bicep version 0.7.4 or later**.

In the following example, the `resourceGroup()` definition is broken into multiple lines.

```
var foo = resourceGroup(
  mySubscription,
  myRgName)
```

See Arrays and Objects for multiple-line declaration samples.

Known limitations

- No support for the concept of apiProfile, which is used to map a single apiProfile to a set apiVersion for each resource type.
- No support for user-defined functions.
- Some Bicep features require a corresponding change to the intermediate language (Azure Resource Manager JSON templates). We announce these features as available when all of the required updates have been deployed to global Azure. If you're using a different environment, such as Azure Stack, there may be a delay in the availability of the feature. The Bicep feature is only available when the intermediate language has also been updated in that environment.

Next steps

For an introduction to Bicep, see What is Bicep?. For Bicep data types, see Data types.

Data types in Bicep

9/21/2022 • 6 minutes to read • Edit Online

This article describes the data types supported in Bicep.

Supported types

Within a Bicep, you can use these data types:

- array
- bool
- int
- object
- secureObject - indicated by modifier in Bicep
- secureString - indicated by modifier in Bicep
- string

Arrays

Arrays start with a left bracket (`[`) and end with a right bracket (`]`). In Bicep, an array can be declared in single line or multiple lines. Commas (`,`) are used between values in single-line declarations, but not used in multiple-line declarations, You can mix and match single-line and multiple-line declarations. The multiple-line declaration requires **Bicep version 0.7.4 or later**.

```
var multiLineArray = [
  'abc'
  'def'
  'ghi'
]

var singleLineArray = ['abc', 'def', 'ghi']

var mixedArray = ['abc', 'def'
    'ghi']
```

In an array, each item is represented by the any type. You can have an array where each item is the same data type, or an array that holds different data types.

The following example shows an array of integers and an array different types.

```
var integerArray = [
  1
  2
  3
]

var mixedArray = [
  resourceGroup().name
  1
  true
  'example string'
]
```

Arrays in Bicep are zero-based. In the following example, the expression `exampleArray[0]` evaluates to 1 and `exampleArray[2]` evaluates to 3. The index of the indexer may itself be another expression. The expression `exampleArray[index]` evaluates to 2. Integer indexers are only allowed on expression of array types.

```
var index = 1

var exampleArray = [
  1
  2
  3
]
```

Booleans

When specifying boolean values, use `true` or `false`. Don't surround the value with quotation marks.

```
param exampleBool bool = true
```

Integers

When specifying integer values, don't use quotation marks.

```
param exampleInt int = 1
```

In Bicep, integers are 64-bit integers. When passed as inline parameters, the range of values may be limited by the SDK or command-line tool you use for deployment. For example, when using PowerShell to deploy a Bicep, integer types can range from -2147483648 to 2147483647. To avoid this limitation, specify large integer values in a parameter file. Resource types apply their own limits for integer properties.

Floating point, decimal or binary formats aren't currently supported.

Objects

Objects start with a left brace (`{`) and end with a right brace (`}`). In Bicep, an object can be declared in single line or multiple lines. Each property in an object consists of key and value. The key and value are separated by a colon (`:`). An object allows any property of any type. Commas (`,`) are used between properties for single-line declarations, but not used between properties for multiple-line declarations. You can mix and match single-line and multiple-line declarations. The multiple-line declaration requires **Bicep version 0.7.4 or later**.

```
param singleLineObject object = {name: 'test name', id: '123-abc', isCurrent: true, tier: 1}

param multiLineObject object = {
  name: 'test name'
  id: '123-abc'
  isCurrent: true
  tier: 1
}

param mixedObject object = {name: 'test name', id: '123-abc', isCurrent: true
    tier: 1}
```

In Bicep, quotes are optionally allowed on object property keys:

```
var test = {
  'my - special. key': 'value'
}
```

In the preceding example, quotes are used when the object property keys contain special characters. For example space, '-', or '.'. The following example shows how to use interpolation in object property keys.

```
var stringVar = 'example value'
var objectVar = {
  '${stringVar}': 'this value'
}
```

Property accessors are used to access properties of an object. They're constructed using the `.` operator.

```
var a = {
  b: 'Dev'
  c: 42
  d: {
    e: true
  }
}

output result1 string = a.b // returns 'Dev'
output result2 int = a.c // returns 42
output result3 bool = a.d.e // returns true
```

Property accessors can be used with any object, including parameters and variables of object types and object literals. Using a property accessor on an expression of non-object type is an error.

You can also use the `[]` syntax to access a property. The following example returns `Development`.

```
var environmentSettings = {
  dev: {
    name: 'Development'
  }
  prod: {
    name: 'Production'
  }
}

output accessorResult string = environmentSettings['dev'].name
```

In JSON, an object is an unordered collection of zero or more key/value pairs. The ordering can be different depending on the implementations. For example, the Bicep items() function sorts the objects in the alphabetical order. In other places, the original ordering can be preserved. Because of this non-determinism, avoid making any assumptions about the ordering of object keys when writing code, which interacts with deployments parameters & outputs.

Strings

In Bicep, strings are marked with singled quotes, and must be declared on a single line. All Unicode characters with code points between *0* and *10FFFF* are allowed.

```
param exampleString string = 'test value'
```

The following table lists the set of reserved characters that must be escaped by a backslash (`\`) character:

ESCAPE SEQUENCE	REPRESENTED VALUE	NOTES
`\\`	`\`	
`\'`	`'`	
`\n`	line feed (LF)	
`\r`	carriage return (CR)	
`\t`	tab character	
`\u{x}`	Unicode code point `x`	**x** represents a hexadecimal code point value between *0* and *10FFFF* (both inclusive). Leading zeros are allowed. Code points above *FFFF* are emitted as a surrogate pair.
`\$`	`$`	Only escape when followed by `{`.

```
// evaluates to "what's up?"
var myVar = 'what\'s up?'
```

All strings in Bicep support interpolation. To inject an expression, surround it by `${` and `}`. Expressions that are referenced can't span multiple lines.

```
var storageName = 'storage${uniqueString(resourceGroup().id)}'
```

Multi-line strings

In Bicep, multi-line strings are defined between three single quote characters (`'''`) followed optionally by a newline (the opening sequence), and three single quote characters (`'''` - the closing sequence). Characters that are entered between the opening and closing sequence are read verbatim, and no escaping is necessary or possible.

> **NOTE**
>
> Because the Bicep parser reads all characters as is, depending on the line endings of your Bicep file, newlines can be interpreted as either `\r\n` or `\n`. Interpolation is not currently supported in multi-line strings. Multi-line strings containing `'''` are not supported.

```
// evaluates to "hello!"
var myVar = '''hello!'''

// evaluates to "hello!" because the first newline is skipped
var myVar2 = '''
hello!'''

// evaluates to "hello!\n" because the final newline is included
var myVar3 = '''
hello!
'''

// evaluates to "  this\n    is\n      indented\n"
var myVar4 = '''
  this
    is
      indented
'''

// evaluates to "comments // are included\n/* because everything is read as-is */\n"
var myVar5 = '''
comments // are included
/* because everything is read as-is */
'''

// evaluates to "interpolation\nis ${blocked}"
// note ${blocked} is part of the string, and is not evaluated as an expression
myVar6 = '''interpolation
is ${blocked}'''
```

Secure strings and objects

Secure string uses the same format as string, and secure object uses the same format as object. With Bicep, you add the `@secure()` modifier to a string or object.

When you set a parameter to a secure string or secure object, the value of the parameter isn't saved to the deployment history and isn't logged. However, if you set that secure value to a property that isn't expecting a secure value, the value isn't protected. For example, if you set a secure string to a tag, that value is stored as plain text. Use secure strings for passwords and secrets.

The following example shows two secure parameters:

```
@secure()
param password string

@secure()
param configValues object
```

Data type assignability

In Bicep, a value of one type (source type) can be assigned to another type (target type). The following table shows which source type (listed horizontally) can or can't be assigned to which target type (listed vertically). In the table, `X` means assignable, empty space means not assignable, and `?` means only if they types are compatible.

TYPES	`ANY`	`ERROR`	`STRING`	`NUMBER`	`INT`	`BOOL`	`NULL`	`OBJECT`	`ARRAY`	NAMED RESOURCE	NAMED MODULE	`SCOPE`
`any`	X		X	X	X	X	X	X	X	X	X	X
`error`												
`string`	X		X									
`number`	X			X	X							
`int`	X				X							
`bool`	X					X						
`null`	X						X					
`object`	X							X				
`array`	X								X			
`resource`	X									X		
`module`	X										X	
`scope`												?
named resource	X							?		?		
named module	X							?			?	

Next steps

To learn about the structure and syntax of Bicep, see Bicep file structure.

Parameters in Bicep

9/21/2022 • 6 minutes to read • Edit Online

This article describes how to define and use parameters in a Bicep file. By providing different values for parameters, you can reuse a Bicep file for different environments.

Resource Manager resolves parameter values before starting the deployment operations. Wherever the parameter is used, Resource Manager replaces it with the resolved value.

Each parameter must be set to one of the data types.

You are limited to 256 parameters. For more information, see Template limits.

For parameter best practices, see Parameters.

Training resources

If you would rather learn about parameters through step-by-step guidance, see Build reusable Bicep templates by using parameters.

Declaration

Each parameter has a name and data type. Optionally, you can provide a default value for the parameter.

```
param <parameter-name> <parameter-data-type> = <default-value>
```

A parameter can't have the same name as a variable, resource, output, or other parameter in the same scope.

The following example shows basic declarations of parameters.

```
param demoString string
param demoInt int
param demoBool bool
param demoObject object
param demoArray array
```

Default value

You can specify a default value for a parameter. The default value is used when a value isn't provided during deployment.

```
param demoParam string = 'Contoso'
```

You can use expressions with the default value. Expressions aren't allowed with other parameter properties. You can't use the reference function or any of the list functions in the parameters section. These functions get the resource's runtime state, and can't be executed before deployment when parameters are resolved.

```
param location string = resourceGroup().location
```

You can use another parameter value to build a default value. The following template constructs a host plan name from the site name.

```
param siteName string = 'site${uniqueString(resourceGroup().id)}'
param hostingPlanName string = '${siteName}-plan'

output siteNameOutput string = siteName
output hostingPlanOutput string = hostingPlanName
```

Decorators

Parameters use decorators for constraints or metadata. The decorators are in the format `@expression` and are placed above the parameter's declaration. You can mark a parameter as secure, specify allowed values, set the minimum and maximum length for a string, set the minimum and maximum value for an integer, and provide a description of the parameter.

The following example shows two common uses for decorators.

```
@secure()
param demoPassword string

@description('Must be at least Standard_A3 to support 2 NICs.')
param virtualMachineSize string = 'Standard_DS1_v2'
```

The following table describes the available decorators and how to use them.

DECORATOR	APPLY TO	ARGUMENT	DESCRIPTION
allowed	all	array	Allowed values for the parameter. Use this decorator to make sure the user provides correct values.
description	all	string	Text that explains how to use the parameter. The description is displayed to users through the portal.
maxLength	array, string	int	The maximum length for string and array parameters. The value is inclusive.
maxValue	int	int	The maximum value for the integer parameter. This value is inclusive.
metadata	all	object	Custom properties to apply to the parameter. Can include a description property that is equivalent to the description decorator.
minLength	array, string	int	The minimum length for string and array parameters. The value is inclusive.

DECORATOR	APPLY TO	ARGUMENT	DESCRIPTION
minValue	int	int	The minimum value for the integer parameter. This value is inclusive.
secure	string, object	none	Marks the parameter as secure. The value for a secure parameter isn't saved to the deployment history and isn't logged. For more information, see Secure strings and objects.

Decorators are in the sys namespace. If you need to differentiate a decorator from another item with the same name, preface the decorator with `sys`. For example, if your Bicep file includes a parameter named `description`, you must add the sys namespace when using the **description** decorator.

```
@sys.description('The name of the instance.')
param name string
@sys.description('The description of the instance to display.')
param description string
```

The available decorators are described in the following sections.

Secure parameters

You can mark string or object parameters as secure. The value of a secure parameter isn't saved to the deployment history and isn't logged.

```
@secure()
param demoPassword string

@secure()
param demoSecretObject object
```

Allowed values

You can define allowed values for a parameter. You provide the allowed values in an array. The deployment fails during validation if a value is passed in for the parameter that isn't one of the allowed values.

```
@allowed([
  'one'
  'two'
])
param demoEnum string
```

Length constraints

You can specify minimum and maximum lengths for string and array parameters. You can set one or both constraints. For strings, the length indicates the number of characters. For arrays, the length indicates the number of items in the array.

The following example declares two parameters. One parameter is for a storage account name that must have 3-24 characters. The other parameter is an array that must have from 1-5 items.

```
@minLength(3)
@maxLength(24)
param storageAccountName string

@minLength(1)
@maxLength(5)
param appNames array
```

Integer constraints

You can set minimum and maximum values for integer parameters. You can set one or both constraints.

```
@minValue(1)
@maxValue(12)
param month int
```

Description

To help users understand the value to provide, add a description to the parameter. When a user deploys the template through the portal, the description's text is automatically used as a tip for that parameter. Only add a description when the text provides more information than can be inferred from the parameter name.

```
@description('Must be at least Standard_A3 to support 2 NICs.')
param virtualMachineSize string = 'Standard_DS1_v2'
```

Markdown-formatted text can be used for the description text:

```
@description('''
Storage account name restrictions:
- Storage account names must be between 3 and 24 characters in length and may contain numbers and lowercase
letters only.
- Your storage account name must be unique within Azure. No two storage accounts can have the same name.
''')
@minLength(3)
@maxLength(24)
param storageAccountName string
```

When you hover your cursor over **storageAccountName** in VSCode, you see the formatted text:

Make sure the text is well-formatted Markdown. Otherwise the text won't be rendered correctly.

Metadata

If you have custom properties that you want to apply to a parameter, add a metadata decorator. Within the metadata, define an object with the custom names and values. The object you define for the metadata can contain properties of any name and type.

You might use this decorator to track information about the parameter that doesn't make sense to add to the description.

```
@description('Configuration values that are applied when the application starts.')
@metadata({
  source: 'database'
  contact: 'Web team'
})
param settings object
```

Use parameter

To reference the value for a parameter, use the parameter name. The following example uses a parameter value for a key vault name.

```
param vaultName string = 'keyVault${uniqueString(resourceGroup().id)}'

resource keyvault 'Microsoft.KeyVault/vaults@2019-09-01' = {
  name: vaultName
  ...
}
```

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.

The following example shows a parameter that is an object. The default value shows the expected properties for the object. Those properties are used when defining the resource to deploy.

```
param vNetSettings object = {
  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'
    }
  ]
}

resource vnet 'Microsoft.Network/virtualNetworks@2020-06-01' = {
  name: vNetSettings.name
  location: vNetSettings.location
  properties: {
    addressSpace: {
      addressPrefixes: [
        vNetSettings.addressPrefixes[0].addressPrefix
      ]
    }
    subnets: [
      {
        name: vNetSettings.subnets[0].name
        properties: {
          addressPrefix: vNetSettings.subnets[0].addressPrefix
        }
      }
      {
        name: vNetSettings.subnets[1].name
        properties: {
          addressPrefix: vNetSettings.subnets[1].addressPrefix
        }
      }
    ]
  }
}
```

Next steps

- To learn about the available properties for parameters, see Understand the structure and syntax of Bicep files.
- To learn about passing in parameter values as a file, see Create a Bicep parameter file.

Variables in Bicep

9/21/2022 • 2 minutes to read • Edit Online

This article describes how to define and use variables in your Bicep file. You use variables to simplify your Bicep file development. Rather than repeating complicated expressions throughout your Bicep file, you define a variable that contains the complicated expression. Then, you use that variable as needed throughout your Bicep file.

Resource Manager resolves variables before starting the deployment operations. Wherever the variable is used in the Bicep file, Resource Manager replaces it with the resolved value.

Define variable

The syntax for defining a variable is:

```
var <variable-name> = <variable-value>
```

A variable can't have the same name as a parameter, module, or resource.

Notice that you don't specify a data type for the variable. The type is inferred from the value. The following example sets a variable to a string.

```
var stringVar = 'example value'
```

You can use the value from a parameter or another variable when constructing the variable.

```
param inputValue string = 'deployment parameter'

var stringVar = 'preset variable'
var concatToVar =  '${stringVar}AddToVar'
var concatToParam = '${inputValue}AddToParam'

output addToVar string = concatToVar
output addToParam string = concatToParam
```

The preceding example returns:

```
{
  "addToParam": {
    "type": "String",
    "value": "deployment parameterAddToParam"
  },
  "addToVar": {
    "type": "String",
    "value": "preset variableAddToVar"
  }
}
```

You can use Bicep functions to construct the variable value. The following example uses Bicep functions to create a string value for a storage account name.

```
param storageNamePrefix string = 'stg'
var storageName = '${toLower(storageNamePrefix)}${uniqueString(resourceGroup().id)}'

output uniqueStorageName string = storageName
```

The preceding example returns a value like the following:

```
"uniqueStorageName": {
  "type": "String",
  "value": "stghzuunrvapn6sw"
}
```

You can use iterative loops when defining a variable. The following example creates an array of objects with three properties.

```
param itemCount int = 3

var objectArray = [for i in range(0, itemCount): {
  name: 'myDataDisk${(i + 1)}'
  diskSizeGB: '1'
  diskIndex: i
}]

output arrayResult array = objectArray
```

The output returns an array with the following values:

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

For more information about the types of loops you can use with variables, see Iterative loops in Bicep.

Use variable

The following example shows how to use the variable for a resource property. You reference the value for the variable by providing the variable's name: `storageName`.

```
param rgLocation string
param storageNamePrefix string = 'STG'

var storageName = '${toLower(storageNamePrefix)}${uniqueString(resourceGroup().id)}'

resource demoAccount 'Microsoft.Storage/storageAccounts@2021-02-01' = {
  name: storageName
  location: rgLocation
  kind: 'Storage'
  sku: {
    name: 'Standard_LRS'
  }
}

output stgOutput string = storageName
```

Because storage account names must use lowercase letters, the `storageName` variable uses the `toLower` function to make the `storageNamePrefix` value lowercase. The `uniqueString` function creates a unique value from the resource group ID. The values are concatenated to a string.

Configuration variables

You can define variables that hold related values for configuring an environment. You define the variable as an object with the values. The following example shows an object that holds values for two environments - **test** and **prod**. Pass in one of these values during deployment.

```
@allowed([
  'test'
  'prod'
])
param environmentName string

var environmentSettings = {
  test: {
    instanceSize: 'Small'
    instanceCount: 1
  }
  prod: {
    instanceSize: 'Large'
    instanceCount: 4
  }
}

output instanceSize string = environmentSettings[environmentName].instanceSize
output instanceCount int = environmentSettings[environmentName].instanceCount
```

Next steps

- To learn about the available properties for variables, see Understand the structure and syntax of Bicep files.
- To learn about using loop syntax, see Iterative loops in Bicep.

Resource declaration in Bicep

9/21/2022 • 3 minutes to read • Edit Online

This article describes the syntax you use to add a resource to your Bicep file.

Declaration

Add a resource declaration by using the `resource` keyword. You set a symbolic name for the resource. The symbolic name isn't the same as the resource name. You use the symbolic name to reference the resource in other parts of your Bicep file.

```
resource <symbolic-name> '<full-type-name>@<api-version>' = {
  <resource-properties>
}
```

So, a declaration for a storage account can start with:

```
resource stg 'Microsoft.Storage/storageAccounts@2021-04-01' = {
  ...
}
```

Symbolic names are case-sensitive. They may contain letters, numbers, and underscores (`_`). They can't start with a number. A resource can't have the same name as a parameter, variable, or module.

For the available resource types and version, see Bicep resource reference. Bicep doesn't support `apiProfile`, which is available in Azure Resource Manager templates (ARM templates) JSON.

To conditionally deploy a resource, use the `if` syntax. For more information, see Conditional deployment in Bicep.

```
resource <symbolic-name> '<full-type-name>@<api-version>' = if (condition) {
  <resource-properties>
}
```

To deploy more than one instance of a resource, use the `for` syntax. You can use the `batchSize` decorator to specify whether the instances are deployed serially or in parallel. For more information, see Iterative loops in Bicep.

```
@batchSize(int) // optional decorator for serial deployment
resource <symbolic-name> '<full-type-name>@<api-version>' = [for <item> in <collection>: {
  <properties-to-repeat>
}]
```

You can also use the `for` syntax on the resource properties to create an array.

```
resource <symbolic-name> '<full-type-name>@<api-version>' = {
  properties: {
    <array-property>: [for <item> in <collection>: <value-to-repeat>]
  }
}
```

Resource name

Each resource has a name. When setting the resource name, pay attention to the rules and restrictions for resource names.

```
resource stg 'Microsoft.Storage/storageAccounts@2019-06-01' = {
  name: 'examplestorage'
  ...
}
```

Typically, you'd set the name to a parameter so you can pass in different values during deployment.

```
@minLength(3)
@maxLength(24)
param storageAccountName string

resource stg 'Microsoft.Storage/storageAccounts@2019-06-01' = {
  name: storageAccountName
  ...
}
```

Location

Many resources require a location. You can determine if the resource needs a location either through intellisense or template reference. The following example adds a location parameter that is used for the storage account.

```
resource stg 'Microsoft.Storage/storageAccounts@2019-06-01' = {
  name: 'examplestorage'
  location: 'eastus'
  ...
}
```

Typically, you'd set location to a parameter so you can deploy to different locations.

```
param location string = resourceGroup().location

resource stg 'Microsoft.Storage/storageAccounts@2019-06-01' = {
  name: 'examplestorage'
  location: location
  ...
}
```

Different resource types are supported in different locations. To get the supported locations for an Azure service, See Products available by region. To get the supported locations for a resource type, use Azure PowerShell or Azure CLI.

- PowerShell
- Azure CLI

```
((Get-AzResourceProvider -ProviderNamespace Microsoft.Batch).ResourceTypes `
  | Where-Object ResourceTypeName -eq batchAccounts).Locations
```

Tags

You can apply tags to a resource during deployment. Tags help you logically organize your deployed resources. For examples of the different ways you can specify the tags, see ARM template tags.

Managed identities for Azure resources

Some resources support managed identities for Azure resources. Those resources have an identity object at the root level of the resource declaration.

You can use either system-assigned or user-assigned identities.

The following example shows how to configure a system-assigned identity for an Azure Kubernetes Service cluster.

```
resource aks 'Microsoft.ContainerService/managedClusters@2020-09-01' = {
  name: clusterName
  location: location
  tags: tags
  identity: {
    type: 'SystemAssigned'
  }
```

The next example shows how to configure a user-assigned identity for a virtual machine.

```
param userAssignedIdentity string

resource vm 'Microsoft.Compute/virtualMachines@2020-06-01' = {
  name: vmName
  location: location
  identity: {
    type: 'UserAssigned'
    userAssignedIdentities: {
      '${userAssignedIdentity}': {}
    }
  }
```

Resource-specific properties

The preceding properties are generic to most resource types. After setting those values, you need to set the properties that are specific to the resource type you're deploying.

Use intellisense or Bicep resource reference to determine which properties are available and which ones are required. The following example sets the remaining properties for a storage account.

```
resource stg 'Microsoft.Storage/storageAccounts@2019-06-01' = {
  name: 'examplestorage'
  location: 'eastus'
  sku: {
    name: 'Standard_LRS'
    tier: 'Standard'
  }
  kind: 'StorageV2'
  properties: {
    accessTier: 'Hot'
  }
}
```

Next steps

- To conditionally deploy a resource, see Conditional deployment in Bicep.
- To reference an existing resource, see Existing resources in Bicep.
- To learn about how deployment order is determined, see Resource dependencies in Bicep.

Existing resources in Bicep

9/21/2022 • 2 minutes to read • Edit Online

To reference an existing resource that isn't deployed in your current Bicep file, declare the resource with the `existing` keyword. Use the `existing` keyword when you're deploying a resource that needs to get a value from an existing resource. You access the existing resource's properties through its symbolic name.

The resource isn't redeployed when referenced with the `existing` keyword.

Same scope

The following example gets an existing storage account in the same resource group as the current deployment. Notice that you provide only the name of the existing resource. The properties are available through the symbolic name.

```
resource stg 'Microsoft.Storage/storageAccounts@2019-06-01' existing = {
  name: 'examplestorage'
}

output blobEndpoint string = stg.properties.primaryEndpoints.blob
```

Different scope

Set the `scope` property to access a resource in a different scope. The following example references an existing storage account in a different resource group.

```
resource stg 'Microsoft.Storage/storageAccounts@2019-06-01' existing = {
  name: 'examplestorage'
  scope: resourceGroup(exampleRG)
}

output blobEndpoint string = stg.properties.primaryEndpoints.blob
```

For more information about setting the scope, see Scope functions for Bicep.

Troubleshooting

If you attempt to reference a resource that doesn't exist, you get the `NotFound` error and your deployment fails. Check the name and scope of the resource you're trying to reference.

Next steps

For the syntax to deploy a resource, see Resource declaration in Bicep.

Set name and type for child resources in Bicep

9/21/2022 • 4 minutes to read • Edit Online

Child resources are resources that exist only within the context of another resource. For example, a virtual machine extension can't exist without a virtual machine. The extension resource is a child of the virtual machine.

Each parent resource accepts only certain resource types as child resources. The hierarchy of resource types is available in the Bicep resource reference.

This article show different ways you can declare a child resource.

Training resources

If you would rather learn about about child resources through step-by-step guidance, see Deploy child and extension resources by using Bicep.

Name and type pattern

In Bicep, you can specify the child resource either within the parent resource or outside of the parent resource. The values you provide for the resource name and resource type vary based on how you declare the child resource. However, the full name and type always resolve to the same pattern.

The **full name** of the child resource uses the pattern:

```
{parent-resource-name}/{child-resource-name}
```

If you have more two levels in the hierarchy, keep repeating parent names:

```
{parent-resource-name}/{child-level1-resource-name}/{child-level2-resource-name}
```

The **full type** of the child resource uses the pattern:

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

If you have more than two levels in the hierarchy, keep repeating parent resource types:

```
{resource-provider-namespace}/{parent-resource-type}/{child-level1-resource-type}/{child-level2-resource-type}
```

If you count the segments between `/` characters, the number of segments in the type is always one more than the number of segments in the name.

Within parent resource

The following example shows the child resource included within the resources property of the parent resource.

```
resource <parent-resource-symbolic-name> '<resource-type>@<api-version>' = {
  <parent-resource-properties>

  resource <child-resource-symbolic-name> '<child-resource-type>' = {
    <child-resource-properties>
  }
}
```

A nested resource declaration must appear at the top level of syntax of the parent resource. Declarations may be nested arbitrarily deep, as long as each level is a child type of its parent resource.

When defined within the parent resource type, you format the type and name values as a single segment without slashes. The following example shows a storage account with a child resource for the file service, and the file service has a child resource for the file share. The file service's name is set to `default` and its type is set to `fileServices`. The file share's name is set `exampleshare` and its type is set to `shares`.

```
resource storage 'Microsoft.Storage/storageAccounts@2021-02-01' = {
  name: 'examplestorage'
  location: resourceGroup().location
  kind: 'StorageV2'
  sku: {
    name: 'Standard_LRS'
  }

  resource service 'fileServices' = {
    name: 'default'

    resource share 'shares' = {
      name: 'exampleshare'
    }
  }
}
```

The full resource types are still `Microsoft.Storage/storageAccounts/fileServices` and `Microsoft.Storage/storageAccounts/fileServices/shares`. You don't provide `Microsoft.Storage/storageAccounts/` because it's assumed from the parent resource type and version. The nested resource may optionally declare an API version using the syntax `<segment>@<version>`. If the nested resource omits the API version, the API version of the parent resource is used. If the nested resource specifies an API version, the API version specified is used.

The child resource names are set to `default` and `exampleshare` but the full names include the parent names. You don't provide `examplestorage` or `default` because they're assumed from the parent resource.

A nested resource can access properties of its parent resource. Other resources declared inside the body of the same parent resource can reference each other by using the symbolic names. A parent resource may not access properties of the resources it contains, this attempt would cause a cyclic-dependency.

To reference a nested resource outside the parent resource, it must be qualified with the containing resource name and the `::` operator. For example, to output a property from a child resource:

```
output childAddressPrefix string = VNet1::VNet1_Subnet1.properties.addressPrefix
```

Outside parent resource

The following example shows the child resource outside of the parent resource. You might use this approach if the parent resource isn't deployed in the same template, or if want to use a loop to create more than one child resource. Specify the parent property on the child with the value set to the symbolic name of the parent. With

this syntax you still need to declare the full resource type, but the name of the child resource is only the name of the child.

```
resource <parent-resource-symbolic-name> '<resource-type>@<api-version>' = {
  name: 'myParent'
  <parent-resource-properties>
}

resource <child-resource-symbolic-name> '<child-resource-type>@<api-version>' = {
  parent: <parent-resource-symbolic-name>
  name: 'myChild'
  <child-resource-properties>
}
```

When defined outside of the parent resource, you format the type and with slashes to include the parent type and name.

The following example shows a storage account, file service, and file share that are all defined at the root level.

```
resource storage 'Microsoft.Storage/storageAccounts@2021-02-01' = {
  name: 'examplestorage'
  location: resourceGroup().location
  kind: 'StorageV2'
  sku: {
    name: 'Standard_LRS'
  }
}

resource service 'Microsoft.Storage/storageAccounts/fileServices@2021-02-01' = {
  name: 'default'
  parent: storage
}

resource share 'Microsoft.Storage/storageAccounts/fileServices/shares@2021-02-01' = {
  name: 'exampleshare'
  parent: service
}
```

Referencing the child resource symbolic name works the same as referencing the parent.

Full resource name outside parent

You can also use the full resource name and type when declaring the child resource outside the parent. You don't set the parent property on the child resource. Because the dependency can't be inferred, you must set it explicitly.

```
resource storage 'Microsoft.Storage/storageAccounts@2021-02-01' = {
  name: 'examplestorage'
  location: resourceGroup().location
  kind: 'StorageV2'
  sku: {
    name: 'Standard_LRS'
  }
}

resource service 'Microsoft.Storage/storageAccounts/fileServices@2021-02-01' = {
  name: 'examplestorage/default'
  dependsOn: [
    storage
  ]
}

resource share 'Microsoft.Storage/storageAccounts/fileServices/shares@2021-02-01' = {
  name: 'examplestorage/default/exampleshare'
  dependsOn: [
    service
  ]
}
```

IMPORTANT

Setting the full resource name and type isn't the recommended approach. It's not as type safe as using one of the other approaches.

Next steps

- To learn about creating Bicep files, see Understand the structure and syntax of Bicep files.
- To learn about the format of the resource name when referencing the resource, see the reference function.

Set scope for extension resources in Bicep

9/21/2022 • 3 minutes to read • Edit Online

An extension resource is a resource that modifies another resource. For example, you can assign a role to a resource. The role assignment is an extension resource type.

For a full list of extension resource types, see Resource types that extend capabilities of other resources.

This article shows how to set the scope for an extension resource type when deployed with a Bicep file. It describes the scope property that is available for extension resources when applying to a resource.

> **NOTE**
>
> The scope property is only available to extension resource types. To specify a different scope for a resource type that isn't an extension type, use a module.

Training resources

If you would rather learn about extension resources through step-by-step guidance, see Deploy child and extension resources by using Bicep.

Apply at deployment scope

To apply an extension resource type at the target deployment scope, add the resource to your template as you would with any other resource type. The available scopes are resource group, subscription, management group, and tenant. The deployment scope must support the resource type.

When deployed to a resource group, the following template adds a lock to that resource group.

```
resource createRgLock 'Microsoft.Authorization/locks@2016-09-01' = {
  name: 'rgLock'
  properties: {
    level: 'CanNotDelete'
    notes: 'Resource group should not be deleted.'
  }
}
```

The next example assigns a role to the subscription it's deployed to.

```
targetScope = 'subscription'

@description('The principal to assign the role to')
param principalId string

@allowed([
  'Owner'
  'Contributor'
  'Reader'
])
@description('Built-in role to assign')
param builtInRoleType string

var role = {
  Owner:
'/subscriptions/${subscription().subscriptionId}/providers/Microsoft.Authorization/roleDefinitions/8e3af657-
a8ff-443c-a75c-2fe8c4bcb635'
  Contributor:
'/subscriptions/${subscription().subscriptionId}/providers/Microsoft.Authorization/roleDefinitions/b24988ac-
6180-42a0-ab88-20f7382dd24c'
  Reader:
'/subscriptions/${subscription().subscriptionId}/providers/Microsoft.Authorization/roleDefinitions/acdd72a7-
3385-48ef-bd42-f606fba81ae7'
}

resource roleAssignSub 'Microsoft.Authorization/roleAssignments@2020-04-01-preview' = {
  name: guid(subscription().id, principalId, role[builtInRoleType])
  properties: {
    roleDefinitionId: role[builtInRoleType]
    principalId: principalId
  }
}
```

Apply to resource

To apply an extension resource to a resource, use the `scope` property. In the scope property, reference the resource you're adding the extension to. You reference the resource by providing the symbolic name for the resource. The scope property is a root property for the extension resource type.

The following example creates a storage account and applies a role to it.

```
@description('The principal to assign the role to')
param principalId string

@allowed([
  'Owner'
  'Contributor'
  'Reader'
])
@description('Built-in role to assign')
param builtInRoleType string

param location string = resourceGroup().location

var role = {
  Owner:
'/subscriptions/${subscription().subscriptionId}/providers/Microsoft.Authorization/roleDefinitions/8e3af657-
a8ff-443c-a75c-2fe8c4bcb635'
  Contributor:
'/subscriptions/${subscription().subscriptionId}/providers/Microsoft.Authorization/roleDefinitions/b24988ac-
6180-42a0-ab88-20f7382dd24c'
  Reader:
'/subscriptions/${subscription().subscriptionId}/providers/Microsoft.Authorization/roleDefinitions/acdd72a7-
3385-48ef-bd42-f606fba81ae7'
}
var uniqueStorageName = 'storage${uniqueString(resourceGroup().id)}'

resource demoStorageAcct 'Microsoft.Storage/storageAccounts@2019-04-01' = {
  name: uniqueStorageName
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'Storage'
  properties: {}
}

resource roleAssignStorage 'Microsoft.Authorization/roleAssignments@2020-04-01-preview' = {
  name: guid(demoStorageAcct.id, principalId, role[builtInRoleType])
  properties: {
    roleDefinitionId: role[builtInRoleType]
    principalId: principalId
  }
  scope: demoStorageAcct
}
```

You can apply an extension resource to an existing resource. The following example adds a lock to an existing storage account.

```
resource demoStorageAcct 'Microsoft.Storage/storageAccounts@2021-04-01' existing = {
  name: 'examplestore'
}

resource createStorageLock 'Microsoft.Authorization/locks@2016-09-01' = {
  name: 'storeLock'
  scope: demoStorageAcct
  properties: {
    level: 'CanNotDelete'
    notes: 'Storage account should not be deleted.'
  }
}
```

The same requirements apply to extension resources as other resource when targeting a scope that is different than the target scope of the deployment. To learn about deploying to more than one scope, see:

- Resource group deployments
- Subscription deployments
- Management group deployments
- Tenant deployments

The resourceGroup and subscription properties are only allowed on modules. These properties are not allowed on individual resources. Use modules if you want to deploy an extension resource with the scope set to a resource in a different resource group.

The following example shows how to apply a lock on a storage account that resides in a different resource group.

- **main.bicep**:

```
param resourceGroup2Name string
param storageAccountName string

module applyStoreLock './storageLock.bicep' = {
  name: 'addStorageLock'
  scope: resourceGroup(resourceGroup2Name)
  params: {
    storageAccountName: storageAccountName
  }
}
```

- **storageLock.bicep**:

```
param storageAccountName string

resource storage 'Microsoft.Storage/storageAccounts@2021-09-01' existing = {
  name: storageAccountName
}

resource storeLock 'Microsoft.Authorization/locks@2017-04-01' = {
  scope: storage
  name: 'storeLock'
  properties: {
    level: 'CanNotDelete'
    notes: 'Storage account should not be deleted.'
  }
}
```

Next steps

For a full list of extension resource types, see Resource types that extend capabilities of other resources.

Resource dependencies in Bicep

9/21/2022 • 2 minutes to read • Edit Online

When deploying resources, you may need to make sure some resources are deployed before other resources. For example, you need a logical SQL server before deploying a database. You establish this relationship by marking one resource as dependent on the other resource. The order of resource deployment is determined in two ways: implicit dependency and explicit dependency

Azure 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 Bicep file.

Implicit dependency

An implicit dependency is created when one resource declaration references another resource in the same deployment. In the following example, `otherResource` gets a property from `exampleDnsZone`. The resource named `otherResource` is implicitly dependent on `exampleDnsZone`.

```
resource exampleDnsZone 'Microsoft.Network/dnszones@2018-05-01' = {
  name: 'myZone'
  location: 'global'
}

resource otherResource 'Microsoft.Example/examples@2020-06-01' = {
  name: 'exampleResource'
  properties: {
    // get read-only DNS zone property
    nameServers: exampleDnsZone.properties.nameServers
  }
}
```

A nested resource also has an implicit dependency on its containing resource.

```
resource myParent 'My.Rp/parentType@2020-01-01' = {
  name: 'myParent'
  location: 'West US'

  // implicit dependency on 'myParent'
  resource myChild 'childType' = {
    name: 'myChild'
  }
}
```

When an implicit dependency exists, **don't add an explicit dependency**.

For more information about nested resources, see Set name and type for child resources in Bicep.

Explicit dependency

An explicit dependency is declared with the `dependsOn` property. The property accepts an array of resource identifiers, so you can specify more than one dependency.

The following example shows a DNS zone named `otherZone` that depends on a DNS zone named `dnsZone`:

```
resource dnsZone 'Microsoft.Network/dnszones@2018-05-01' = {
  name: 'demoeZone1'
  location: 'global'
}

resource otherZone 'Microsoft.Network/dnszones@2018-05-01' = {
  name: 'demoZone2'
  location: 'global'
  dependsOn: [
    dnsZone
  ]
}
```

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. After deployment, the resource doesn't retain deployment dependencies in its properties, so there are no commands or operations that let you see dependencies. Setting unnecessary dependencies slows deployment time because Resource Manager can't deploy those resources in parallel.

Even though explicit dependencies are sometimes required, the need for them is rare. In most cases, you can use a symbolic name to imply the dependency between resources. If you find yourself setting explicit dependencies, you should consider if there's a way to remove it.

Visualize dependencies

Visual Studio Code provides a tool for visualizing the dependencies. Open a Bicep file in Visual Studio Code, and select the visualizer button on the upper left corner. The following screenshot shows the dependencies of a virtual machine.

Next steps

For the syntax to deploy a resource, see Resource declaration in Bicep.

Bicep modules

9/21/2022 • 10 minutes to read • Edit Online

Bicep enables you to organize deployments into modules. A module is a Bicep file (or an ARM JSON template) that is deployed from another Bicep file. With modules, you improve the readability of your Bicep files by encapsulating complex details of your deployment. You can also easily reuse modules for different deployments.

To share modules with other people in your organization, create a template spec, public registry, or private registry. Template specs and modules in the registry are only available to users with the correct permissions.

> **TIP**
>
> The choice between module registry and template specs is mostly a matter of preference. There are a few things to consider when you choose between the two:
>
> - Module registry is only supported by Bicep. If you are not yet using Bicep, use template specs.
> - Content in the Bicep module registry can only be deployed from another Bicep file. Template specs can be deployed directly from the API, Azure PowerShell, Azure CLI, and the Azure portal. You can even use `UiFormDefinition` to customize the portal deployment experience.
> - Bicep has some limited capabilities for embedding other project artifacts (including non-Bicep and non-ARM-template files. For example, PowerShell scripts, CLI scripts and other binaries) by using the `loadTextContent` and `loadFileAsBase64` functions. Template specs can't package these artifacts.

Bicep modules are converted into a single Azure Resource Manager template with nested templates.

Training resources

If you would rather learn about modules through step-by-step guidance, see Create composable Bicep files by using modules.

Definition syntax

The basic syntax for defining a module is:

```
module <symbolic-name> '<path-to-file>' = {
  name: '<linked-deployment-name>'
  params: {
    <parameter-names-and-values>
  }
}
```

So, a simple, real-world example would look like:

```
module stgModule '../storageAccount.bicep' = {
  name: 'storageDeploy'
  params: {
    storagePrefix: 'examplestg1'
  }
}
```

You can also use an ARM JSON template as a module:

```
module stgModule '../storageAccount.json' = {
  name: 'storageDeploy'
  params: {
    storagePrefix: 'examplestg1'
  }
}
```

Use the symbolic name to reference the module in another part of the Bicep file. For example, you can use the symbolic name to get the output from a module. The symbolic name may contain a-z, A-Z, 0-9, and underscore (`_`). The name can't start with a number. A module can't have the same name as a parameter, variable, or resource.

The path can be either a local file or a file in a registry. The local file can be either a Bicep file or an ARM JSON template. For more information, see Path to module.

The **name** property is required. It becomes the name of the nested deployment resource in the generated template.

If a module with a static name is deployed concurrently to the same scope, there's the potential for one deployment to interfere with the output from the other deployment. For example, if two Bicep files use the same module with the same static name (`examplemodule`) and targeted to the same resource group, one deployment may show the wrong output. If you're concerned about concurrent deployments to the same scope, give your module a unique name.

The following example concatenates the deployment name to the module name. If you provide a unique name for the deployment, the module name is also unique.

```
module stgModule 'storageAccount.bicep' = {
  name: '${deployment().name}-storageDeploy'
  scope: resourceGroup('demoRG')
```

If you need to **specify a scope** that is different than the scope for the main file, add the scope property. For more information, see Set module scope.

```
// deploy to different scope
module <symbolic-name> '<path-to-file>' = {
  name: '<linked-deployment-name>'
  scope: <scope-object>
  params: {
    <parameter-names-and-values>
  }
}
```

To **conditionally deploy a module**, add an `if` expression. The use is similar to conditionally deploying a resource.

```
// conditional deployment
module <symbolic-name> '<path-to-file>' = if (<condition-to-deploy>) {
  name: '<linked-deployment-name>'
  params: {
    <parameter-names-and-values>
  }
}
```

To deploy **more than one instance** of a module, add the `for` expression. You can use the `batchSize` decorator to specify whether the instances are deployed serially or in parallel. For more information, see

Iterative loops in Bicep.

```
// iterative deployment
@batchSize(int) // optional decorator for serial deployment
module <symbolic-name> '<path-to-file>' = [for <item> in <collection>: {
  name: '<linked-deployment-name>'
  params: {
    <parameter-names-and-values>
  }
}]
```

Like resources, modules are deployed in parallel unless they depend on other modules or resources. Typically, you don't need to set dependencies as they're determined implicitly. If you need to set an explicit dependency, you can add `dependsOn` to the module definition. To learn more about dependencies, see Resource dependencies.

```
module <symbolic-name> '<path-to-file>' = {
  name: '<linked-deployment-name>'
  params: {
    <parameter-names-and-values>
  }
  dependsOn: [
    <symbolic-names-to-deploy-before-this-item>
  ]
}
```

Path to module

The file for the module can be either a local file or an external file. The external file can be in template spec or a Bicep module registry. All of these options are shown below.

Local file

If the module is a **local file**, provide a relative path to that file. All paths in Bicep must be specified using the forward slash (/) directory separator to ensure consistent compilation across platforms. The Windows backslash (\) character is unsupported. Paths can contain spaces.

For example, to deploy a file that is up one level in the directory from your main file, use:

```
module stgModule '../storageAccount.bicep' = {
  name: 'storageDeploy'
  params: {
    storagePrefix: 'examplestg1'
  }
}
```

File in registry

Public module registry

The public module registry is hosted in a Microsoft container registry (MCR). The source code and the modules are stored in GitHub. The README file in the GitHub repo lists the available modules and their latest versions:

README.md

Bicep Registry Modules

This repo contains the source code of all currently available Bicep modules in the Bicep public module registry.

Modules

Below is a table containing all published modules. Each version badge shows the latest version of the corresponding module. You may click on a version badge to check all available versions for a module.

Module	Version	Docs
`compute/availability-set`	mcr 1.0.1	Code | Readme
`network/virtual-network`	mcr 1.0.1	Code | Readme
`samples/array-loop`	mcr 1.0.1	Code | Readme
`samples/hello-world`	mcr 1.0.2	Code | Readme

Contributing

We only accept contributions from Microsoft employees at this time. Teams within Microsoft can refer to Contributing to Bicep public registry for information on contributing modules. External customers can propose new modules or report bugs by opening an issue.

Select the versions to see the available versions. You can also select **Code** to see the module source code, and open the Readme files.

There are only a few published modules currently. More modules are coming. If you like to contribute to the registry, see the contribution guide.

To link to a public registry module, specify the module path with the following syntax:

```
module <symbolic-name> 'br/public:<file-path>:<tag>' = {}
```

- **br/public** is the alias for the public module registry. This alias is predefined in your configuration.
- **file path** can contain segments that can be separated by the `/` character.
- **tag** is used for specifying a version for the module.

For example:

```
module hw 'br/public:samples/hello-world:1.0.2' = {
  name: 'helloWorld'
  params: {
    name: 'John Dole'
  }
}
```

NOTE

br/public is the alias for the public registry. It can also be written as

```
module <symbolic-name> 'br:mcr.microsoft.com/bicep/<file-path>:<tag>' = {}
```

Private module registry

If you've published a module to a registry, you can link to that module. Provide the name for the Azure container registry and a path to the module. Specify the module path with the following syntax:

```
module <symbolic-name> 'br:<registry-name>.azurecr.io/<file-path>:<tag>' = {
```

- **br** is the scheme name for a Bicep registry.
- **file path** is called `repository` in Azure Container Registry. The **file path** can contain segments that are separated by the `/` character.
- **tag** is used for specifying a version for the module.

For example:

```
module stgModule 'br:exampleregistry.azurecr.io/bicep/modules/storage:v1' = {
  name: 'storageDeploy'
  params: {
    storagePrefix: 'examplestg1'
  }
}
```

When you reference a module in a registry, the Bicep extension in Visual Studio Code automatically calls bicep restore to copy the external module to the local cache. It takes a few moments to restore the external module. If intellisense for the module doesn't work immediately, wait for the restore to complete.

The full path for a module in a registry can be long. Instead of providing the full path each time you want to use the module, you can configure aliases in the bicepconfig.json file. The aliases make it easier to reference the module. For example, with an alias, you can shorten the path to:

```
module stgModule 'br/ContosoModules:storage:v1' = {
  name: 'storageDeploy'
  params: {
    storagePrefix: 'examplestg1'
  }
}
```

An alias for the public module registry has been predefined:

```
module hw 'br/public:samples/hello-world:1.0.2' = {
  name: 'helloWorld'
  params: {
    name: 'John Dole'
  }
}
```

You can override the public alias in the bicepconfig.json file.

File in template spec

After creating a template spec, you can link to that template spec in a module. Specify the template spec in the following format:

```
module <symbolic-name> 'ts:<sub-id>/<rg-name>/<template-spec-name>:<version>' = {
```

However, you can simplify your Bicep file by creating an alias for the resource group that contains your template specs. When you use an alias, the syntax becomes:

```
module <symbolic-name> 'ts/<alias>:<template-spec-name>:<version>' = {
```

The following module deploys a template spec to create a storage account. The subscription and resource group for the template spec is defined in the alias named **ContosoSpecs**.

```
module stgModule 'ts/ContosoSpecs:storageSpec:2.0' = {
  name: 'storageDeploy'
  params: {
    storagePrefix: 'examplestg1'
  }
}
```

Parameters

The parameters you provide in your module definition match the parameters in the Bicep file.

The following Bicep example has three parameters - storagePrefix, storageSKU, and location. The storageSKU parameter has a default value so you don't have to provide a value for that parameter during deployment.

```
@minLength(3)
@maxLength(11)
param storagePrefix string

@allowed([
  'Standard_LRS'
  'Standard_GRS'
  'Standard_RAGRS'
  'Standard_ZRS'
  'Premium_LRS'
  'Premium_ZRS'
  'Standard_GZRS'
  'Standard_RAGZRS'
])
param storageSKU string = 'Standard_LRS'

param location string

var uniqueStorageName = '${storagePrefix}${uniqueString(resourceGroup().id)}'

resource stg 'Microsoft.Storage/storageAccounts@2021-04-01' = {
  name: uniqueStorageName
  location: location
  sku: {
    name: storageSKU
  }
  kind: 'StorageV2'
  properties: {
    supportsHttpsTrafficOnly: true
  }
}

output storageEndpoint object = stg.properties.primaryEndpoints
```

To use the preceding example as a module, provide values for those parameters.

```
targetScope = 'subscription'

@minLength(3)
@maxLength(11)
param namePrefix string

resource demoRG 'Microsoft.Resources/resourceGroups@2021-04-01' existing = {
  name: 'demogroup1'
}

module stgModule '../create-storage-account/main.bicep' = {
  name: 'storageDeploy'
  scope: demoRG
  params: {
    storagePrefix: namePrefix
    location: demoRG.location
  }
}

output storageEndpoint object = stgModule.outputs.storageEndpoint
```

Set module scope

When declaring a module, you can set a scope for the module that is different than the scope for the containing Bicep file. Use the `scope` property to set the scope for the module. When the scope property isn't provided, the module is deployed at the parent's target scope.

The following Bicep file creates a resource group and a storage account in that resource group. The file is deployed to a subscription, but the module is scoped to the new resource group.

```
// set the target scope for this file
targetScope = 'subscription'

@minLength(3)
@maxLength(11)
param namePrefix string

param location string = deployment().location

var resourceGroupName = '${namePrefix}rg'

resource newRG 'Microsoft.Resources/resourceGroups@2021-04-01' = {
  name: resourceGroupName
  location: location
}

module stgModule '../create-storage-account/main.bicep' = {
  name: 'storageDeploy'
  scope: newRG
  params: {
    storagePrefix: namePrefix
    location: location
  }
}

output storageEndpoint object = stgModule.outputs.storageEndpoint
```

The next example deploys storage accounts to two different resource groups. Both of these resource groups must already exist.

```
targetScope = 'subscription'

resource firstRG 'Microsoft.Resources/resourceGroups@2021-04-01' existing = {
  name: 'demogroup1'
}

resource secondRG 'Microsoft.Resources/resourceGroups@2021-04-01' existing = {
  name: 'demogroup2'
}

module storage1 '../create-storage-account/main.bicep' = {
  name: 'westusdeploy'
  scope: firstRG
  params: {
    storagePrefix: 'stg1'
    location: 'westus'
  }
}

module storage2 '../create-storage-account/main.bicep' = {
  name: 'eastusdeploy'
  scope: secondRG
  params: {
    storagePrefix: 'stg2'
    location: 'eastus'
  }
}
```

Set the scope property to a valid scope object. If your Bicep file deploys a resource group, subscription, or management group, you can set the scope for a module to the symbolic name for that resource. Or, you can use the scope functions to get a valid scope.

Those functions are:

- resourceGroup
- subscription
- managementGroup
- tenant

The following example uses the `managementGroup` function to set the scope.

```
param managementGroupName string

module mgDeploy 'main.bicep' = {
  name: 'deployToMG'
  scope: managementGroup(managementGroupName)
}
```

Output

You can get values from a module and use them in the main Bicep file. To get an output value from a module, use the `outputs` property on the module object.

The first example creates a storage account and returns the primary endpoints.

```
@minLength(3)
@maxLength(11)
param storagePrefix string

@allowed([
  'Standard_LRS'
  'Standard_GRS'
  'Standard_RAGRS'
  'Standard_ZRS'
  'Premium_LRS'
  'Premium_ZRS'
  'Standard_GZRS'
  'Standard_RAGZRS'
])
param storageSKU string = 'Standard_LRS'

param location string

var uniqueStorageName = '${storagePrefix}${uniqueString(resourceGroup().id)}'

resource stg 'Microsoft.Storage/storageAccounts@2021-04-01' = {
  name: uniqueStorageName
  location: location
  sku: {
    name: storageSKU
  }
  kind: 'StorageV2'
  properties: {
    supportsHttpsTrafficOnly: true
  }
}

output storageEndpoint object = stg.properties.primaryEndpoints
```

When used as module, you can get that output value.

```
targetScope = 'subscription'

@minLength(3)
@maxLength(11)
param namePrefix string

resource demoRG 'Microsoft.Resources/resourceGroups@2021-04-01' existing = {
  name: 'demogroup1'
}

module stgModule '../create-storage-account/main.bicep' = {
  name: 'storageDeploy'
  scope: demoRG
  params: {
    storagePrefix: namePrefix
    location: demoRG.location
  }
}

output storageEndpoint object = stgModule.outputs.storageEndpoint
```

Next steps

- For a tutorial, see Deploy Azure resources by using Bicep templates.
- To pass a sensitive value to a module, use the getSecret function.

Outputs in Bicep

9/21/2022 • 3 minutes to read • Edit Online

This article describes how to define output values in a Bicep file. You use outputs when you need to return values from the deployed resources.

Define output values

The syntax for defining an output value is:

```
output <name> <data-type> = <value>
```

An output can have the same name as a parameter, variable, module, or resource. Each output value must resolve to one of the data types.

The following example shows how to return a property from a deployed resource. In the example, `publicIP` is the symbolic name for a public IP address that is deployed in the Bicep file. The output value gets the fully qualified domain name for the public IP address.

```
output hostname string = publicIP.properties.dnsSettings.fqdn
```

The next example shows how to return outputs of different types.

```
output stringOutput string = deployment().name
output integerOutput int = length(environment().authentication.audiences)
output booleanOutput bool = contains(deployment().name, 'demo')
output arrayOutput array = environment().authentication.audiences
output objectOutput object = subscription()
```

If you need to output a property that has a hyphen in the name, use brackets around the name instead of dot notation. For example, use `['property-name']` instead of `.property-name`.

```
var user = {
  'user-name': 'Test Person'
}

output stringOutput string = user['user-name']
```

Conditional output

When the value to return depends on a condition in the deployment, use the the `?` operator.

```
output <name> <data-type> = <condition> ? <true-value> : <false-value>
```

Typically, you use a conditional output when you've conditionally deployed a resource. The following example shows how to conditionally return the resource ID for a public IP address based on whether a new one was deployed.

To specify a conditional output in Bicep, use the `?` operator. The following example either returns an endpoint

URL or an empty string depending on a condition.

```
param deployStorage bool = true
param storageName string
param location string = resourceGroup().location

resource myStorageAccount 'Microsoft.Storage/storageAccounts@2019-06-01' = if (deployStorage) {
  name: storageName
  location: location
  kind: 'StorageV2'
  sku:{
    name:'Standard_LRS'
    tier: 'Standard'
  }
  properties: {
    accessTier: 'Hot'
  }
}

output endpoint string = deployStorage ? myStorageAccount.properties.primaryEndpoints.blob : ''
```

Dynamic number of outputs

In some scenarios, you don't know the number of instances of a value you need to return when creating the template. You can return a variable number of values by using the `for` expression.

```
output <name> <data-type> = [for <item> in <collection>: {
  ...
}]
```

The following example iterates over an array.

```
param nsgLocation string = resourceGroup().location
param orgNames array = [
  'Contoso'
  'Fabrikam'
  'Coho'
]

resource nsg 'Microsoft.Network/networkSecurityGroups@2020-06-01' = [for name in orgNames: {
  name: 'nsg-${name}'
  location: nsgLocation
}]

output deployedNSGs array = [for (name, i) in orgNames: {
  orgName: name
  nsgName: nsg[i].name
  resourceId: nsg[i].id
}]
```

For more information about loops, see Iterative loops in Bicep.

Outputs from modules

To get an output value from a module, use the following syntax:

```
<module-name>.outputs.<property-name>
```

The following example shows how to set the IP address on a load balancer by retrieving a value from a module.

```
module publicIP 'modules/public-ip-address.bicep' = {
  name: 'public-ip-address-module'
}

resource loadBalancer 'Microsoft.Network/loadBalancers@2020-11-01' = {
  name: loadBalancerName
  location: location
  properties: {
    frontendIPConfigurations: [
      {
        name: 'name'
        properties: {
          publicIPAddress: {
            id: publicIP.outputs.resourceId
          }
        }
      }
    ]
    // ...
  }
}
```

Get output values

When the deployment succeeds, the output values are automatically returned in the results of the deployment.

To get output values from the deployment history, you can use Azure CLI or Azure PowerShell script.

- PowerShell
- Azure CLI

```
(Get-AzResourceGroupDeployment `
  -ResourceGroupName <resource-group-name> `
  -Name <deployment-name>).Outputs.resourceID.value
```

Object sorting in outputs

In JSON, an object is an unordered collection of zero or more key/value pairs. The ordering can be different depending on the implementations. For example, the Bicep items() function sorts the objects in the alphabetical order. In other places, the original ordering can be preserved. Because of this non-determinism, avoid making any assumptions about the ordering of object keys when writing code, which interacts with deployments parameters & outputs.

Next steps

- To learn about the available properties for outputs, see Understand the structure and syntax of Bicep.

Iterative loops in Bicep

9/21/2022 • 7 minutes to read • Edit Online

This article shows you how to use the `for` syntax to iterate over items in a collection. This functionality is supported starting in v0.3.1 onward. You can use loops to define multiple copies of a resource, module, variable, property, or output. Use loops to avoid repeating syntax in your Bicep file and to dynamically set the number of copies to create during deployment. To go through a quickstart, see Quickstart: Create multiple instances.

Training resources

If you would rather learn about loops through step-by-step guidance, see Build flexible Bicep templates by using conditions and loops.

Loop syntax

Loops can be declared by:

- Using an **integer index**. This option works when your scenario is: "I want to create this many instances." The range function creates an array of integers that begins at the start index and contains the number of specified elements. Within the loop, you can use the integer index to modify values. For more information, see Integer index.

```
[for <index> in range(<startIndex>, <numberOfElements>): {
  ...
}]
```

- Using **items in an array**. This option works when your scenario is: "I want to create an instance for each element in an array." Within the loop, you can use the value of the current array element to modify values. For more information, see Array elements.

```
[for <item> in <collection>: {
  ...
}]
```

- Using **items in a dictionary object**. This option works when your scenario is: "I want to create an instance for each item in an object." The items function converts the object to an array. Within the loop, you can use properties from the object to create values. For more information, see Dictionary object.

```
[for <item> in items(<object>): {
  ...
}]
```

- Using **integer index and items in an array**. This option works when your scenario is: "I want to create an instance for each element in an array, but I also need the current index to create another value." For more information, see Loop array and index.

```
[for (<item>, <index>) in <collection>: {
  ...
}]
```

- Adding a **conditional deployment**. This option works when your scenario is: "I want to create multiple

instances, but for each instance I want to deploy only when a condition is true." For more information, see Loop with condition.

```
[for <item> in <collection>: if(<condition>) {
  ...
}]
```

Loop limits

Using loops in Bicep has these limitations:

- Loop iterations can't be a negative number or exceed 800 iterations.
- Can't loop a resource with nested child resources. Change the child resources to top-level resources. See Iteration for a child resource.
- Can't loop on multiple levels of properties.

Integer index

For a simple example of using an index, create a **variable** that contains an array of strings.

```
param itemCount int = 5

var stringArray = [for i in range(0, itemCount): 'item${(i + 1)}']

output arrayResult array = stringArray
```

The output returns an array with the following values:

```
[
  "item1",
  "item2",
  "item3",
  "item4",
  "item5"
]
```

The next example creates the number of storage accounts specified in the `storageCount` parameter. It returns three properties for each storage account.

```
param location string = resourceGroup().location
param storageCount int = 2

resource storageAcct 'Microsoft.Storage/storageAccounts@2021-06-01' = [for i in range(0, storageCount): {
  name: '${i}storage${uniqueString(resourceGroup().id)}'
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'Storage'
}]

output storageInfo array = [for i in range(0, storageCount): {
  id: storageAcct[i].id
  blobEndpoint: storageAcct[i].properties.primaryEndpoints.blob
  status: storageAcct[i].properties.statusOfPrimary
}]
```

Notice the index `i` is used in creating the storage account resource name.

The next example deploys a module multiple times.

```
param location string = resourceGroup().location
param storageCount int = 2

var baseName = 'store${uniqueString(resourceGroup().id)}'

module stgModule './storageAccount.bicep' = [for i in range(0, storageCount): {
  name: '${i}deploy${baseName}'
  params: {
    storageName: '${i}${baseName}'
    location: location
  }
}]
```

Array elements

The following example creates one storage account for each name provided in the `storageNames` parameter.

```
param location string = resourceGroup().location
param storageNames array = [
  'contoso'
  'fabrikam'
  'coho'
]

resource storageAcct 'Microsoft.Storage/storageAccounts@2021-06-01' = [for name in storageNames: {
  name: '${name}${uniqueString(resourceGroup().id)}'
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'Storage'
}]
```

The next example iterates over an array to define a property. It creates two subnets within a virtual network.

```
param rgLocation string = resourceGroup().location

var subnets = [
  {
    name: 'api'
    subnetPrefix: '10.144.0.0/24'
  }
  {
    name: 'worker'
    subnetPrefix: '10.144.1.0/24'
  }
]

resource vnet 'Microsoft.Network/virtualNetworks@2020-07-01' = {
  name: 'vnet'
  location: rgLocation
  properties: {
    addressSpace: {
      addressPrefixes: [
        '10.144.0.0/20'
      ]
    }
    subnets: [for subnet in subnets: {
      name: subnet.name
      properties: {
        addressPrefix: subnet.subnetPrefix
      }
    }]
  }
}
```

Array and index

The following example uses both the array element and index value when defining the storage account.

```
param storageAccountNamePrefix string

var storageConfigurations = [
  {
    suffix: 'local'
    sku: 'Standard_LRS'
  }
  {
    suffix: 'geo'
    sku: 'Standard_GRS'
  }
]

resource storageAccountResources 'Microsoft.Storage/storageAccounts@2021-06-01' = [for (config, i) in
storageConfigurations: {
  name: '${storageAccountNamePrefix}${config.suffix}${i}'
  location: resourceGroup().location
  sku: {
    name: config.sku
  }
  kind: 'StorageV2'
}]
```

The next example uses both the elements of an array and an index to output information about the new resources.

```
param location string = resourceGroup().location
param orgNames array = [
  'Contoso'
  'Fabrikam'
  'Coho'
]

resource nsg 'Microsoft.Network/networkSecurityGroups@2020-06-01' = [for name in orgNames: {
  name: 'nsg-${name}'
  location: location
}]

output deployedNSGs array = [for (name, i) in orgNames: {
  orgName: name
  nsgName: nsg[i].name
  resourceId: nsg[i].id
}]
```

Dictionary object

To iterate over elements in a dictionary object, use the items function, which converts the object to an array. Use the `value` property to get properties on the objects.

```
param nsgValues object = {
  nsg1: {
    name: 'nsg-westus1'
    location: 'westus'
  }
  nsg2: {
    name: 'nsg-east1'
    location: 'eastus'
  }
}

resource nsg 'Microsoft.Network/networkSecurityGroups@2020-06-01' = [for nsg in items(nsgValues): {
  name: nsg.value.name
  location: nsg.value.location
}]
```

Loop with condition

For **resources and modules**, you can add an `if` expression with the loop syntax to conditionally deploy the collection.

The following example shows a loop combined with a condition statement. In this example, a single condition is applied to all instances of the module.

```
param location string = resourceGroup().location
param storageCount int = 2
param createNewStorage bool = true

var baseName = 'store${uniqueString(resourceGroup().id)}'

module stgModule './storageAccount.bicep' = [for i in range(0, storageCount): if(createNewStorage) {
  name: '${i}deploy${baseName}'
  params: {
    storageName: '${i}${baseName}'
    location: location
  }
}]
```

The next example shows how to apply a condition that is specific to the current element in the array.

```
resource parentResources 'Microsoft.Example/examples@2020-06-06' = [for parent in parents:
if(parent.enabled) {
  name: parent.name
  properties: {
    children: [for child in parent.children: {
      name: child.name
      setting: child.settingValue
    }]
  }
}]
```

Deploy in batches

By default, Azure resources are deployed in parallel. When you use a loop to create multiple instances of a resource type, those instances are all deployed at the same time. The order in which they're created isn't guaranteed. There's no limit to the number of resources deployed in parallel, other than the total limit of 800 resources in the Bicep file.

You might not want to update all instances of a resource type at the same time. 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. You can specify that a subset of the instances be batched together and deployed at the same time. The other instances wait for that batch to complete.

To serially deploy instances of a resource, add the batchSize decorator. Set its value to the number of instances to deploy concurrently. A dependency is created on earlier instances in the loop, so it doesn't start one batch until the previous batch completes.

```
param location string = resourceGroup().location

@batchSize(2)
resource storageAcct 'Microsoft.Storage/storageAccounts@2021-06-01' = [for i in range(0, 4): {
  name: '${i}storage${uniqueString(resourceGroup().id)}'
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'Storage'
}]
```

For sequential deployment, set the batch size to 1.

The `batchSize` decorator is in the sys namespace. If you need to differentiate this decorator from another item with the same name, preface the decorator with **sys**: `@sys.batchSize(2)`

Iteration for a child resource

You can't use a loop for a nested child resource. To create more than one instance of a child resource, change the child resource to a top-level resource.

For example, suppose you typically define a file service and file share as nested resources for a storage account.

```
resource stg 'Microsoft.Storage/storageAccounts@2021-06-01' = {
  name: 'examplestorage'
  location: resourceGroup().location
  kind: 'StorageV2'
  sku: {
    name: 'Standard_LRS'
  }
  resource service 'fileServices' = {
    name: 'default'
    resource share 'shares' = {
      name: 'exampleshare'
    }
  }
}
```

To create more than one file share, move it outside of the storage account. You define the relationship with the parent resource through the `parent` property.

The following example shows how to create a storage account, file service, and more than one file share:

```
resource stg 'Microsoft.Storage/storageAccounts@2021-06-01' = {
  name: 'examplestorage'
  location: resourceGroup().location
  kind: 'StorageV2'
  sku: {
    name: 'Standard_LRS'
  }
}

resource service 'Microsoft.Storage/storageAccounts/fileServices@2021-06-01' = {
  name: 'default'
  parent: stg
}

resource share 'Microsoft.Storage/storageAccounts/fileServices/shares@2021-06-01' = [for i in range(0, 3): {
  name: 'exampleshare${i}'
  parent: service
}]
```

Next steps

- To set dependencies on resources that are created in a loop, see Resource dependencies.

Conditional deployment in Bicep

9/21/2022 • 2 minutes to read • Edit Online

Sometimes you need to optionally deploy a resource or module in Bicep. Use the `if` keyword to specify whether the resource or module is deployed. The value for the condition resolves to true or false. When the value is true, the resource is created. When the value is false, the resource isn't created. The value can only be applied to the whole resource or module.

> **NOTE**
> Conditional deployment doesn't cascade to child resources. If you want to conditionally deploy a resource and its child resources, you must apply the same condition to each resource type.

Training resources

If you would rather learn about conditions through step-by-step guidance, see Build flexible Bicep templates by using conditions and loops.

Deploy condition

You can pass in a parameter value that indicates whether a resource is deployed. The following example conditionally deploys a DNS zone.

```
param deployZone bool

resource dnsZone 'Microsoft.Network/dnszones@2018-05-01' = if (deployZone) {
  name: 'myZone'
  location: 'global'
}
```

The next example conditionally deploys a module.

```
param deployZone bool

module dnsZone 'dnszones.bicep' = if (deployZone) {
  name: 'myZoneModule'
}
```

Conditions may be used with dependency declarations. For explicit dependencies, Azure Resource Manager automatically removes it from the required dependencies when the resource isn't deployed. For implicit dependencies, referencing a property of a conditional resource is allowed but may produce a deployment error.

New or existing resource

You can use conditional deployment to create a new resource or use an existing one. The following example shows how to either deploy a new storage account or use an existing storage account.

```
param storageAccountName string
param location string = resourceGroup().location

@allowed([
  'new'
  'existing'
])
param newOrExisting string = 'new'

resource sa 'Microsoft.Storage/storageAccounts@2019-06-01' = if (newOrExisting == 'new') {
  name: storageAccountName
  location: location
  sku: {
    name: 'Standard_LRS'
    tier: 'Standard'
  }
  kind: 'StorageV2'
  properties: {
    accessTier: 'Hot'
  }
}
```

When the parameter `newOrExisting` is set to **new**, the condition evaluates to true. The storage account is deployed. However, when `newOrExisting` is set to **existing**, the condition evaluates to false and the storage account isn't deployed.

Runtime functions

If you use a reference or list function with a resource that is conditionally deployed, the function is evaluated even if the resource isn't deployed. You get an error if the function refers to a resource that doesn't exist.

Use the conditional expression ?: operator to make sure the function is only evaluated for conditions when the resource is deployed. The following example template shows how to use this function with expressions that are only conditionally valid.

```
param vmName string
param location string
param logAnalytics string = ''

resource vmName_omsOnboarding 'Microsoft.Compute/virtualMachines/extensions@2017-03-30' = if
(!empty(logAnalytics)) {
  name: '${vmName}/omsOnboarding'
  location: location
  properties: {
    publisher: 'Microsoft.EnterpriseCloud.Monitoring'
    type: 'MicrosoftMonitoringAgent'
    typeHandlerVersion: '1.0'
    autoUpgradeMinorVersion: true
    settings: {
      workspaceId: ((!empty(logAnalytics)) ? reference(logAnalytics, '2015-11-01-preview').customerId :
null)
    }
    protectedSettings: {
      workspaceKey: ((!empty(logAnalytics)) ? listKeys(logAnalytics, '2015-11-01-preview').primarySharedKey
: null)
    }
  }
}

output mgmtStatus string = ((!empty(logAnalytics)) ? 'Enabled monitoring for VM!' : 'Nothing to enable')
```

Next steps

- Review the Learn module Build flexible Bicep templates by using conditions and loops.
- For recommendations about creating Bicep files, see Best practices for Bicep.
- To create multiple instances of a resource, see Iterative loops in Bicep.

Resource group deployments with Bicep files

9/21/2022 • 6 minutes to read • Edit Online

This article describes how to set scope with Bicep when deploying to a resource group.

Supported resources

Most resources can be deployed to a resource group. For a list of available resources, see ARM template reference.

Set scope

By default, a Bicep file is scoped to the resource group. If you want to explicitly set the scope, use:

```
targetScope = 'resourceGroup'
```

But, setting the target scope to resource group is unnecessary because that scope is used by default.

Deployment commands

To deploy to a resource group, use the resource group deployment commands.

- Azure CLI
- PowerShell

For Azure CLI, use az deployment group create. The following example deploys a template to create a resource group. The resource group you specify in the `--resource-group` parameter is the **target resource group**.

```
az deployment group create \
  --name demoRGDeployment \
  --resource-group ExampleGroup \
  --template-file main.bicep \
  --parameters storageAccountType=Standard_GRS
```

For more detailed information about deployment commands and options for deploying ARM templates, see:

- Deploy resources with ARM templates and Azure CLI
- Deploy resources with ARM templates and Azure PowerShell
- Deploy ARM templates from Cloud Shell

Deployment scopes

When deploying to a resource group, you can deploy resources to:

- the target resource group for the deployment operation
- other resource groups in the same subscription or other subscriptions
- any subscription in the tenant
- the tenant for the resource group

An extension resource can be scoped to a target that is different than the deployment target.

The user deploying the template must have access to the specified scope.

This section shows how to specify different scopes. You can combine these different scopes in a single template.

Scope to target resource group

To deploy resources to the target resource group, add those resources to the Bicep file.

```
// resource deployed to target resource group
resource exampleResource 'Microsoft.Storage/storageAccounts@2019-06-01' = {
  ...
}
```

For an example template, see Deploy to target resource group.

Scope to different resource group

To deploy resources to a resource group that isn't the target resource group, add a module. Use the resourceGroup function to set the `scope` property for that module.

If the resource group is in a different subscription, provide the subscription ID and the name of the resource group. If the resource group is in the same subscription as the current deployment, provide only the name of the resource group. If you don't specify a subscription in the resourceGroup function, the current subscription is used.

The following example shows a module that targets a resource group in a different subscription.

```
param otherResourceGroup string
param otherSubscriptionID string

// module deployed to different subscription and resource group
module exampleModule 'module.bicep' = {
  name: 'otherSubAndRG'
  scope: resourceGroup(otherSubscriptionID, otherResourceGroup)
}
```

The next example shows a module that targets a resource group in the same subscription.

```
param otherResourceGroup string

// module deployed to resource group in the same subscription
module exampleModule 'module.bicep' = {
  name: 'otherRG'
  scope: resourceGroup(otherResourceGroup)
}
```

For an example template, see Deploy to multiple resource groups.

Scope to subscription

To deploy resources to a subscription, add a module. Use the subscription function to set its `scope` property.

To deploy to the current subscription, use the subscription function without a parameter.

```
// module deployed at subscription level
module exampleModule 'module.bicep' = {
  name: 'deployToSub'
  scope: subscription()
}
```

To deploy to a different subscription, specify that subscription ID as a parameter in the subscription function.

```
param otherSubscriptionID string

// module deployed at subscription level but in a different subscription
module exampleModule 'module.bicep' = {
  name: 'deployToSub'
  scope: subscription(otherSubscriptionID)
}
```

For an example template, see Create resource group.

Scope to tenant

To create resources at the tenant, add a module. Use the tenant function to set its `scope` property.

The user deploying the template must have the required access to deploy at the tenant.

The following example includes a module that is deployed to the tenant.

```
// module deployed at tenant level
module exampleModule 'module.bicep' = {
  name: 'deployToTenant'
  scope: tenant()
}
```

Instead of using a module, you can set the scope to `tenant()` for some resource types. The following example deploys a management group at the tenant.

```
param mgName string = 'mg-${uniqueString(newGuid())}'

// ManagementGroup deployed at tenant
resource managementGroup 'Microsoft.Management/managementGroups@2020-05-01' = {
  scope: tenant()
  name: mgName
  properties: {}
}

output output string = mgName
```

For more information, see Management group.

Deploy to target resource group

To deploy resources in the target resource group, define those resources in the `resources` section of the template. The following template creates a storage account in the resource group that is specified in the deployment operation.

```
@minLength(3)
@maxLength(11)
param storagePrefix string

@allowed([
  'Standard_LRS'
  'Standard_GRS'
  'Standard_RAGRS'
  'Standard_ZRS'
  'Premium_LRS'
  'Premium_ZRS'
  'Standard_GZRS'
  'Standard_RAGZRS'
])
param storageSKU string = 'Standard_LRS'

param location string = resourceGroup().location

var uniqueStorageName = '${storagePrefix}${uniqueString(resourceGroup().id)}'

resource stg 'Microsoft.Storage/storageAccounts@2021-04-01' = {
  name: uniqueStorageName
  location: location
  sku: {
    name: storageSKU
  }
  kind: 'StorageV2'
  properties: {
    supportsHttpsTrafficOnly: true
  }
}

output storageEndpoint object = stg.properties.primaryEndpoints
```

Deploy to multiple resource groups

You can deploy to more than one resource group in a single Bicep file.

> **NOTE**
>
> You can deploy to **800 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 799 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 800 resource groups in nested or linked deployments.

The following example deploys two storage accounts. The first storage account is deployed to the resource group specified in the deployment operation. The second storage account is deployed to the resource group specified in the `secondResourceGroup` and `secondSubscriptionID` parameters:

```
@maxLength(11)
param storagePrefix string

param firstStorageLocation string = resourceGroup().location

param secondResourceGroup string
param secondSubscriptionID string = ''
param secondStorageLocation string

var firstStorageName = '${storagePrefix}${uniqueString(resourceGroup().id)}'
var secondStorageName = '${storagePrefix}${uniqueString(secondSubscriptionID, secondResourceGroup)}'

module firstStorageAcct 'storage.bicep' = {
  name: 'storageModule1'
  params: {
    storageLocation: firstStorageLocation
    storageName: firstStorageName
  }
}

module secondStorageAcct 'storage.bicep' = {
  name: 'storageModule2'
  scope: resourceGroup(secondSubscriptionID, secondResourceGroup)
  params: {
    storageLocation: secondStorageLocation
    storageName: secondStorageName
  }
}
```

Both modules use the same Bicep file named **storage.bicep**.

```
param storageLocation string
param storageName string

resource storageAcct 'Microsoft.Storage/storageAccounts@2019-06-01' = {
  name: storageName
  location: storageLocation
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'Storage'
  properties: {}
}
```

Create resource group

From a resource group deployment, you can switch to the level of a subscription and create a resource group. The following template deploys a storage account to the target resource group, and creates a new resource group in the specified subscription.

```
@maxLength(11)
param storagePrefix string

param firstStorageLocation string = resourceGroup().location

param secondResourceGroup string
param secondSubscriptionID string = ''
param secondLocation string

var firstStorageName = '${storagePrefix}${uniqueString(resourceGroup().id)}'

// resource deployed to target resource group
module firstStorageAcct 'storage2.bicep' = {
  name: 'storageModule1'
  params: {
    storageLocation: firstStorageLocation
    storageName: firstStorageName
  }
}

// module deployed to subscription
module newRG 'resourceGroup.bicep' = {
  name: 'newResourceGroup'
  scope: subscription(secondSubscriptionID)
  params: {
    resourceGroupName: secondResourceGroup
    resourceGroupLocation: secondLocation
  }
}
```

The preceding example uses the following Bicep file for the module that creates the new resource group.

```
targetScope='subscription'

param resourceGroupName string
param resourceGroupLocation string

resource newRG 'Microsoft.Resources/resourceGroups@2021-01-01' = {
  name: resourceGroupName
  location: resourceGroupLocation
}
```

Next steps

To learn about other scopes, see:

- Subscription deployments
- Management group deployments
- Tenant deployments

Subscription deployments with Bicep files

9/21/2022 • 7 minutes to read • Edit Online

This article describes how to set scope with Bicep when deploying to a subscription.

To simplify the management of resources, you can deploy resources at the level of your Azure subscription. For example, you can deploy policies and Azure role-based access control (Azure RBAC) to your subscription, which applies them across your subscription. You can also create resource groups within the subscription and deploy resources to resource groups in the subscription.

> **NOTE**
> You can deploy to 800 different resource groups in a subscription level deployment.

Training resources

If you would rather learn about deployment scopes through step-by-step guidance, see Deploy resources to subscriptions, management groups, and tenants by using Bicep.

Supported resources

Not all resource types can be deployed to the subscription level. This section lists which resource types are supported.

For Azure Blueprints, use:

- artifacts
- blueprints
- blueprintAssignments
- versions (Blueprints)

For Azure Policies, use:

- policyAssignments
- policyDefinitions
- policySetDefinitions
- remediations

For access control, use:

- accessReviewScheduleDefinitions
- accessReviewScheduleSettings
- roleAssignments
- roleAssignmentScheduleRequests
- roleDefinitions
- roleEligibilityScheduleRequests
- roleManagementPolicyAssignments

For nested templates that deploy to resource groups, use:

- deployments

For creating new resource groups, use:

- resourceGroups

For managing your subscription, use:

- budgets
- configurations - Advisor
- lineOfCredit
- locks
- profile - Change Analysis
- supportPlanTypes
- tags

For monitoring, use:

- diagnosticSettings
- logprofiles

For security, use:

- advancedThreatProtectionSettings
- alertsSuppressionRules
- assessmentMetadata
- assessments
- autoProvisioningSettings
- connectors
- deviceSecurityGroups
- ingestionSettings
- pricings
- securityContacts
- settings
- workspaceSettings

Other supported types include:

- scopeAssignments
- eventSubscriptions
- peerAsns

Set scope

To set the scope to subscription, use:

```
targetScope = 'subscription'
```

Deployment commands

To deploy to a subscription, use the subscription-level deployment commands.

- Azure CLI
- PowerShell

For Azure CLI, use az deployment sub create. The following example deploys a template to create a resource group:

```
az deployment sub create \
  --name demoSubDeployment \
  --location centralus \
  --template-file main.bicep \
  --parameters rgName=demoResourceGroup rgLocation=centralus
```

For more detailed information about deployment commands and options for deploying ARM templates, see:

- Deploy resources with ARM templates and Azure CLI
- Deploy resources with ARM templates and Azure PowerShell
- Deploy ARM templates from Cloud Shell

Deployment location and name

For subscription level deployments, you must provide a location for the deployment. The location of the deployment is separate from the location of the resources you deploy. The deployment location specifies where to store deployment data. Management group and tenant deployments also require a location. For resource group deployments, the location of the resource group is used to store the deployment data.

You can provide a name for the deployment, or use the default deployment name. The default name is the name of the template file. For example, deploying a template named *main.json* creates a default deployment name of **main**.

For each deployment name, the location is immutable. You can't create a deployment in one location when there's an existing deployment with the same name in a different location. For example, if you create a subscription deployment with the name **deployment1** in **centralus**, you can't later create another deployment with the name **deployment1** but a location of **westus**. If you get the error code `InvalidDeploymentLocation`, either use a different name or the same location as the previous deployment for that name.

Deployment scopes

When deploying to a subscription, you can deploy resources to:

- the target subscription from the operation
- any subscription in the tenant
- resource groups within the subscription or other subscriptions
- the tenant for the subscription

An extension resource can be scoped to a target that is different than the deployment target.

The user deploying the template must have access to the specified scope.

Scope to subscription

To deploy resources to the target subscription, add those resources with the `resource` keyword.

```
targetScope = 'subscription'

// resource group created in target subscription
resource exampleResource 'Microsoft.Resources/resourceGroups@2020-10-01' = {
  ...
}
```

For examples of deploying to the subscription, see Create resource groups and Assign policy definition.

To deploy resources to a subscription that is different than the subscription from the operation, add a module. Use the subscription function to set the `scope` property. Provide the `subscriptionId` property to the ID of the subscription you want to deploy to.

```
targetScope = 'subscription'

param otherSubscriptionID string

// module deployed at subscription level but in a different subscription
module exampleModule 'module.bicep' = {
  name: 'deployToDifferentSub'
  scope: subscription(otherSubscriptionID)
}
```

Scope to resource group

To deploy resources to a resource group within the subscription, add a module and set its `scope` property. If the resource group already exists, use the resourceGroup function to set the scope value. Provide the resource group name.

```
targetScope = 'subscription'

param resourceGroupName string

module exampleModule 'module.bicep' = {
  name: 'exampleModule'
  scope: resourceGroup(resourceGroupName)
}
```

If the resource group is created in the same Bicep file, use the symbolic name of the resource group to set the scope value. For an example of setting the scope to the symbolic name, see Create resource group and resources.

Scope to tenant

To create resources at the tenant, add a module. Use the tenant function to set its `scope` property.

The user deploying the template must have the required access to deploy at the tenant.

The following example includes a module that is deployed to the tenant.

```
targetScope = 'subscription'

// module deployed at tenant level
module exampleModule 'module.bicep' = {
  name: 'deployToTenant'
  scope: tenant()
}
```

Instead of using a module, you can set the scope to `tenant()` for some resource types. The following example deploys a management group at the tenant.

```
targetScope = 'subscription'

param mgName string = 'mg-${uniqueString(newGuid())}'

// management group created at tenant
resource managementGroup 'Microsoft.Management/managementGroups@2020-05-01' = {
  scope: tenant()
  name: mgName
  properties: {}
}

output output string = mgName
```

For more information, see Management group.

Resource groups

Create resource groups

To create a resource group, define a Microsoft.Resources/resourceGroups resource with a name and location for the resource group.

The following example creates an empty resource group.

```
targetScope='subscription'

param resourceGroupName string
param resourceGroupLocation string

resource newRG 'Microsoft.Resources/resourceGroups@2021-01-01' = {
  name: resourceGroupName
  location: resourceGroupLocation
}
```

Create resource group and resources

To create the resource group and deploy resources to it, add a module. The module includes the resources to deploy to the resource group. Set the scope for the module to the symbolic name for the resource group you create. You can deploy to up to 800 resource groups.

The following example creates a resource group, and deploys a storage account to the resource group. Notice that the `scope` property for the module is set to `newRG`, which is the symbolic name for the resource group that is being created.

```
targetScope='subscription'

param resourceGroupName string
param resourceGroupLocation string
param storageName string
param storageLocation string

resource newRG 'Microsoft.Resources/resourceGroups@2021-01-01' = {
  name: resourceGroupName
  location: resourceGroupLocation
}

module storageAcct 'storage.bicep' = {
  name: 'storageModule'
  scope: newRG
  params: {
    storageLocation: storageLocation
    storageName: storageName
  }
}
```

The module uses a Bicep file named **storage.bicep** with the following contents:

```
param storageLocation string
param storageName string

resource storageAcct 'Microsoft.Storage/storageAccounts@2019-06-01' = {
  name: storageName
  location: storageLocation
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'Storage'
  properties: {}
}
```

Azure Policy

Assign policy definition

The following example assigns an existing policy definition to the subscription. If the policy definition takes parameters, provide them as an object. If the policy definition doesn't take parameters, use the default empty object.

```
targetScope = 'subscription'

param policyDefinitionID string
param policyName string
param policyParameters object = {}

resource policyAssign 'Microsoft.Authorization/policyAssignments@2020-09-01' = {
  name: policyName
  properties: {
    policyDefinitionId: policyDefinitionID
    parameters: policyParameters
  }
}
```

Create and assign policy definitions

You can define and assign a policy definition in the same Bicep file.

```
targetScope = 'subscription'

resource locationPolicy 'Microsoft.Authorization/policyDefinitions@2020-09-01' = {
  name: 'locationpolicy'
  properties: {
    policyType: 'Custom'
    parameters: {}
    policyRule: {
      if: {
        field: 'location'
        equals: 'northeurope'
      }
      then: {
        effect: 'deny'
      }
    }
  }
}

resource locationRestrict 'Microsoft.Authorization/policyAssignments@2020-09-01' = {
  name: 'allowedLocation'
  properties: {
    policyDefinitionId: locationPolicy.id
  }
}
```

Access control

To learn about assigning roles, see Add Azure role assignments using Azure Resource Manager templates.

The following example creates a resource group, applies a lock to it, and assigns a role to a principal.

```
targetScope = 'subscription'

@description('Name of the resourceGroup to create')
param resourceGroupName string

@description('Location for the resourceGroup')
param resourceGroupLocation string

@description('principalId of the user that will be given contributor access to the resourceGroup')
param principalId string

@description('roleDefinition to apply to the resourceGroup - default is contributor')
param roleDefinitionId string = 'b24988ac-6180-42a0-ab88-20f7382dd24c'

@description('Unique name for the roleAssignment in the format of a guid')
param roleAssignmentName string = guid(principalId, roleDefinitionId, resourceGroupName)

var roleID =
'/subscriptions/${subscription().subscriptionId}/providers/Microsoft.Authorization/roleDefinitions/${roleDef
initionId}'

resource newResourceGroup 'Microsoft.Resources/resourceGroups@2019-10-01' = {
  name: resourceGroupName
  location: resourceGroupLocation
  properties: {}
}

module applyLock 'lock.bicep' = {
  name: 'applyLock'
  scope: newResourceGroup
}

module assignRole 'role.bicep' = {
  name: 'assignRBACRole'
  scope: newResourceGroup
  params: {
    principalId: principalId
    roleNameGuid: roleAssignmentName
    roleDefinitionId: roleID
  }
}
```

The following example shows the module to apply the lock:

```
resource createRgLock 'Microsoft.Authorization/locks@2016-09-01' = {
  name: 'rgLock'
  properties: {
    level: 'CanNotDelete'
    notes: 'Resource group should not be deleted.'
  }
}
```

The next example shows the module to assign the role:

```
@description('The principal to assign the role to')
param principalId string

@description('A GUID used to identify the role assignment')
param roleNameGuid string = newGuid()

param roleDefinitionId string

resource roleNameGuid_resource 'Microsoft.Authorization/roleAssignments@2020-04-01-preview' = {
  name: roleNameGuid
  properties: {
    roleDefinitionId: roleDefinitionId
    principalId: principalId
  }
}
```

Next steps

To learn about other scopes, see:

- Resource group deployments
- Management group deployments
- Tenant deployments

Management group deployments with Bicep files

9/21/2022 • 6 minutes to read • Edit Online

This article describes how to set scope with Bicep when deploying to a management group.

As your organization matures, you can deploy a Bicep file to create resources at the management group level. For example, you may need to define and assign policies or Azure role-based access control (Azure RBAC) for a management group. With management group level templates, you can declaratively apply policies and assign roles at the management group level.

Training resources

If you would rather learn about deployment scopes through step-by-step guidance, see Deploy resources to subscriptions, management groups, and tenants by using Bicep.

Supported resources

Not all resource types can be deployed to the management group level. This section lists which resource types are supported.

For Azure Blueprints, use:

- artifacts
- blueprints
- blueprintAssignments
- versions

For Azure Policy, use:

- policyAssignments
- policyDefinitions
- policySetDefinitions
- remediations

For access control, use:

- privateLinkAssociations
- roleAssignments
- roleAssignmentScheduleRequests
- roleDefinitions
- roleEligibilityScheduleRequests
- roleManagementPolicyAssignments

For nested templates that deploy to subscriptions or resource groups, use:

- deployments

For managing your resources, use:

- diagnosticSettings
- tags

Management groups are tenant-level resources. However, you can create management groups in a management

group deployment by setting the scope of the new management group to the tenant. See Management group.

Set scope

To set the scope to management group, use:

```
targetScope = 'managementGroup'
```

Deployment commands

To deploy to a management group, use the management group deployment commands.

- Azure CLI
- PowerShell

For Azure CLI, use az deployment mg create:

```
az deployment mg create \
  --name demoMGDeployment \
  --location WestUS \
  --management-group-id myMG \
  --template-uri "https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/management-level-deployment/azuredeploy.json"
```

For more detailed information about deployment commands and options for deploying ARM templates, see:

- Deploy resources with ARM templates and Azure CLI
- Deploy resources with ARM templates and Azure PowerShell
- Deploy ARM templates from Cloud Shell

Deployment location and name

For management group level deployments, you must provide a location for the deployment. The location of the deployment is separate from the location of the resources you deploy. The deployment location specifies where to store deployment data. Subscription and tenant deployments also require a location. For resource group deployments, the location of the resource group is used to store the deployment data.

You can provide a name for the deployment, or use the default deployment name. The default name is the name of the template file. For example, deploying a template named *main.bicep* creates a default deployment name of **main**.

For each deployment name, the location is immutable. You can't create a deployment in one location when there's an existing deployment with the same name in a different location. For example, if you create a management group deployment with the name **deployment1** in **centralus**, you can't later create another deployment with the name **deployment1** but a location of **westus**. If you get the error code `InvalidDeploymentLocation`, either use a different name or the same location as the previous deployment for that name.

Deployment scopes

When deploying to a management group, you can deploy resources to:

- the target management group from the operation
- another management group in the tenant

- subscriptions in the management group
- resource groups in the management group
- the tenant for the resource group

An extension resource can be scoped to a target that is different than the deployment target.

The user deploying the template must have access to the specified scope.

Scope to management group

To deploy resources to the target management group, add those resources with the `resource` keyword.

```
targetScope = 'managementGroup'

// policy definition created in the management group
resource policyDefinition 'Microsoft.Authorization/policyDefinitions@2019-09-01' = {
  ...
}
```

To target another management group, add a module. Use the managementGroup function to set the `scope` property. Provide the management group name.

```
targetScope = 'managementGroup'

param otherManagementGroupName string

// module deployed at management group level but in a different management group
module exampleModule 'module.bicep' = {
  name: 'deployToDifferentMG'
  scope: managementGroup(otherManagementGroupName)
}
```

Scope to subscription

You can also target subscriptions within a management group. The user deploying the template must have access to the specified scope.

To target a subscription within the management group, add a module. Use the subscription function to set the `scope` property. Provide the subscription ID.

```
targetScope = 'managementGroup'

param subscriptionID string

// module deployed to subscription in the management group
module exampleModule 'module.bicep' = {
  name: 'deployToSub'
  scope: subscription(subscriptionID)
}
```

Scope to resource group

You can also target resource groups within the management group. The user deploying the template must have access to the specified scope.

To target a resource group within the management group, add a module. Use the resourceGroup function to set the `scope` property. Provide the subscription ID and resource group name.

```
targetScope = 'managementGroup'

param subscriptionID string
param resourceGroupName string

// module deployed to resource group in the management group
module exampleModule 'module.bicep' = {
  name: 'deployToRG'
  scope: resourceGroup(subscriptionID, resourceGroupName)
}
```

Scope to tenant

To create resources at the tenant, add a module. Use the tenant function to set its `scope` property. The user deploying the template must have the required access to deploy at the tenant.

```
targetScope = 'managementGroup'

// module deployed at tenant level
module exampleModule 'module.bicep' = {
  name: 'deployToTenant'
  scope: tenant()
}
```

Or, you can set the scope to `/` for some resource types, like management groups. Creating a new management group is described in the next section.

Management group

To create a management group in a management group deployment, you must set the scope to the tenant.

The following example creates a new management group in the root management group.

```
targetScope = 'managementGroup'

param mgName string = 'mg-${uniqueString(newGuid())}'

resource newMG 'Microsoft.Management/managementGroups@2020-05-01' = {
  scope: tenant()
  name: mgName
  properties: {}
}

output newManagementGroup string = mgName
```

The next example creates a new management group in the management group targeted for the deployment. It uses the management group function.

```
targetScope = 'managementGroup'

param mgName string = 'mg-${uniqueString(newGuid())}'

resource newMG 'Microsoft.Management/managementGroups@2020-05-01' = {
  scope: tenant()
  name: mgName
  properties: {
    details: {
      parent: {
        id: managementGroup().id
      }
    }
  }
}

output newManagementGroup string = mgName
```

Subscriptions

To use an ARM template to create a new Azure subscription in a management group, see:

- Programmatically create Azure Enterprise Agreement subscriptions
- Programmatically create Azure subscriptions for a Microsoft Customer Agreement
- Programmatically create Azure subscriptions for a Microsoft Partner Agreement

To deploy a template that moves an existing Azure subscription to a new management group, see Move subscriptions in ARM template

Azure Policy

Custom policy definitions that are deployed to the management group are extensions of the management group. To get the ID of a custom policy definition, use the extensionResourceId() function. Built-in policy definitions are tenant level resources. To get the ID of a built-in policy definition, use the tenantResourceId() function.

The following example shows how to define a policy at the management group level, and assign it.

```
targetScope = 'managementGroup'

@description('An array of the allowed locations, all other locations will be denied by the created policy.')
param allowedLocations array = [
  'australiaeast'
  'australiasoutheast'
  'australiacentral'
]

resource policyDefinition 'Microsoft.Authorization/policyDefinitions@2020-09-01' = {
  name: 'locationRestriction'
  properties: {
    policyType: 'Custom'
    mode: 'All'
    parameters: {}
    policyRule: {
      if: {
        not: {
          field: 'location'
          in: allowedLocations
        }
      }
      then: {
        effect: 'deny'
      }
    }
  }
}

resource policyAssignment 'Microsoft.Authorization/policyAssignments@2020-09-01' = {
  name: 'locationAssignment'
  properties: {
    policyDefinitionId: policyDefinition.id
  }
}
```

Next steps

To learn about other scopes, see:

- Resource group deployments
- Subscription deployments
- Tenant deployments

Tenant deployments with Bicep file

9/21/2022 • 5 minutes to read • Edit Online

As your organization matures, you may need to define and assign policies or Azure role-based access control (Azure RBAC) across your Azure AD tenant. With tenant level templates, you can declaratively apply policies and assign roles at a global level.

Training resources

If you would rather learn about deployment scopes through step-by-step guidance, see Deploy resources to subscriptions, management groups, and tenants by using Bicep.

Supported resources

Not all resource types can be deployed to the tenant level. This section lists which resource types are supported.

For Azure role-based access control (Azure RBAC), use:

- roleAssignments

For nested templates that deploy to management groups, subscriptions, or resource groups, use:

- deployments

For creating management groups, use:

- managementGroups

For creating subscriptions, use:

- aliases

For managing costs, use:

- billingProfiles
- billingRoleAssignments
- instructions
- invoiceSections
- policies

For configuring the portal, use:

- tenantConfigurations

Built-in policy definitions are tenant-level resources, but you can't deploy custom policy definitions at the tenant. For an example of assigning a built-in policy definition to a resource, see tenantResourceId example.

Set scope

To set the scope to tenant, use:

```
targetScope = 'tenant'
```

Required access

The principal deploying the template must have permissions to create resources at the tenant scope. The principal must have permission to execute the deployment actions (`Microsoft.Resources/deployments/*`) and to create the resources defined in the template. For example, to create a management group, the principal must have Contributor permission at the tenant scope. To create role assignments, the principal must have Owner permission.

The Global Administrator for the Azure Active Directory doesn't automatically have permission to assign roles. To enable template deployments at the tenant scope, the Global Administrator must do the following steps:

1. Elevate account access so the Global Administrator can assign roles. For more information, see Elevate access to manage all Azure subscriptions and management groups.
2. Assign Owner or Contributor to the principal that needs to deploy the templates.

```
New-AzRoleAssignment -SignInName "[userId]" -Scope "/" -RoleDefinitionName "Owner"
```

```
az role assignment create --assignee "[userId]" --scope "/" --role "Owner"
```

The principal now has the required permissions to deploy the template.

Deployment commands

The commands for tenant deployments are different than the commands for resource group deployments.

- Azure CLI
- PowerShell

For Azure CLI, use az deployment tenant create:

```
az deployment tenant create \
  --name demoTenantDeployment \
  --location WestUS \
  --template-file main.bicep
```

For more detailed information about deployment commands and options for deploying ARM templates, see:

- Deploy resources with ARM templates and Azure CLI
- Deploy resources with ARM templates and Azure PowerShell
- Deploy ARM templates from Cloud Shell

Deployment location and name

For tenant level deployments, you must provide a location for the deployment. The location of the deployment is separate from the location of the resources you deploy. The deployment location specifies where to store deployment data. Subscription and management group deployments also require a location. For resource group deployments, the location of the resource group is used to store the deployment data.

You can provide a name for the deployment, or use the default deployment name. The default name is the name of the template file. For example, deploying a file named *main.bicep* creates a default deployment name of **main**.

For each deployment name, the location is immutable. You can't create a deployment in one location when

there's an existing deployment with the same name in a different location. For example, if you create a tenant deployment with the name **deployment1** in **centralus**, you can't later create another deployment with the name **deployment1** but a location of **westus**. If you get the error code `InvalidDeploymentLocation`, either use a different name or the same location as the previous deployment for that name.

Deployment scopes

When deploying to a tenant, you can deploy resources to:

- the tenant
- management groups within the tenant
- subscriptions
- resource groups

An extension resource can be scoped to a target that is different than the deployment target.

The user deploying the template must have access to the specified scope.

This section shows how to specify different scopes. You can combine these different scopes in a single template.

Scope to tenant

Resources defined within the Bicep file are applied to the tenant.

```
targetScope = 'tenant'

// create resource at tenant
resource mgName_resource 'Microsoft.Management/managementGroups@2020-02-01' = {
  ...
}
```

Scope to management group

To target a management group within the tenant, add a module. Use the managementGroup function to set its `scope` property. Provide the management group name.

```
targetScope = 'tenant'

param managementGroupName string

// create resources at management group level
module  'module.bicep' = {
  name: 'deployToMG'
  scope: managementGroup(managementGroupName)
}
```

Scope to subscription

To target a subscription within the tenant, add a module. Use the subscription function to set its `scope` property. Provide the subscription ID.

```
targetScope = 'tenant'

param subscriptionID string

// create resources at subscription level
module  'module.bicep' = {
  name: 'deployToSub'
  scope: subscription(subscriptionID)
}
```

Scope to resource group

To target a resource group within the tenant, add a module. Use the resourceGroup function to set its `scope` property. Provide the subscription ID and resource group name.

```
targetScope = 'tenant'

param resourceGroupName string
param subscriptionID string

// create resources at resource group level
module  'module.bicep' = {
  name: 'deployToRG'
  scope: resourceGroup(subscriptionID, resourceGroupName)
}
```

Create management group

The following template creates a management group.

```
targetScope = 'tenant'
param mgName string = 'mg-${uniqueString(newGuid())}'

resource mgName_resource 'Microsoft.Management/managementGroups@2020-02-01' = {
  name: mgName
  properties: {}
}
```

If your account doesn't have permission to deploy to the tenant, you can still create management groups by deploying to another scope. For more information, see Management group.

Assign role

The following template assigns a role at the tenant scope.

```
targetScope = 'tenant'

@description('principalId of the user that will be given contributor access to the resourceGroup')
param principalId string

@description('roleDefinition for the assignment - default is owner')
param roleDefinitionId string = '8e3af657-a8ff-443c-a75c-2fe8c4bcb635'

var roleAssignmentName = guid(principalId, roleDefinitionId)

resource roleAssignment 'Microsoft.Authorization/roleAssignments@2020-03-01-preview' = {
  name: roleAssignmentName
  properties: {
    roleDefinitionId: tenantResourceId('Microsoft.Authorization/roleDefinitions', roleDefinitionId)
    principalId: principalId
  }
}
```

Next steps

To learn about other scopes, see:

- Resource group deployments
- Subscription deployments
- Management group deployments

Bicep functions

9/21/2022 • 3 minutes to read • Edit Online

This article describes all the functions you can use in a Bicep file. For a description of the sections in a Bicep file, see Understand the structure and syntax of Bicep files.

Most functions work the same when deployed to a resource group, subscription, management group, or tenant. A few functions can't be used in all scopes. They're noted in the lists below.

Namespaces for functions

All Bicep functions are contained within two namespaces - `az` and `sys`. Typically, you don't need to specify the namespace when you use the function. You specify the namespace only when the function name is the same as another item you've defined in the Bicep file. For example, if you create a parameter named `range`, you need to differentiate the `range` function by adding the `sys` namespace.

```
// Parameter contains the same name as a function
param range int

// Must use sys namespace to call the function.
// The second use of range refers to the parameter.
output result array = sys.range(1, range)
```

The `az` namespace contains functions that are specific to an Azure deployment. The `sys` namespace contains functions that are used to construct values. The `sys` namespace also includes decorators for parameters and resource loops. The namespaces are noted in this article.

Any function

The any function is available in Bicep to help resolve issues around data type warnings. This function is in the `sys` namespace.

Array functions

The following functions are available for working with arrays. All of these functions are in the `sys` namespace.

- array
- concat
- contains
- empty
- indexOf
- first
- flatten
- intersection
- last
- lastIndexOf
- length
- min
- max

- range
- skip
- take
- union

Date functions

The following functions are available for working with dates. All of these functions are in the `sys` namespace.

- dateTimeAdd
- dateTimeFromEpoch
- dateTimeToEpoch
- utcNow

Deployment value functions

The following functions are available for getting values related to the deployment. All of these functions are in the `az` namespace.

- deployment
- environment

File functions

The following functions are available for loading the content from external files into your Bicep file. All of these functions are in the `sys` namespace.

- loadFileAsBase64
- loadJsonContent
- loadTextContent

Lambda functions

The following functions are available for working with lambda expressions. All of these functions are in the `sys` namespace.

- filter
- map
- reduce
- sort

Logical functions

The following function is available for working with logical conditions. This function is in the `sys` namespace.

- bool

Numeric functions

The following functions are available for working with integers. All of these functions are in the `sys` namespace.

- int
- min
- max

Object functions

The following functions are available for working with objects. All of these functions are in the `sys` namespace.

- contains
- empty
- intersection
- items
- json
- length
- union

Resource functions

The following functions are available for getting resource values. Most of these functions are in the `az` namespace. The list functions and the getSecret function are called directly on the resource type, so they don't have a namespace qualifier.

- extensionResourceId
- getSecret
- listAccountSas
- listKeys
- listSecrets
- list*
- pickZones
- providers (deprecated)
- reference
- resourceId - can be used at any scope, but the valid parameters change depending on the scope.
- subscriptionResourceId
- tenantResourceId

Scope functions

The following functions are available for getting scope values. All of these functions are in the `az` namespace.

- managementGroup
- resourceGroup - can only be used in deployments to a resource group.
- subscription - can only be used in deployments to a resource group or subscription.
- tenant

String functions

Bicep provides the following functions for working with strings. All of these functions are in the `sys` namespace.

- base64
- base64ToJson
- base64ToString
- concat
- contains
- dataUri

- dataUriToString
- empty
- endsWith
- first
- format
- guid
- indexOf
- join
- last
- lastIndexOf
- length
- newGuid
- padLeft
- replace
- skip
- split
- startsWith
- string
- substring
- take
- toLower
- toUpper
- trim
- uniqueString
- uri
- uriComponent
- uriComponentToString

Next steps

- For a description of the sections in a Bicep file, see Understand the structure and syntax of Bicep files.
- To iterate a specified number of times when creating a type of resource, see Iterative loops in Bicep.
- To see how to deploy the Bicep file you've created, see Deploy resources with Bicep and Azure PowerShell.

Any function for Bicep

9/21/2022 • 2 minutes to read • Edit Online

Bicep supports a function called `any()` to resolve type errors in the Bicep type system. You use this function when the format of the value you provide doesn't match what the type system expects. For example, if the property requires a number but you need to provide it as a string, like `'0.5'`. Use the `any()` function to suppress the error reported by the type system.

This function doesn't exist in the Azure Resource Manager template runtime. It's only used by Bicep and isn't emitted in the JSON for the built template.

> **NOTE**
> To help resolve type errors, let us know when missing or incorrect types required you to use the `any()` function. Add your details to the missing type validation/inaccuracies GitHub issue.

any

`any(value)`

Returns a value that is compatible with any data type.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
value	Yes	all types	The value to convert to a compatible type.

Return value

The value in a form that is compatible with any data type.

Examples

The following example shows how to use the `any()` function to provide numeric values as strings.

```
resource wpAci 'microsoft.containerInstance/containerGroups@2019-12-01' = {
  name: 'wordpress-containerinstance'
  location: location
  properties: {
    containers: [
      {
        name: 'wordpress'
        properties: {
          ...
          resources: {
            requests: {
              cpu: any('0.5')
              memoryInGB: any('0.7')
            }
          }
        }
      }
    ]
  }
}
```

The function works on any assigned value in Bicep. The following example uses `any()` with a ternary expression as an argument.

```
publicIPAddress: any((pipId == '') ? null : {
  id: pipId
})
```

Next steps

For more complex uses of the `any()` function, see the following examples:

- Child resources that require a specific names
- A resource property not defined in the resource's type, even though it exists

Array functions for Bicep

9/21/2022 • 13 minutes to read • Edit Online

This article describes the Bicep functions for working with arrays. The lambda functions for working with arrays can be found here.

array

`array(convertToArray)`

Converts the value to an array.

Namespace: sys.

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 shows how to use the array function with different types.

```
param intToConvert int = 1
param stringToConvert string = 'efgh'
param objectToConvert object = {
  'a': 'b'
  'c': 'd'
}

output intOutput array = array(intToConvert)
output stringOutput array = array(stringToConvert)
output objectOutput array = array(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"}]

concat

`concat(arg1, arg2, arg3, ...)`

Combines multiple arrays and returns the concatenated array.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	array	The first array for concatenation.
more arguments	No	array	More arrays in sequential order for concatenation.

This function takes any number of arrays and combines them.

Return value

An array of concatenated values.

Example

The following example shows how to combine two arrays.

```
param firstArray array = [
  '1-1'
  '1-2'
  '1-3'
]
param secondArray array = [
  '2-1'
  '2-2'
  '2-3'
]

output return array = concat(firstArray, secondArray)
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
return	Array	["1-1", "1-2", "1-3", "2-1", "2-2", "2-3"]

contains

`contains(container, itemToFind)`

Checks whether an array contains a value, an object contains a key, or a string contains a substring. The string comparison is case-sensitive. However, when testing if an object contains a key, the comparison is case-insensitive.

Namespace: sys.

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 shows how to use contains with different types:

```
param stringToTest string = 'OneTwoThree'
param objectToTest object = {
  'one': 'a'
  'two': 'b'
  'three': 'c'
}
param arrayToTest array = [
  'one'
  'two'
  'three'
]

output stringTrue bool = contains(stringToTest, 'e')
output stringFalse bool = contains(stringToTest, 'z')
output objectTrue bool = contains(objectToTest, 'one')
output objectFalse bool = contains(objectToTest, 'a')
output arrayTrue bool = contains(arrayToTest, 'three')
output arrayFalse bool = contains(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

empty

`empty(itemToTest)`

Determines if an array, object, or string is empty.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
itemToTest	Yes	array, object, or string	The value to check if it's empty.

Return value

Returns **True** if the value is empty; otherwise, **False**.

Example

The following example checks whether an array, object, and string are empty.

```
param testArray array = []
param testObject object = {}
param testString string = ''

output arrayEmpty bool = empty(testArray)
output objectEmpty bool = empty(testObject)
output stringEmpty bool = empty(testString)
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayEmpty	Bool	True
objectEmpty	Bool	True
stringEmpty	Bool	True

first

`first(arg1)`

Returns the first element of the array, or first character of the string.

Namespace: sys.

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 shows how to use the first function with an array and string.

```
param arrayToTest array = [
  'one'
  'two'
  'three'
]

output arrayOutput string = first(arrayToTest)
output stringOutput string = first('One Two Three')
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayOutput	String	one
stringOutput	String	O

flatten

`flatten(arrayToFlatten)`

Takes an array of arrays, and returns an array of sub-array elements, in the original order. Sub-arrays are only flattened once, not recursively.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arrayToFlattern	Yes	array	The array of sub-arrays to flatten.

Return value

Array

Example

The following example shows how to use the flatten function.

```
param arrayToTest array = [
  ['one', 'two']
  ['three']
  ['four', 'five']
]
output arrayOutput array = flatten(arrayToTest)
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayOutput	array	['one', 'two', 'three', 'four', 'five']

indexOf

`indexOf(arrayToSearch, itemToFind)`

Returns an integer for the index of the first occurrence of an item in an array. The comparison is **case-sensitive** for strings.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arrayToSearch	Yes	array	The array to use for finding the index of the searched item.
itemToFind	Yes	int, string, array, or object	The item to find in the array.

Return value

An integer representing the first index of the item in the array. The index is zero-based. If the item isn't found, -1 is returned.

Examples

The following example shows how to use the indexOf and lastIndexOf functions:

```
var names = [
  'one'
  'two'
  'three'
]

var numbers = [
  4
  5
  6
]

var collection = [
  names
  numbers
]

var duplicates = [
  1
  2
  3
  1
]

output index1 int = lastIndexOf(names, 'two')
output index2 int = indexOf(names, 'one')
output notFoundIndex1 int = lastIndexOf(names, 'Three')

output index3 int = lastIndexOf(numbers, 4)
output index4 int = indexOf(numbers, 6)
output notFoundIndex2 int = lastIndexOf(numbers, '5')

output index5 int = indexOf(collection, numbers)

output index6 int = indexOf(duplicates, 1)
output index7 int = lastIndexOf(duplicates, 1)
```

The output from the preceding example is:

NAME	TYPE	VALUE
index1	int	1
index2	int	0

NAME	TYPE	VALUE
index3	int	0
index4	int	2
index5	int	1
index6	int	0
index7	int	3
notFoundIndex1	int	-1
notFoundIndex2	int	-1

intersection

```
intersection(arg1, arg2, arg3, ...)
```

Returns a single array or object with the common elements from the parameters.

Namespace: sys.

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.
more arguments	No	array or object	More values to use for finding common elements.

Return value

An array or object with the common elements. The order of the elements is determined by the first array parameter.

Example

The following example shows how to use intersection with arrays and objects:

```
param firstObject object = {
  'one': 'a'
  'two': 'b'
  'three': 'c'
}

param secondObject object = {
  'one': 'a'
  'two': 'z'
  'three': 'c'
}

param firstArray array = [
  'one'
  'two'
  'three'
]

param secondArray array = [
  'two'
  'three'
]

output objectOutput object = intersection(firstObject, secondObject)
output arrayOutput array = intersection(firstArray, 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"]

The first array parameter determines the order of the intersected elements. The following example shows how the order of the returned elements is based on which array is first.

```
var array1 = [
  1
  2
  3
  4
]

var array2 = [
  3
  2
  1
]

var array3 = [
  4
  1
  3
  2
]

output commonUp array = intersection(array1, array2, array3)
output commonDown array = intersection(array2, array3, array1)
```

The output from the preceding example is:

NAME	TYPE	VALUE
commonUp	array	[1, 2, 3]
commonDown	array	[3, 2, 1]

last

`last(arg1)`

Returns the last element of the array, or last character of the string.

Namespace: sys.

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 shows how to use the last function with an array and string.

```
param arrayToTest array = [
  'one'
  'two'
  'three'
]

output arrayOutput string = last(arrayToTest)
output stringOutput string = last('One Two three')
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayOutput	String	three
stringOutput	String	e

lastIndexOf

`lastIndexOf(arrayToSearch, itemToFind)`

Returns an integer for the index of the last occurrence of an item in an array. The comparison is **case-sensitive** for strings.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arrayToSearch	Yes	array	The array to use for finding the index of the searched item.
itemToFind	Yes	int, string, array, or object	The item to find in the array.

Return value

An integer representing the last index of the item in the array. The index is zero-based. If the item isn't found, -1 is returned.

Examples

The following example shows how to use the indexOf and lastIndexOf functions:

```
var names = [
  'one'
  'two'
  'three'
]

var numbers = [
  4
  5
  6
]

var collection = [
  names
  numbers
]

var duplicates = [
  1
  2
  3
  1
]

output index1 int = lastIndexOf(names, 'two')
output index2 int = indexOf(names, 'one')
output notFoundIndex1 int = lastIndexOf(names, 'Three')

output index3 int = lastIndexOf(numbers, 4)
output index4 int = indexOf(numbers, 6)
output notFoundIndex2 int = lastIndexOf(numbers, '5')

output index5 int = indexOf(collection, numbers)

output index6 int = indexOf(duplicates, 1)
output index7 int = lastIndexOf(duplicates, 1)
```

The output from the preceding example is:

NAME	TYPE	VALUE
index1	int	1
index2	int	0

NAME	TYPE	VALUE
index3	int	0
index4	int	2
index5	int	1
index6	int	0
index7	int	3
notFoundIndex1	int	-1
notFoundIndex2	int	-1

length

`length(arg1)`

Returns the number of elements in an array, characters in a string, or root-level properties in an object.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	array, string, or object	The array to use for getting the number of elements, the string to use for getting the number of characters, or the object to use for getting the number of root-level properties.

Return value

An int.

Example

The following example shows how to use length with an array and string:

```
param arrayToTest array = [
  'one'
  'two'
  'three'
]
param stringToTest string = 'One Two Three'
param objectToTest object = {
  'propA': 'one'
  'propB': 'two'
  'propC': 'three'
  'propD': {
    'propD-1': 'sub'
    'propD-2': 'sub'
  }
}

output arrayLength int = length(arrayToTest)
output stringLength int = length(stringToTest)
output objectLength int = length(objectToTest)
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayLength	Int	3
stringLength	Int	13
objectLength	Int	4

max

`max(arg1)`

Returns the maximum value from an array of integers or a comma-separated list of integers.

Namespace: sys.

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 shows how to use max with an array and a list of integers:

```
param arrayToTest array = [
  0
  3
  2
  5
  4
]

output arrayOutput int = max(arrayToTest)
output intOutput int = 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

min

`min(arg1)`

Returns the minimum value from an array of integers or a comma-separated list of integers.

Namespace: sys.

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 shows how to use min with an array and a list of integers:

```
param arrayToTest array = [
  0
  3
  2
  5
  4
]

output arrayOutput int = min(arrayToTest)
output intOutput int = min(0,3,2,5,4)
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayOutput	Int	0

NAME	TYPE	VALUE
intOutput	Int	0

range

`range(startIndex, count)`

Creates an array of integers from a starting integer and containing the number of items.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
startIndex	Yes	int	The first integer in the array. The sum of startIndex and count must be no greater than 2147483647.
count	Yes	int	The number of integers in the array. Must be non-negative integer up to 10000.

Return value

An array of integers.

Example

The following example shows how to use the range function:

```
param startingInt int = 5
param numberOfElements int = 3

output rangeOutput array = range(startingInt, numberOfElements)
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
rangeOutput	Array	[5, 6, 7]

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.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
originalValue	Yes	array or string	The array or string to use for skipping.
numberToSkip	Yes	int	The number of elements or characters to skip. If this value is 0 or less, all the elements or characters in the value are returned. If it's larger than the length of the array or string, an empty array or string is returned.

Return value

An array or string.

Example

The following example skips the specified number of elements in the array, and the specified number of characters in a string.

```
param testArray array = [
  'one'
  'two'
  'three'
]
param elementsToSkip int = 2
param testString string = 'one two three'
param charactersToSkip int = 4

output arrayOutput array = skip(testArray, elementsToSkip)
output stringOutput string = skip(testString, charactersToSkip)
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayOutput	Array	["three"]
stringOutput	String	two three

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.

Namespace: sys.

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's larger than the length of the given array or string, all the elements in the array or string are returned.

Return value

An array or string.

Example

The following example takes the specified number of elements from the array, and characters from a string.

```
param testArray array = [
  'one'
  'two'
  'three'
]
param elementsToTake int = 2
param testString string = 'one two three'
param charactersToTake int = 2

output arrayOutput array = take(testArray, elementsToTake)
output stringOutput string = take(testString, charactersToTake)
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayOutput	Array	["one", "two"]
stringOutput	String	on

union

`union(arg1, arg2, arg3, ...)`

Returns a single array or object with all elements from the parameters. For arrays, duplicate values are included once. For objects, duplicate property names are only included once.

Namespace: sys.

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.

PARAMETER	REQUIRED	TYPE	DESCRIPTION
more arguments	No	array or object	More values to use for joining elements.

Return value

An array or object.

Remarks

The union function uses the sequence of the parameters to determine the order and values of the result.

For arrays, the function iterates through each element in the first parameter and adds it to the result if it isn't already present. Then, it repeats the process for the second parameter and any more parameters. If a value is already present, its earlier placement in the array is preserved.

For objects, property names and values from the first parameter are added to the result. For later parameters, any new names are added to the result. If a later parameter has a property with the same name, that value overwrites the existing value. The order of the properties isn't guaranteed.

Example

The following example shows how to use union with arrays and objects:

```
param firstObject object = {
  'one': 'a'
  'two': 'b'
  'three': 'c1'
}

param secondObject object = {
  'three': 'c2'
  'four': 'd'
  'five': 'e'
}

param firstArray array = [
  'one'
  'two'
  'three'
]

param secondArray array = [
  'three'
  'four'
  'two'
]

output objectOutput object = union(firstObject, secondObject)
output arrayOutput array = union(firstArray, 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"]

Next steps

- To get an array of string values delimited by a value, see split.

Date functions for Bicep

9/21/2022 • 5 minutes to read • Edit Online

This article describes the Bicep functions for working with dates.

dateTimeAdd

`dateTimeAdd(base, duration, [format])`

Adds a time duration to a base value. ISO 8601 format is expected.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
base	Yes	string	The starting datetime value for the addition. Use ISO 8601 timestamp format.
duration	Yes	string	The time value to add to the base. It can be a negative value. Use ISO 8601 duration format.
format	No	string	The output format for the date time result. If not provided, the format of the base value is used. Use either standard format strings or custom format strings.

Return value

The datetime value that results from adding the duration value to the base value.

Examples

The following example shows different ways of adding time values.

```
param baseTime string = utcNow('u')

var add3Years = dateTimeAdd(baseTime, 'P3Y')
var subtract9Days = dateTimeAdd(baseTime, '-P9D')
var add1Hour = dateTimeAdd(baseTime, 'PT1H')

output add3YearsOutput string = add3Years
output subtract9DaysOutput string = subtract9Days
output add1HourOutput string = add1Hour
```

When the preceding example is deployed with a base time of `2020-04-07 14:53:14Z`, the output is:

NAME	TYPE	VALUE
add3YearsOutput	String	4/7/2023 2:53:14 PM
subtract9DaysOutput	String	3/29/2020 2:53:14 PM
add1HourOutput	String	4/7/2020 3:53:14 PM

The next example shows how to set the start time for an Automation schedule.

```
param omsAutomationAccountName string = 'demoAutomation'
param scheduleName string = 'demSchedule1'
param baseTime string = utcNow('u')

var startTime = dateTimeAdd(baseTime, 'PT1H')

...

resource scheduler 'Microsoft.Automation/automationAccounts/schedules@2015-10-31' = {
  name: concat(omsAutomationAccountName, '/', scheduleName)
  properties: {
    description: 'Demo Scheduler'
    startTime: startTime
    interval: 1
    frequency: 'Hour'
  }
}
```

dateTimeFromEpoch

`dateTimeFromEpoch(epochTime)`

Converts an epoch time integer value to an ISO 8601 datetime.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
epochTime	Yes	int	The epoch time to convert to a datetime string.

Return value

An ISO 8601 datetime string.

Remarks

This function requires **Bicep version 0.5.6 or later**.

Example

The following example shows output values for the epoch time functions.

```
param convertedEpoch int = dateTimeToEpoch(dateTimeAdd(utcNow(), 'P1Y'))

var convertedDatetime = dateTimeFromEpoch(convertedEpoch)

output epochValue int = convertedEpoch
output datetimeValue string = convertedDatetime
```

The output is:

NAME	TYPE	VALUE
datetimeValue	String	2023-05-02T15:16:13Z
epochValue	Int	1683040573

dateTimeToEpoch

`dateTimeToEpoch(dateTime)`

Converts an ISO 8601 datetime string to an epoch time integer value.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
dateTime	Yes	string	The datetime string to convert to an epoch time.

Return value

An integer that represents the number of seconds from midnight on January 1, 1970.

Remarks

This function requires **Bicep version 0.5.6 or later**.

Examples

The following example shows output values for the epoch time functions.

```
param convertedEpoch int = dateTimeToEpoch(dateTimeAdd(utcNow(), 'P1Y'))

var convertedDatetime = dateTimeFromEpoch(convertedEpoch)

output epochValue int = convertedEpoch
output datetimeValue string = convertedDatetime
```

The output is:

NAME	TYPE	VALUE
datetimeValue	String	2023-05-02T15:16:13Z
epochValue	Int	1683040573

The next example uses the epoch time value to set the expiration for a key in a key vault.

```
@description('The location into which the resources should be deployed.')
param location string = resourceGroup().location

@description('The Tenant Id that should be used throughout the deployment.')
param tenantId string = subscription().tenantId

@description('The name of the existing User Assigned Identity.')
param userAssignedIdentityName string

@description('The name of the resource group for the User Assigned Identity.')
param userAssignedIdentityResourceGroupName string

@description('The name of the Key Vault.')
param keyVaultName string  = 'vault-${uniqueString(resourceGroup().id)}'

@description('Name of the key in the Key Vault')
param keyVaultKeyName string = 'cmkey'

@description('Expiration time of the key')
param keyExpiration int = dateTimeToEpoch(dateTimeAdd(utcNow(), 'P1Y'))

@description('The name of the Storage Account')
param storageAccountName string =  'storage${uniqueString(resourceGroup().id)}'


resource userAssignedIdentity 'Microsoft.ManagedIdentity/userAssignedIdentities@2018-11-30' existing = {
  scope: resourceGroup(userAssignedIdentityResourceGroupName)
  name: userAssignedIdentityName
}

resource keyVault 'Microsoft.KeyVault/vaults@2021-10-01' = {
  name: keyVaultName
  location: location
  properties: {
    sku: {
      name: 'standard'
      family: 'A'
    }
    enableSoftDelete: true
    enablePurgeProtection: true
    enabledForDiskEncryption: true
    tenantId: tenantId
    accessPolicies: [
      {
        tenantId: tenantId
        permissions: {
          keys: [
            'unwrapKey'
            'wrapKey'
            'get'
          ]
        }
        objectId: userAssignedIdentity.properties.principalId
      }
    ]
  }
}

resource kvKey 'Microsoft.KeyVault/vaults/keys@2021-10-01' = {
  parent: keyVault
  name: keyVaultKeyName
  properties: {
    attributes: {
      enabled: true
      exp: keyExpiration
    }
    keySize: 4096
    kty: 'RSA'
```

```
    }
  }

  resource storage 'Microsoft.Storage/storageAccounts@2021-04-01' = {
    name: storageAccountName
    location: location
    sku: {
      name: 'Standard_LRS'
    }
    kind: 'StorageV2'
    identity: {
      type: 'UserAssigned'
      userAssignedIdentities: {
        '${userAssignedIdentity.id}': {}
      }
    }
    properties: {
      accessTier: 'Hot'
      supportsHttpsTrafficOnly: true
      minimumTlsVersion: 'TLS1_2'
      encryption: {
        identity: {
          userAssignedIdentity: userAssignedIdentity.id
        }
        services: {
           blob: {
             enabled: true
           }
        }
        keySource: 'Microsoft.Keyvault'
        keyvaultproperties: {
          keyname: kvKey.name
          keyvaulturi: endsWith(keyVault.properties.vaultUri,'/') ?
  substring(keyVault.properties.vaultUri,0,length(keyVault.properties.vaultUri)-1) :
  keyVault.properties.vaultUri
        }
      }
    }
  }
```

utcNow

`utcNow(format)`

Returns the current (UTC) datetime value in the specified format. If no format is provided, the ISO 8601 (`yyyyMMddTHHmmssZ`) format is used. **This function can only be used in the default value for a parameter.**

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
format	No	string	The URI encoded value to convert to a string. Use either standard format strings or custom format strings.

Remarks

You can only use this function within an expression for the default value of a parameter. Using this function anywhere else in a Bicep file returns an error. The function isn't allowed in other parts of the Bicep file because it returns a different value each time it's called. Deploying the same Bicep file with the same parameters wouldn't

reliably produce the same results.

If you use the option to rollback on error to an earlier successful deployment, and the earlier deployment includes a parameter that uses utcNow, the parameter isn't reevaluated. Instead, the parameter value from the earlier deployment is automatically reused in the rollback deployment.

Be careful redeploying a Bicep file that relies on the utcNow function for a default value. When you redeploy and don't provide a value for the parameter, the function is reevaluated. If you want to update an existing resource rather than create a new one, pass in the parameter value from the earlier deployment.

Return value

The current UTC datetime value.

Examples

The following example shows different formats for the datetime value.

```
param utcValue string = utcNow()
param utcShortValue string = utcNow('d')
param utcCustomValue string = utcNow('M d')

output utcOutput string = utcValue
output utcShortOutput string = utcShortValue
output utcCustomOutput string = utcCustomValue
```

The output from the preceding example varies for each deployment but will be similar to:

NAME	TYPE	VALUE
utcOutput	string	20190305T175318Z
utcShortOutput	string	03/05/2019
utcCustomOutput	string	3 5

The next example shows how to use a value from the function when setting a tag value.

```
param utcShort string = utcNow('d')
param rgName string

resource myRg 'Microsoft.Resources/resourceGroups@2020-10-01' = {
  name: rgName
  location: 'westeurope'
  tags: {
    createdDate: utcShort
  }
}

output utcShortOutput string = utcShort
```

Next steps

- For a description of the sections in a Bicep file, see Understand the structure and syntax of Bicep files.

Deployment functions for Bicep

9/21/2022 • 2 minutes to read • Edit Online

This article describes the Bicep functions for getting values related to the current deployment.

deployment

`deployment()`

Returns information about the current deployment operation.

Namespace: az.

Return value

This function returns the object that is passed during deployment. The properties in the returned object differ based on whether you are:

- deploying a local Bicep file.
- deploying to a resource group or deploying to one of the other scopes (Azure subscription, management group, or tenant).

When deploying a local Bicep file to a resource group, the function returns the following format:

```
{
  "name": "",
  "properties": {
    "template": {
      "$schema": "",
      "contentVersion": "",
      "parameters": {},
      "variables": {},
      "resources": [],
      "outputs": {}
    },
    "templateHash": "",
    "parameters": {},
    "mode": "",
    "provisioningState": ""
  }
}
```

When you deploy to an Azure subscription, management group, or tenant, the return object includes a `location` property. The `location` property is not included when deploying a local Bicep file. The format is:

```
{
  "name": "",
  "location": "",
  "properties": {
    "template": {
      "$schema": "",
      "contentVersion": "",
      "resources": [],
      "outputs": {}
    },
    "templateHash": "",
    "parameters": {},
    "mode": "",
    "provisioningState": ""
  }
}
```

Example

The following example returns the deployment object:

```
output deploymentOutput object = deployment()
```

The preceding example returns the following object:

```
{
  "name": "deployment",
  "properties": {
    "template": {
      "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
      "contentVersion": "1.0.0.0",
      "resources": [],
      "outputs": {
        "deploymentOutput": {
          "type": "Object",
          "value": "[deployment()]"
        }
      }
    },
    "templateHash": "13135986259522608210",
    "parameters": {},
    "mode": "Incremental",
    "provisioningState": "Accepted"
  }
}
```

environment

`environment()`

Returns information about the Azure environment used for deployment.

Namespace: az.

Remarks

To see a list of registered environments for your account, use az cloud list or Get-AzEnvironment.

Return value

This function returns properties for the current Azure environment. The following example shows the properties for global Azure. Sovereign clouds may return slightly different properties.

```
{
  "name": "",
  "gallery": "",
  "graph": "",
  "portal": "",
  "graphAudience": "",
  "activeDirectoryDataLake": "",
  "batch": "",
  "media": "",
  "sqlManagement": "",
  "vmImageAliasDoc": "",
  "resourceManager": "",
  "authentication": {
    "loginEndpoint": "",
    "audiences": [
      "",
      ""
    ],
    "tenant": "",
    "identityProvider": ""
  },
  "suffixes": {
    "acrLoginServer": "",
    "azureDatalakeAnalyticsCatalogAndJob": "",
    "azureDatalakeStoreFileSystem": "",
    "azureFrontDoorEndpointSuffix": "",
    "keyvaultDns": "",
    "sqlServerHostname": "",
    "storage": ""
  }
}
```

Example

The following example Bicep file returns the environment object.

```
output environmentOutput object = environment()
```

The preceding example returns the following object when deployed to global Azure:

```
{
  "name": "AzureCloud",
  "gallery": "https://gallery.azure.com/",
  "graph": "https://graph.windows.net/",
  "portal": "https://portal.azure.com",
  "graphAudience": "https://graph.windows.net/",
  "activeDirectoryDataLake": "https://datalake.azure.net/",
  "batch": "https://batch.core.windows.net/",
  "media": "https://rest.media.azure.net",
  "sqlManagement": "https://management.core.windows.net:8443/",
  "vmImageAliasDoc": "https://raw.githubusercontent.com/Azure/azure-rest-api-specs/master/arm-
compute/quickstart-templates/aliases.json",
  "resourceManager": "https://management.azure.com/",
  "authentication": {
    "loginEndpoint": "https://login.windows.net/",
    "audiences": [
      "https://management.core.windows.net/",
      "https://management.azure.com/"
    ],
    "tenant": "common",
    "identityProvider": "AAD"
  },
  "suffixes": {
    "acrLoginServer": ".azurecr.io",
    "azureDatalakeAnalyticsCatalogAndJob": "azuredatalakeanalytics.net",
    "azureDatalakeStoreFileSystem": "azuredatalakestore.net",
    "azureFrontDoorEndpointSuffix": "azurefd.net",
    "keyvaultDns": ".vault.azure.net",
    "sqlServerHostname": ".database.windows.net",
    "storage": "core.windows.net"
  }
}
```

Next steps

- To get values from resources, resource groups, or subscriptions, see Resource functions.

File functions for Bicep

9/21/2022 • 3 minutes to read • Edit Online

This article describes the Bicep functions for loading content from external files.

loadFileAsBase64

`loadFileAsBase64(filePath)`

Loads the file as a base64 string.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
filePath	Yes	string	The path to the file to load. The path is relative to the deployed Bicep file.

Remarks

Use this function when you have binary content you would like to include in deployment. Rather than manually encoding the file to a base64 string and adding it to your Bicep file, load the file with this function. The file is loaded when the Bicep file is compiled to a JSON template. During deployment, the JSON template contains the contents of the file as a hard-coded string.

This function requires **Bicep version 0.4.412 or later**.

The maximum allowed size of the file is **96 Kb**.

Return value

The file as a base64 string.

loadJsonContent

`loadJsonContent(filePath, [jsonPath], [encoding])`

Loads the specified JSON file as an Any object.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
filePath	Yes	string	The path to the file to load. The path is relative to the deployed Bicep file.
jsonPath	No	string	JSONPath expression to take only a part of the JSON into ARM.

PARAMETER	REQUIRED	TYPE	DESCRIPTION
encoding	No	string	The file encoding. The default value is `utf-8`. The available options are: `iso-8859-1`, `us-ascii`, `utf-16`, `utf-16BE`, or `utf-8`.

Remarks

Use this function when you have JSON content or minified JSON content that is stored in a separate file. Rather than duplicating the JSON content in your Bicep file, load the content with this function. You can load a part of a JSON file by specifying a JSON path. The file is loaded when the Bicep file is compiled to the JSON template. During deployment, the JSON template contains the contents of the file as a hard-coded string.

In VS Code, the properties of the loaded object are available intellisense. For example, you can create a file with values to share across many Bicep files. An example is shown in this article.

This function requires **Bicep version 0.7.4 or later**.

The maximum allowed size of the file is **1,048,576 characters**, including line endings.

Return value

The contents of the file as an Any object.

Examples

The following example creates a JSON file that contains values for a network security group.

```
{
  "description": "Allows SSH traffic",
  "protocol": "Tcp",
  "sourcePortRange": "*",
  "destinationPortRange": "22",
  "sourceAddressPrefix": "*",
  "destinationAddressPrefix": "*",
  "access": "Allow",
  "priority": 100,
  "direction": "Inbound"
}
```

You load that file and convert it to a JSON object. You use the object to assign values to the resource.

```
param location string = resourceGroup().location

var nsgconfig = loadJsonContent('nsg-security-rules.json')

resource newNSG 'Microsoft.Network/networkSecurityGroups@2021-02-01' = {
  name: 'example-nsg'
  location: location
  properties: {
    securityRules: [
      {
        name: 'SSH'
        properties: {
          description: nsgconfig.description
          protocol: nsgconfig.protocol
          sourcePortRange: nsgconfig.sourcePortRange
          destinationPortRange: nsgconfig.destinationPortRange
          sourceAddressPrefix: nsgconfig.sourceAddressPrefix
          destinationAddressPrefix: nsgconfig.destinationAddressPrefix
          access: nsgconfig.access
          priority: nsgconfig.priority
          direction: nsgconfig.direction
        }
      }
    ]
  }
}
```

You can reuse the file of values in other Bicep files that deploy a network security group.

loadTextContent

`loadTextContent(filePath, [encoding])`

Loads the content of the specified file as a string.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
filePath	Yes	string	The path to the file to load. The path is relative to the deployed Bicep file.
encoding	No	string	The file encoding. The default value is `utf-8`. The available options are: `iso-8859-1`, `us-ascii`, `utf-16`, `utf-16BE`, or `utf-8`.

Remarks

Use this function when you have content that is more stored in a separate file. Rather than duplicating the content in your Bicep file, load the content with this function. For example, you can load a deployment script from a file. The file is loaded when the Bicep file is compiled to the JSON template. During deployment, the JSON template contains the contents of the file as a hard-coded string.

Use the `loadJsonContent()` function to load JSON files.

This function requires **Bicep version 0.4.412 or later**.

The maximum allowed size of the file is **131,072 characters**, including line endings.

Return value

The contents of the file as a string.

Examples

The following example loads a script from a file and uses it for a deployment script.

```
resource exampleScript 'Microsoft.Resources/deploymentScripts@2020-10-01' = {
  name: 'exampleScript'
  location: resourceGroup().location
  kind: 'AzurePowerShell'
  identity: {
    type: 'UserAssigned'
    userAssignedIdentities: {
      '/subscriptions/{sub-id}/resourcegroups/{rg-
name}/providers/Microsoft.ManagedIdentity/userAssignedIdentities/{id-name}': {}
    }
  }
  properties: {
    azPowerShellVersion: '3.0'
    scriptContent: loadTextContent('myscript.ps1')
    retentionInterval: 'P1D'
  }
}
```

Next steps

- For a description of the sections in a Bicep file, see Understand the structure and syntax of Bicep files.

Lambda functions for Bicep

9/21/2022 • 4 minutes to read • Edit Online

This article describes the lambda functions to use in Bicep. Lambda expressions (or lambda functions) are essentially blocks of code that can be passed as an argument. In Bicep, lambda expression is in this format:

```
<lambda variable> => <expression>
```

> **NOTE**
> The lambda functions are only supported in Bicep CLI version 0.10.61 or newer.

Limitations

Bicep lambda function has these limitations:

- Lambda expression can only be specified directly as function arguments in these functions: `filter()`, `map()`, `reduce()`, and `sort()`.
- Using lambda variables (the temporary variables used in the lambda expressions) inside resource or module array access isn't currently supported.
- Using lambda variables inside the `listKeys` function isn't currently supported.
- Using lambda variables inside the reference function isn't currently supported.

filter

`filter(inputArray, lambda expression)`

Filters an array with a custom filtering function.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
inputArray	Yes	array	The array to filter.
lambda expression	Yes	expression	The lambda expression applied to each input array element. If false, the item will be filtered out of the output array.

Return value

An array.

Examples

The following examples show how to use the filter function.

```
var dogs = [
  {
    name: 'Evie'
    age: 5
    interests: ['Ball', 'Frisbee']
  }
  {
    name: 'Casper'
    age: 3
    interests: ['Other dogs']
  }
  {
    name: 'Indy'
    age: 2
    interests: ['Butter']
  }
  {
    name: 'Kira'
    age: 8
    interests: ['Rubs']
  }
]

output oldDogs array = filter(dogs, dog => dog.age >=5)
```

The output from the preceding example shows the dogs that are five or older:

NAME	TYPE	VALUE
oldDogs	Array	[{"name":"Evie","age":5,"interests":["Ball","Frisbee"]},{"name":"Kira","age":8,"interests":["Rubs"]}]

```
var itemForLoop = [for item in range(0, 10): item]

output filteredLoop array = filter(itemForLoop, i => i > 5)
output isEven array = filter(range(0, 10), i => 0 == i % 2)
```

The output from the preceding example:

NAME	TYPE	VALUE
filteredLoop	Array	[6, 7, 8, 9]
isEven	Array	[0, 2, 4, 6, 8]

filterdLoop shows the numbers in an array that are greater than 5; and **isEven** shows the even numbers in the array.

map

`map(inputArray, lambda expression)`

Applies a custom mapping function to each element of an array.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
inputArray	Yes	array	The array to map.
lambda expression	Yes	expression	The lambda expression applied to each input array element, in order to generate the output array.

Return value

An array.

Example

The following example shows how to use the map function.

```
var dogs = [
  {
    name: 'Evie'
    age: 5
    interests: ['Ball', 'Frisbee']
  }
  {
    name: 'Casper'
    age: 3
    interests: ['Other dogs']
  }
  {
    name: 'Indy'
    age: 2
    interests: ['Butter']
  }
  {
    name: 'Kira'
    age: 8
    interests: ['Rubs']
  }
]

output dogNames array = map(dogs, dog => dog.name)
output sayHi array = map(dogs, dog => 'Hello ${dog.name}!')
output mapObject array = map(range(0, length(dogs)), i => {
  i: i
  dog: dogs[i].name
  greeting: 'Ahoy, ${dogs[i].name}!'
})
```

The output from the preceding example is:

NAME	TYPE	VALUE
dogNames	Array	["Evie","Casper","Indy","Kira"]
sayHi	Array	["Hello Evie!","Hello Casper!","Hello Indy!","Hello Kira!"]

NAME	TYPE	VALUE
mapObject	Array	[{"i":0,"dog":"Evie","greeting":"Ahoy, Evie!"}, {"i":1,"dog":"Casper","greeting":"Ahoy, Casper!"}, {"i":2,"dog":"Indy","greeting":"Ahoy, Indy!"}, {"i":3,"dog":"Kira","greeting":"Ahoy, Kira!"}]

dogNames shows the dog names from the array of objects; **sayHi** concatenates "Hello" and each of the dog names; and **mapObject** creates another array of objects.

reduce

```
reduce(inputArray, initialValue, lambda expression)
```

Reduces an array with a custom reduce function.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
inputArray	Yes	array	The array to reduce.
initialValue	No	any	Initial value.
lambda expression	Yes	expression	The lambda expression used to aggregate the current value and the next value.

Return value

Any.

Example

The following examples show how to use the reduce function.

```
var dogs = [
  {
    name: 'Evie'
    age: 5
    interests: ['Ball', 'Frisbee']
  }
  {
    name: 'Casper'
    age: 3
    interests: ['Other dogs']
  }
  {
    name: 'Indy'
    age: 2
    interests: ['Butter']
  }
  {
    name: 'Kira'
    age: 8
    interests: ['Rubs']
  }
]
var ages = map(dogs, dog => dog.age)
output totalAge int = reduce(ages, 0, (cur, prev) => cur + prev)
output totalAgeAdd1 int = reduce(ages, 1, (cur, prev) => cur + prev)
```

The output from the preceding example is:

NAME	TYPE	VALUE
totalAge	int	18
totalAgeAdd1	int	19

totalAge sums the ages of the dogs; **totalAgeAdd1** has an initial value of 1, and adds all the dog ages to the initial values.

```
output reduceObjectUnion object = reduce([
  { foo: 123 }
  { bar: 456 }
  { baz: 789 }
], {}, (cur, next) => union(cur, next))
```

The output from the preceding example is:

NAME	TYPE	VALUE
reduceObjectUnion	object	{"foo":123,"bar":456,"baz":789}

The union function returns a single object with all elements from the parameters. The function call unionizes the key value pairs of the objects into a new object.

sort

`sort(inputArray, lambda expression)`

Sorts an array with a custom sort function.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
inputArray	Yes	array	The array to sort.
lambda expression	Yes	expression	The lambda expression used to compare two array elements for ordering. If true, the second element will be ordered after the first in the output array.

Return value

An array.

Example

The following example shows how to use the sort function.

```
var dogs = [
  {
    name: 'Evie'
    age: 5
    interests: ['Ball', 'Frisbee']
  }
  {
    name: 'Casper'
    age: 3
    interests: ['Other dogs']
  }
  {
    name: 'Indy'
    age: 2
    interests: ['Butter']
  }
  {
    name: 'Kira'
    age: 8
    interests: ['Rubs']
  }
]

output dogsByAge array = sort(dogs, (a, b) => a.age < b.age)
```

The output from the preceding example sorts the dog objects from the youngest to the oldest:

NAME	TYPE	VALUE
dogsByAge	Array	[{"name":"Indy","age":2,"interests":["Butter"]},{"name":"Casper","age":3,"interests":["Other dogs"]},{"name":"Evie","age":5,"interests":["Ball","Frisbee"]},{"name":"Kira","age":8,"interests":["Rubs"]}]

Next steps

- See Bicep functions - arrays for additional array related Bicep functions.

Logical functions for Bicep

9/21/2022 • 2 minutes to read • Edit Online

Bicep provides the `bool` function for converting values to a boolean.

Most of the logical functions in Azure Resource Manager templates are replaced with logical operators in Bicep.

bool

`bool(arg1)`

Converts the parameter to a boolean.

Namespace: sys.

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 shows how to use bool with a string or integer.

```
output trueString bool = bool('true')
output falseString bool = bool('false')
output trueInt bool = bool(1)
output falseInt bool = bool(0)
```

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

Next steps

- For other actions involving logical values, see logical operators.

Numeric functions for Bicep

9/21/2022 • 2 minutes to read • Edit Online

This article describes the Bicep functions for working with integers.

Some of the Azure Resource Manager JSON numeric functions are replaced with Bicep numeric operators.

int

`int(valueToConvert)`

Converts the specified value to an integer.

Namespace: sys.

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 converts the user-provided parameter value to integer.

```
param stringToConvert string = '4'

output inResult int = int(stringToConvert)
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
intResult	Int	4

max

`max(arg1)`

Returns the maximum value from an array of integers or a comma-separated list of integers.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION

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 shows how to use max with an array and a list of integers:

```
param arrayToTest array = [
  0
  3
  2
  5
  4
]

output arrayOutPut int = max(arrayToTest)
output intOutput int = 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

min

`min(arg1)`

Returns the minimum value from an array of integers or a comma-separated list of integers.

Namespace: sys.

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 shows how to use min with an array and a list of integers:

```
param arrayToTest array = [
  0
  3
  2
  5
  4
]

output arrayOutPut int = min(arrayToTest)
output intOutput int = 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

Next steps

- For other actions involving numbers, see Bicep numeric operators.

Object functions for Bicep

9/21/2022 • 7 minutes to read • Edit Online

This article describes the Bicep functions for working with objects.

contains

`contains(container, itemToFind)`

Checks whether an array contains a value, an object contains a key, or a string contains a substring. The string comparison is case-sensitive. However, when testing if an object contains a key, the comparison is case-insensitive.

Namespace: sys.

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 shows how to use contains with different types:

```
param stringToTest string = 'OneTwoThree'
param objectToTest object = {
  'one': 'a'
  'two': 'b'
  'three': 'c'
}
param arrayToTest array = [
  'one'
  'two'
  'three'
]

output stringTrue bool = contains(stringToTest, 'e')
output stringFalse bool = contains(stringToTest, 'z')
output objectTrue bool = contains(objectToTest, 'one')
output objectFalse bool = contains(objectToTest, 'a')
output arrayTrue bool = contains(arrayToTest, 'three')
output arrayFalse bool = contains(arrayToTest, 'four')
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
stringTrue	Bool	True

NAME	TYPE	VALUE
stringFalse	Bool	False
objectTrue	Bool	True
objectFalse	Bool	False
arrayTrue	Bool	True
arrayFalse	Bool	False

empty

`empty(itemToTest)`

Determines if an array, object, or string is empty.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
itemToTest	Yes	array, object, or string	The value to check if it's empty.

Return value

Returns **True** if the value is empty; otherwise, **False**.

Example

The following example checks whether an array, object, and string are empty.

```
param testArray array = []
param testObject object = {}
param testString string = ''

output arrayEmpty bool = empty(testArray)
output objectEmpty bool = empty(testObject)
output stringEmpty bool = empty(testString)
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayEmpty	Bool	True
objectEmpty	Bool	True
stringEmpty	Bool	True

intersection

`intersection(arg1, arg2, arg3, ...)`

Returns a single array or object with the common elements from the parameters.

Namespace: sys.

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 shows how to use intersection with arrays and objects:

```
param firstObject object = {
  'one': 'a'
  'two': 'b'
  'three': 'c'
}
param secondObject object = {
  'one': 'a'
  'two': 'z'
  'three': 'c'
}
param firstArray array = [
  'one'
  'two'
  'three'
]
param secondArray array = [
  'two'
  'three'
]

output objectOutput object = intersection(firstObject, secondObject)
output arrayOutput array = intersection(firstArray, 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"]

items

`items(object)`

Converts a dictionary object to an array.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
object	Yes	object	The dictionary object to convert to an array.

Return value

An array of objects for the converted dictionary. Each object in the array has a `key` property that contains the key value for the dictionary. Each object also has a `value` property that contains the properties for the object.

Example

The following example converts a dictionary object to an array. For each object in the array, it creates a new object with modified values.

```
var entities = {
  item002: {
    enabled: false
    displayName: 'Example item 2'
    number: 200
  }
  item001: {
    enabled: true
    displayName: 'Example item 1'
    number: 300
  }
}

var modifiedListOfEntities = [for entity in items(entities): {
  key: entity.key
  fullName: entity.value.displayName
  itemEnabled: entity.value.enabled
}]

output modifiedResult array = modifiedListOfEntities
```

The preceding example returns:

```
"modifiedResult": {
  "type": "Array",
  "value": [
    {
      "fullName": "Example item 1",
      "itemEnabled": true,
      "key": "item001"
    },
    {
      "fullName": "Example item 2",
      "itemEnabled": false,
      "key": "item002"
    }
  ]
}
```

The following example shows the array that is returned from the items function.

```
var entities = {
  item002: {
    enabled: false
    displayName: 'Example item 2'
    number: 200
  }
  item001: {
    enabled: true
    displayName: 'Example item 1'
    number: 300
  }
}

var entitiesArray = items(entities)

output itemsResult array = entitiesArray
```

The example returns:

```
"itemsResult": {
  "type": "Array",
  "value": [
    {
      "key": "item001",
      "value": {
        "displayName": "Example item 1",
        "enabled": true,
        "number": 300
      }
    },
    {
      "key": "item002",
      "value": {
        "displayName": "Example item 2",
        "enabled": false,
        "number": 200
      }
    }
  ]
}
```

In JSON, an object is an unordered collection of zero or more key/value pairs. The ordering can be different depending on the implementations. For example, the Bicep items() function sorts the objects in the alphabetical order. In other places, the original ordering can be preserved. Because of this non-determinism, avoid making any assumptions about the ordering of object keys when writing code, which interacts with deployments parameters & outputs.

json

`json(arg1)`

Converts a valid JSON string into a JSON data type.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	string	The value to convert to JSON. The string must be a properly formatted JSON string.

Return value

The JSON data type from the specified string, or an empty value when **null** is specified.

Remarks

If you need to include a parameter value or variable in the JSON object, use the concat function to create the string that you pass to the function.

Example

The following example shows how to use the json function. Notice that you can pass in **null** for an empty object.

```
param jsonEmptyObject string = 'null'
param jsonObject string = '{\'a\': \'b\'}'
param jsonString string = '\'test\''
param jsonBoolean string = 'true'
param jsonInt string = '3'
param jsonArray string = '[[1,2,3]]'
param concatValue string = 'demo value'

output emptyObjectOutput bool = empty(json(jsonEmptyObject))
output objectOutput object = json(jsonObject)
output stringOutput string =json(jsonString)
output booleanOutput bool = json(jsonBoolean)
output intOutput int = json(jsonInt)
output arrayOutput array = json(jsonArray)
output concatObjectOutput object = json(concat('{"a": "', concatValue, '"}'))
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
emptyObjectOutput	Boolean	True
objectOutput	Object	{"a": "b"}
stringOutput	String	test
booleanOutput	Boolean	True
intOutput	Integer	3
arrayOutput	Array	[1, 2, 3]
concatObjectOutput	Object	{ "a": "demo value" }

length

`length(arg1)`

Returns the number of elements in an array, characters in a string, or root-level properties in an object.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	array, string, or object	The array to use for getting the number of elements, the string to use for getting the number of characters, or the object to use for getting the number of root-level properties.

Return value

An int.

Example

The following example shows how to use length with an array and string:

```
param arrayToTest array = [
  'one'
  'two'
  'three'
]
param stringToTest string = 'One Two Three'
param objectToTest object = {
  'propA': 'one'
  'propB': 'two'
  'propC': 'three'
  'propD': {
      'propD-1': 'sub'
      'propD-2': 'sub'
  }
}

output arrayLength int = length(arrayToTest)
output stringLength int = length(stringToTest)
output objectLength int = length(objectToTest)
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayLength	Int	3
stringLength	Int	13
objectLength	Int	4

union

`union(arg1, arg2, arg3, ...)`

Returns a single array or object with all elements from the parameters. For arrays, duplicate values are included once. For objects, duplicate property names are only included once.

Namespace: sys.

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.

Remarks

The union function uses the sequence of the parameters to determine the order and values of the result.

For arrays, the function iterates through each element in the first parameter and adds it to the result if it isn't already present. Then, it repeats the process for the second parameter and any additional parameters. If a value is already present, it's earlier placement in the array is preserved.

For objects, property names and values from the first parameter are added to the result. For later parameters, any new names are added to the result. If a later parameter has a property with the same name, that value overwrites the existing value. The order of the properties isn't guaranteed.

Example

The following example shows how to use union with arrays and objects:

```
param firstObject object = {
  'one': 'a'
  'two': 'b'
  'three': 'c1'
}

param secondObject object = {
  'three': 'c2'
  'four': 'd'
  'five': 'e'
}

param firstArray array = [
  'one'
  'two'
  'three'
]

param secondArray array = [
  'three'
  'four'
  'two'
]

output objectOutput object = union(firstObject, secondObject)
output arrayOutput array = union(firstArray, 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"]

Next steps

- For a description of the sections in a Bicep file, see Understand the structure and syntax of Bicep files.

Resource functions for Bicep

9/21/2022 • 14 minutes to read • Edit Online

This article describes the Bicep functions for getting resource values.

To get values from the current deployment, see Deployment value functions.

extensionResourceId

`extensionResourceId(resourceId, resourceType, resourceName1, [resourceName2], ...)`

Returns the resource ID for an extension resource. An extension resource is a resource type that's applied to another resource to add to its capabilities.

Namespace: az.

The `extensionResourceId` function is available in Bicep files, but typically you don't need it. Instead, use the symbolic name for the resource and access the `id` property.

The basic format of the resource ID returned by this function is:

```
{scope}/providers/{extensionResourceProviderNamespace}/{extensionResourceType}/{extensionResourceName}
```

The scope segment varies by the resource being extended.

When the extension resource is applied to a **resource**, the resource ID is returned in the following format:

```
/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/{baseResourceProviderNamespace}
/{baseResourceType}/{baseResourceName}/providers/{extensionResourceProviderNamespace}/{extensionResourceType
}/{extensionResourceName}
```

When the extension resource is applied to a **resource group**, the format is:

```
/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/{extensionResourceProviderNames
pace}/{extensionResourceType}/{extensionResourceName}
```

When the extension resource is applied to a **subscription**, the format is:

```
/subscriptions/{subscriptionId}/providers/{extensionResourceProviderNamespace}/{extensionResourceType}/{exte
nsionResourceName}
```

When the extension resource is applied to a **management group**, the format is:

```
/providers/Microsoft.Management/managementGroups/{managementGroupName}/providers/{extensionResourceProviderN
amespace}/{extensionResourceType}/{extensionResourceName}
```

A custom policy definition deployed to a management group is implemented as an extension resource. To create and assign a policy, deploy the following Bicep file to a management group.

```
targetScope = 'managementGroup'

@description('An array of the allowed locations, all other locations will be denied by the created policy.')
param allowedLocations array = [
  'australiaeast'
  'australiasoutheast'
  'australiacentral'
]

resource policyDefinition 'Microsoft.Authorization/policyDefinitions@2019-09-01' = {
  name: 'locationRestriction'
  properties: {
    policyType: 'Custom'
    mode: 'All'
    parameters: {}
    policyRule: {
      if: {
        not: {
          field: 'location'
          in: allowedLocations
        }
      }
      then: {
        effect: 'deny'
      }
    }
  }
}

resource policyAssignment 'Microsoft.Authorization/policyAssignments@2019-09-01' = {
  name: 'locationAssignment'
  properties: {
    policyDefinitionId: policyDefinition.id
  }
}
```

Built-in policy definitions are tenant level resources. For an example of deploying a built-in policy definition, see tenantResourceId.

getSecret

`keyVaultName.getSecret(secretName)`

Returns a secret from an Azure Key Vault. Use this function to pass a secret to a secure string parameter of a Bicep module.

You can only use the `getSecret` function from within the `params` section of a module. You can only use it with a `Microsoft.KeyVault/vaults` resource.

```
module sql './sql.bicep' = {
  name: 'deploySQL'
  params: {
    adminPassword: keyVault.getSecret('vmAdminPassword')
  }
}
```

You'll get an error if you attempt to use this function in any other part of the Bicep file. You'll also get an error if you use this function with string interpolation, even when used in the params section.

The function can be used only with a module parameter that has the `@secure()` decorator.

The key vault must have `enabledForTemplateDeployment` set to `true`. The user deploying the Bicep file must have access to the secret. For more information, see Use Azure Key Vault to pass secure parameter value during Bicep deployment.

A namespace qualifier isn't needed because the function is used with a resource type.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
secretName	Yes	string	The name of the secret stored in a key vault.

Return value

The secret value for the secret name.

Example

The following Bicep file is used as a module. It has an `adminPassword` parameter defined with the `@secure()` decorator.

```
param sqlServerName string
param adminLogin string

@secure()
param adminPassword string

resource sqlServer 'Microsoft.Sql/servers@2020-11-01-preview' = {
  ...
}
```

The following Bicep file consumes the preceding Bicep file as a module. The Bicep file references an existing key vault, and calls the `getSecret` function to retrieve the key vault secret, and then passes the value as a parameter to the module.

```
param sqlServerName string
param adminLogin string

param subscriptionId string
param kvResourceGroup string
param kvName string

resource keyVault 'Microsoft.KeyVault/vaults@2019-09-01' existing = {
  name: kvName
  scope: resourceGroup(subscriptionId, kvResourceGroup )
}

module sql './sql.bicep' = {
  name: 'deploySQL'
  params: {
    sqlServerName: sqlServerName
    adminLogin: adminLogin
    adminPassword: keyVault.getSecret('vmAdminPassword')
  }
}
```

list*

`resourceName.list([apiVersion], [functionValues])`

You can call a list function for any resource type with an operation that starts with `list`. Some common usages are `list`, `listKeys`, `listKeyValue`, and `listSecrets`.

The syntax for this function varies by the name of the list operation. The returned values also vary by operation. Bicep doesn't currently support completions and validation for `list*` functions.

With **Bicep version 0.4.412 or later**, you call the list function by using the accessor operator. For example, `storageAccount.listKeys()`.

A namespace qualifier isn't needed because the function is used with a resource type.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
apiVersion	No	string	If you don't provide this parameter, the API version for the resource is used. Only provide a custom API version when you need the function to be run with a specific version. Use the format, **yyyy-mm-dd**.
functionValues	No	object	An object that has values for the function. Only provide this object for functions that support receiving an object with parameter values, such as `listAccountSas` on a storage account. An example of passing function values is shown in this article.

Valid uses

The `list` functions can be used in the properties of a resource definition. Don't use a `list` function that exposes sensitive information in the outputs section of a Bicep file. Output values are stored in the deployment history and could be retrieved by a malicious user.

When used with an iterative loop, you can use the `list` functions for `input` because the expression is assigned to the resource property. You can't use them with `count` because the count must be determined before the `list` function is resolved.

If you use a `list` function in a resource that is conditionally deployed, the function is evaluated even if the resource isn't deployed. You get an error if the `list` function refers to a resource that doesn't exist. Use the conditional expression **?:** operator to make sure the function is only evaluated when the resource is being deployed.

Return value

The returned object varies by the list function you use. For example, the `listKeys` for a storage account returns the following format:

```
{
  "keys": [
    {
      "keyName": "key1",
      "permissions": "Full",
      "value": "{value}"
    },
    {
      "keyName": "key2",
      "permissions": "Full",
      "value": "{value}"
    }
  ]
}
```

Other `list` functions have different return formats. To see the format of a function, include it in the outputs section as shown in the example Bicep file.

List example

The following example deploys a storage account and then calls `listKeys` on that storage account. The key is used when setting a value for deployment scripts.

```
resource storageAccount 'Microsoft.Storage/storageAccounts@2019-06-01' = {
  name: 'dscript${uniqueString(resourceGroup().id)}'
  location: location
  kind: 'StorageV2'
  sku: {
    name: 'Standard_LRS'
  }
}

resource dScript 'Microsoft.Resources/deploymentScripts@2019-10-01-preview' = {
  name: 'scriptWithStorage'
  location: location
  ...
  properties: {
    azCliVersion: '2.0.80'
    storageAccountSettings: {
      storageAccountName: storageAccount.name
      storageAccountKey: storageAccount.listKeys().keys[0].value
    }
    ...
  }
}
```

The next example shows a `list` function that takes a parameter. In this case, the function is `listAccountSas`. Pass an object for the expiry time. The expiry time must be in the future.

```
param accountSasProperties object {
  default: {
    signedServices: 'b'
    signedPermission: 'r'
    signedExpiry: '2020-08-20T11:00:00Z'
    signedResourceTypes: 's'
  }
}
...
sasToken: storageAccount.listAccountSas('2021-04-01', accountSasProperties).accountSasToken
```

Implementations

The possible uses of `list*` are shown in the following table.

RESOURCE TYPE	FUNCTION NAME
Microsoft.Addons/supportProviders	listsupportplaninfo
Microsoft.AnalysisServices/servers	listGatewayStatus
Microsoft.ApiManagement/service/authorizationServers	listSecrets
Microsoft.ApiManagement/service/gateways	listKeys
Microsoft.ApiManagement/service/identityProviders	listSecrets
Microsoft.ApiManagement/service/namedValues	listValue
Microsoft.ApiManagement/service/openidConnectProviders	listSecrets
Microsoft.ApiManagement/service/subscriptions	listSecrets
Microsoft.AppConfiguration/configurationStores	ListKeys
Microsoft.AppPlatform/Spring	listTestKeys
Microsoft.Automation/automationAccounts	listKeys
Microsoft.Batch/batchAccounts	listkeys
Microsoft.BatchAI/workspaces/experiments/jobs	listoutputfiles
Microsoft.BotService/botServices/channels	listChannelWithKeys
Microsoft.Cache/redis	listKeys
Microsoft.CognitiveServices/accounts	listKeys
Microsoft.ContainerRegistry/registries	listBuildSourceUploadUrl
Microsoft.ContainerRegistry/registries	listCredentials
Microsoft.ContainerRegistry/registries	listUsages
Microsoft.ContainerRegistry/registries/agentpools	listQueueStatus
Microsoft.ContainerRegistry/registries/buildTasks	listSourceRepositoryProperties
Microsoft.ContainerRegistry/registries/buildTasks/steps	listBuildArguments
Microsoft.ContainerRegistry/registries/taskruns	listDetails
Microsoft.ContainerRegistry/registries/webhooks	listEvents

RESOURCE TYPE	FUNCTION NAME
Microsoft.ContainerRegistry/registries/runs	listLogSasUrl
Microsoft.ContainerRegistry/registries/tasks	listDetails
Microsoft.ContainerService/managedClusters	listClusterAdminCredential
Microsoft.ContainerService/managedClusters	listClusterMonitoringUserCredential
Microsoft.ContainerService/managedClusters	listClusterUserCredential
Microsoft.ContainerService/managedClusters/accessProfiles	listCredential
Microsoft.DataBox/jobs	listCredentials
Microsoft.DataFactory/datafactories/gateways	listauthkeys
Microsoft.DataFactory/factories/integrationruntimes	listauthkeys
Microsoft.DataLakeAnalytics/accounts/storageAccounts/Containers	listSasTokens
Microsoft.DataShare/accounts/shares	listSynchronizations
Microsoft.DataShare/accounts/shareSubscriptions	listSourceShareSynchronizationSettings
Microsoft.DataShare/accounts/shareSubscriptions	listSynchronizationDetails
Microsoft.DataShare/accounts/shareSubscriptions	listSynchronizations
Microsoft.Devices/iotHubs	listkeys
Microsoft.Devices/iotHubs/iotHubKeys	listkeys
Microsoft.Devices/provisioningServices/keys	listkeys
Microsoft.Devices/provisioningServices	listkeys
Microsoft.DevTestLab/labs	ListVhds
Microsoft.DevTestLab/labs/schedules	ListApplicable
Microsoft.DevTestLab/labs/users/serviceFabrics	ListApplicableSchedules
Microsoft.DevTestLab/labs/virtualMachines	ListApplicableSchedules
Microsoft.DocumentDB/databaseAccounts	listConnectionStrings
Microsoft.DocumentDB/databaseAccounts	listKeys

RESOURCE TYPE	FUNCTION NAME
Microsoft.DocumentDB/databaseAccounts/notebookWorkspaces	listConnectionInfo
Microsoft.DomainRegistration	listDomainRecommendations
Microsoft.DomainRegistration/topLevelDomains	listAgreements
Microsoft.EventGrid/domains	listKeys
Microsoft.EventGrid/topics	listKeys
Microsoft.EventHub/namespaces/authorizationRules	listkeys
Microsoft.EventHub/namespaces/disasterRecoveryConfigs/authorizationRules	listkeys
Microsoft.EventHub/namespaces/eventhubs/authorizationRules	listkeys
Microsoft.ImportExport/jobs	listBitLockerKeys
Microsoft.Kusto/Clusters/Databases	ListPrincipals
Microsoft.LabServices/labs/users	list
Microsoft.LabServices/labs/virtualMachines	list
Microsoft.Logic/integrationAccounts/agreements	listContentCallbackUrl
Microsoft.Logic/integrationAccounts/assemblies	listContentCallbackUrl
Microsoft.Logic/integrationAccounts	listCallbackUrl
Microsoft.Logic/integrationAccounts	listKeyVaultKeys
Microsoft.Logic/integrationAccounts/maps	listContentCallbackUrl
Microsoft.Logic/integrationAccounts/partners	listContentCallbackUrl
Microsoft.Logic/integrationAccounts/schemas	listContentCallbackUrl
Microsoft.Logic/workflows	listCallbackUrl
Microsoft.Logic/workflows	listSwagger
Microsoft.Logic/workflows/runs/actions	listExpressionTraces
Microsoft.Logic/workflows/runs/actions/repetitions	listExpressionTraces
Microsoft.Logic/workflows/triggers	listCallbackUrl

RESOURCE TYPE	FUNCTION NAME
Microsoft.Logic/workflows/versions/triggers	listCallbackUrl
Microsoft.MachineLearning/webServices	listkeys
Microsoft.MachineLearning/Workspaces	listworkspacekeys
Microsoft.MachineLearningServices/workspaces/computes	listKeys
Microsoft.MachineLearningServices/workspaces/computes	listNodes
Microsoft.MachineLearningServices/workspaces	listKeys
Microsoft.Maps/accounts	listKeys
Microsoft.Media/mediaservices/assets	listContainerSas
Microsoft.Media/mediaservices/assets	listStreamingLocators
Microsoft.Media/mediaservices/streamingLocators	listContentKeys
Microsoft.Media/mediaservices/streamingLocators	listPaths
Microsoft.Network/applicationSecurityGroups	listIpConfigurations
Microsoft.NotificationHubs/Namespaces/authorizationRules	listkeys
Microsoft.NotificationHubs/Namespaces/NotificationHubs/authorizationRules	listkeys
Microsoft.OperationalInsights/workspaces	list
Microsoft.OperationalInsights/workspaces	listKeys
Microsoft.PolicyInsights/remediations	listDeployments
Microsoft.RedHatOpenShift/openShiftClusters	listCredentials
Microsoft.Relay/namespaces/authorizationRules	listkeys
Microsoft.Relay/namespaces/disasterRecoveryConfigs/authorizationRules	listkeys
Microsoft.Relay/namespaces/HybridConnections/authorizationRules	listkeys
Microsoft.Relay/namespaces/WcfRelays/authorizationRules	listkeys
Microsoft.Search/searchServices	listAdminKeys
Microsoft.Search/searchServices	listQueryKeys

RESOURCE TYPE	FUNCTION NAME
Microsoft.ServiceBus/namespaces/authorizationRules	listkeys
Microsoft.ServiceBus/namespaces/disasterRecoveryConfigs/authorizationRules	listkeys
Microsoft.ServiceBus/namespaces/queues/authorizationRules	listkeys
Microsoft.ServiceBus/namespaces/topics/authorizationRules	listkeys
Microsoft.SignalRService/SignalR	listkeys
Microsoft.Storage/storageAccounts	listAccountSas
Microsoft.Storage/storageAccounts	listkeys
Microsoft.Storage/storageAccounts	listServiceSas
Microsoft.StorSimple/managers/devices	listFailoverSets
Microsoft.StorSimple/managers/devices	listFailoverTargets
Microsoft.StorSimple/managers	listActivationKey
Microsoft.StorSimple/managers	listPublicEncryptionKey
Microsoft.Synapse/workspaces/integrationRuntimes	listAuthKeys
Microsoft.Web/connectionGateways	ListStatus
microsoft.web/connections	listconsentlinks
Microsoft.Web/customApis	listWsdlInterfaces
microsoft.web/locations	listwsdlinterfaces
microsoft.web/apimanagementaccounts/apis/connections	listconnectionkeys
microsoft.web/apimanagementaccounts/apis/connections	listsecrets
microsoft.web/sites/backups	list
Microsoft.Web/sites/config	list
microsoft.web/sites/functions	listkeys
microsoft.web/sites/functions	listsecrets
microsoft.web/sites/hybridconnectionnamespaces/relays	listkeys
microsoft.web/sites	listsyncfunctiontriggerstatus

RESOURCE TYPE	FUNCTION NAME
microsoft.web/sites/slots/functions	listsecrets
microsoft.web/sites/slots/backups	list
Microsoft.Web/sites/slots/config	list
microsoft.web/sites/slots/functions	listsecrets

To determine which resource types have a list operation, you have the following options:

- View the REST API operations for a resource provider, and look for list operations. For example, storage accounts have the listKeys operation.
- Use the Get-AzProviderOperation PowerShell cmdlet. The following example gets all list operations for storage accounts:

```
Get-AzProviderOperation -OperationSearchString "Microsoft.Storage/*" | where {$_.Operation -like
"*list*"} | FT Operation
```

- Use the following Azure CLI command to filter only the list operations:

```
az provider operation show --namespace Microsoft.Storage --query "resourceTypes[?
name=='storageAccounts'].operations[].name | [?contains(@, 'list')]"
```

pickZones

`pickZones(providerNamespace, resourceType, location, [numberOfZones], [offset])`

Determines whether a resource type supports zones for a region. This function **only supports zonal resources**. Zone redundant services return an empty array. For more information, see Azure Services that support Availability Zones.

Namespace: az.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
providerNamespace	Yes	string	The resource provider namespace for the resource type to check for zone support.
resourceType	Yes	string	The resource type to check for zone support.
location	Yes	string	The region to check for zone support.

PARAMETER	REQUIRED	TYPE	DESCRIPTION
numberOfZones	No	integer	The number of logical zones to return. The default is 1. The number must be a positive integer from 1 to 3. Use 1 for single-zoned resources. For multi-zoned resources, the value must be less than or equal to the number of supported zones.
offset	No	integer	The offset from the starting logical zone. The function returns an error if offset plus numberOfZones exceeds the number of supported zones.

Return value

An array with the supported zones. When using the default values for offset and `numberOfZones`, a resource type and region that supports zones returns the following array:

```
[
    "1"
]
```

When the `numberOfZones` parameter is set to 3, it returns:

```
[
    "1",
    "2",
    "3"
]
```

When the resource type or region doesn't support zones, an empty array is returned.

```
[
]
```

Remarks

There are different categories for Azure Availability Zones - zonal and zone-redundant. The `pickZones` function can be used to return an availability zone for a zonal resource. For zone redundant services (ZRS), the function returns an empty array. Zonal resources typically have a `zones` property at the top level of the resource definition. To determine the category of support for availability zones, see Azure Services that support Availability Zones.

To determine if a given Azure region or location supports availability zones, call the `pickZones` function with a zonal resource type, such as `Microsoft.Network/publicIPAddresses`. If the response isn't empty, the region supports availability zones.

pickZones example

The following Bicep file shows three results for using the `pickZones` function.

```
output supported array = pickZones('Microsoft.Compute', 'virtualMachines', 'westus2')
output notSupportedRegion array = pickZones('Microsoft.Compute', 'virtualMachines', 'westus')
output notSupportedType array = pickZones('Microsoft.Cdn', 'profiles', 'westus2')
```

The output from the preceding examples returns three arrays.

NAME	TYPE	VALUE
supported	array	["1"]
notSupportedRegion	array	[]
notSupportedType	array	[]

You can use the response from `pickZones` to determine whether to provide null for zones or assign virtual machines to different zones.

providers

The providers function has been deprecated in Bicep. We no longer recommend using it. If you used this function to get an API version for the resource provider, we recommend that you provide a specific API version in your Bicep file. Using a dynamically returned API version can break your template if the properties change between versions.

The providers operation is still available through the REST API. It can be used outside of a Bicep file to get information about a resource provider.

Namespace: az.

reference

`reference(resourceName or resourceIdentifier, [apiVersion], ['Full'])`

Returns an object representing a resource's runtime state.

Namespace: az.

The reference function is available in Bicep files, but typically you don't need it. Instead, use the symbolic name for the resource.

The following example deploys a storage account. It uses the symbolic name `storageAccount` for the storage account to return a property.

```
param storageAccountName string

resource storageAccount 'Microsoft.Storage/storageAccounts@2019-06-01' = {
  name: storageAccountName
  location: 'eastus'
  kind: 'Storage'
  sku: {
    name: 'Standard_LRS'
  }
}

output storageEndpoint object = storageAccount.properties.primaryEndpoints
```

To get a property from an existing resource that isn't deployed in the template, use the `existing` keyword:

```
param storageAccountName string

resource storageAccount 'Microsoft.Storage/storageAccounts@2019-06-01' existing = {
  name: storageAccountName
}

// use later in template as often as needed
output blobAddress string = storageAccount.properties.primaryEndpoints.blob
```

To reference a resource that is nested inside a parent resource, use the nested accessor (`::`). You only use this syntax when you're accessing the nested resource from outside of the parent resource.

```
vNet1::subnet1.properties.addressPrefix
```

If you attempt to reference a resource that doesn't exist, you get the `NotFound` error and your deployment fails.

resourceId

`resourceId([subscriptionId], [resourceGroupName], resourceType, resourceName1, [resourceName2], ...)`

Returns the unique identifier of a resource.

Namespace: az.

The `resourceId` function is available in Bicep files, but typically you don't need it. Instead, use the symbolic name for the resource and access the `id` property.

You use this function when the resource name is ambiguous or not provisioned within the same Bicep file. The format of the returned identifier varies based on whether the deployment happens at the scope of a resource group, subscription, management group, or tenant.

For example:

```
param storageAccountName string

resource storageAccount 'Microsoft.Storage/storageAccounts@2019-06-01' = {
  name: storageAccountName
  location: 'eastus'
  kind: 'Storage'
  sku: {
    name: 'Standard_LRS'
  }
}

output storageID string = storageAccount.id
```

To get the resource ID for a resource that isn't deployed in the Bicep file, use the existing keyword.

```
param storageAccountName string

resource storageAccount 'Microsoft.Storage/storageAccounts@2019-06-01' existing = {
    name: storageAccountName
}

output storageID string = storageAccount.id
```

For more information, see the JSON template resourceId function

subscriptionResourceId

`subscriptionResourceId([subscriptionId], resourceType, resourceName1, [resourceName2], ...)`

Returns the unique identifier for a resource deployed at the subscription level.

Namespace: az.

The `subscriptionResourceId` function is available in Bicep files, but typically you don't need it. Instead, use the symbolic name for the resource and access the `id` property.

The identifier is returned in the following format:

```
/subscriptions/{subscriptionId}/providers/{resourceProviderNamespace}/{resourceType}/{resourceName}
```

Remarks

You use this function to get the resource ID for resources that are deployed to the subscription rather than a resource group. The returned ID differs from the value returned by the resourceId function by not including a resource group value.

subscriptionResourceID example

The following Bicep file assigns a built-in role. You can deploy it to either a resource group or subscription. It uses the `subscriptionResourceId` function to get the resource ID for built-in roles.

```
@description('Principal Id')
param principalId string

@allowed([
  'Owner'
  'Contributor'
  'Reader'
])
@description('Built-in role to assign')
param builtInRoleType string

var roleDefinitionId = {
  Owner: {
    id: subscriptionResourceId('Microsoft.Authorization/roleDefinitions', '8e3af657-a8ff-443c-a75c-
2fe8c4bcb635')
  }
  Contributor: {
    id: subscriptionResourceId('Microsoft.Authorization/roleDefinitions', 'b24988ac-6180-42a0-ab88-
20f7382dd24c')
  }
  Reader: {
    id: subscriptionResourceId('Microsoft.Authorization/roleDefinitions', 'acdd72a7-3385-48ef-bd42-
f606fba81ae7')
  }
}

resource roleAssignment 'Microsoft.Authorization/roleAssignments@2018-09-01-preview' = {
  name: guid(resourceGroup().id, principalId, roleDefinitionId[builtInRoleType].id)
  properties: {
    roleDefinitionId: roleDefinitionId[builtInRoleType].id
    principalId: principalId
  }
}
```

managementGroupResourceId

`managementGroupResourceId(resourceType, resourceName1, [resourceName2], ...)`

Returns the unique identifier for a resource deployed at the management group level.

Namespace: az.

The `managementGroupResourceId` function is available in Bicep files, but typically you don't need it. Instead, use the symbolic name for the resource and access the `id` property.

The identifier is returned in the following format:

```
/providers/Microsoft.Management/managementGroups/{managementGroupName}/providers/{resourceType}/{resourceNam
e}
```

Remarks

You use this function to get the resource ID for resources that are deployed to the management group rather than a resource group. The returned ID differs from the value returned by the resourceId function by not including a subscription ID and a resource group value.

managementGroupResourceID example

The following template creates and assigns a policy definition. It uses the `managementGroupResourceId` function to get the resource ID for policy definition.

```
targetScope = 'managementGroup'

@description('Target Management Group')
param targetMG string

@description('An array of the allowed locations, all other locations will be denied by the created policy.')
param allowedLocations array = [
  'australiaeast'
  'australiasoutheast'
  'australiacentral'
]

var mgScope = tenantResourceId('Microsoft.Management/managementGroups', targetMG)
var policyDefinitionName = 'LocationRestriction'

resource policyDefinition 'Microsoft.Authorization/policyDefinitions@2020-03-01' = {
  name: policyDefinitionName
  properties: {
    policyType: 'Custom'
    mode: 'All'
    parameters: {}
    policyRule: {
      if: {
        not: {
          field: 'location'
          in: allowedLocations
        }
      }
      then: {
        effect: 'deny'
      }
    }
  }
}

resource location_lock 'Microsoft.Authorization/policyAssignments@2020-03-01' = {
  name: 'location-lock'
  properties: {
    scope: mgScope
    policyDefinitionId: managementGroupResourceId('Microsoft.Authorization/policyDefinitions',
policyDefinitionName)
  }
  dependsOn: [
    policyDefinition
  ]
}
```

tenantResourceId

`tenantResourceId(resourceType, resourceName1, [resourceName2], ...)`

Returns the unique identifier for a resource deployed at the tenant level.

Namespace: az.

The `tenantResourceId` function is available in Bicep files, but typically you don't need it. Instead, use the symbolic name for the resource and access the `id` property.

The identifier is returned in the following format:

```
/providers/{resourceProviderNamespace}/{resourceType}/{resourceName}
```

Built-in policy definitions are tenant level resources. To deploy a policy assignment that references a built-in

policy definition, use the `tenantResourceId` function.

```
@description('Specifies the ID of the policy definition or policy set definition being assigned.')
param policyDefinitionID string = '0a914e76-4921-4c19-b460-a2d36003525a'

@description('Specifies the name of the policy assignment, can be used defined or an idempotent name as the
defaultValue provides.')
param policyAssignmentName string = guid(policyDefinitionID, resourceGroup().name)

resource policyAssignment 'Microsoft.Authorization/policyAssignments@2019-09-01' = {
  name: policyAssignmentName
  properties: {
    scope: subscriptionResourceId('Microsoft.Resources/resourceGroups', resourceGroup().name)
    policyDefinitionId: tenantResourceId('Microsoft.Authorization/policyDefinitions', policyDefinitionID)
  }
}
```

Next steps

- To get values from the current deployment, see Deployment value functions.
- To iterate a specified number of times when creating a type of resource, see Iterative loops in Bicep.

Scope functions for Bicep

9/21/2022 • 5 minutes to read • Edit Online

This article describes the Bicep functions for getting scope values.

managementGroup

`managementGroup()`

Returns an object with properties from the management group in the current deployment.

`managementGroup(identifier)`

Returns an object used for setting the scope to a management group.

Namespace: az.

Remarks

`managementGroup()` can only be used on a management group deployments. It returns the current management group for the deployment operation. Use when either getting a scope object or getting properties for the current management group.

`managementGroup(identifier)` can be used for any deployment scope, but only when getting the scope object. To retrieve the properties for a management group, you can't pass in the management group identifier.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
identifier	No	string	The unique identifier for the management group to deploy to. Don't use the display name for the management group. If you don't provide a value, the current management group is returned.

Return value

An object used for setting the `scope` property on a module or extension resource type. Or, an object with the properties for the current management group.

Management group example

The following example sets the scope for a module to a management group.

```
param managementGroupIdentifier string

module  'mgModule.bicep' = {
  name: 'deployToMG'
  scope: managementGroup(managementGroupIdentifier)
}
```

The next example returns properties for the current management group.

```
targetScope = 'managementGroup'

var mgInfo = managementGroup()

output mgResult object = mgInfo
```

It returns:

```
"mgResult": {
  "type": "Object",
  "value": {
    "id": "/providers/Microsoft.Management/managementGroups/examplemg1",
    "name": "examplemg1",
    "properties": {
      "details": {
        "parent": {
          "displayName": "Tenant Root Group",
          "id": "/providers/Microsoft.Management/managementGroups/00000000-0000-0000-0000-000000000000",
          "name": "00000000-0000-0000-0000-000000000000"
        },
        "updatedBy": "00000000-0000-0000-0000-000000000000",
        "updatedTime": "2020-07-23T21:05:52.661306Z",
        "version": "1"
      },
      "displayName": "Example MG 1",
      "tenantId": "00000000-0000-0000-0000-000000000000"
    },
    "type": "/providers/Microsoft.Management/managementGroups"
  }
}
```

The next example creates a new management group and uses this function to set the parent management group.

```
targetScope = 'managementGroup'

param mgName string = 'mg-${uniqueString(newGuid())}'

resource newMG 'Microsoft.Management/managementGroups@2020-05-01' = {
  scope: tenant()
  name: mgName
  properties: {
    details: {
      parent: {
        id: managementGroup().id
      }
    }
  }
}

output newManagementGroup string = mgName
```

resourceGroup

`resourceGroup()`

Returns an object that represents the current resource group.

`resourceGroup(resourceGroupName)`

And

`resourceGroup(subscriptionId, resourceGroupName)`

Return an object used for setting the scope to a resource group.

Namespace: az.

Remarks

The resourceGroup function has two distinct uses. One usage is for setting the scope on a module or extension resource type. The other usage is for getting details about the current resource group. The placement of the function determines its usage. When used to set the `scope` property, it returns a scope object.

`resourceGroup()` can be used for either setting scope or getting details about the resource group.

`resourceGroup(resourceGroupName)` and `resourceGroup(subscriptionId, resourceGroupName)` can only be used for setting scope.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
resourceGroupName	No	string	The name of the resource group to deploy to. If you don't provide a value, the current resource group is returned.
subscriptionId	No	string	The unique identifier for the subscription to deploy to. If you don't provide a value, the current subscription is returned.

Return value

When used for setting scope, the function returns an object that is valid for the `scope` property on a module or extension resource type.

When used for getting details about the resource group, the function returns the following format:

```
{
  "id": "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}",
  "name": "{resourceGroupName}",
  "type":"Microsoft.Resources/resourceGroups",
  "location": "{resourceGroupLocation}",
  "managedBy": "{identifier-of-managing-resource}",
  "tags": {
  },
  "properties": {
    "provisioningState": "{status}"
  }
}
```

The **managedBy** property is returned only for resource groups that contain resources that are managed by another service. For Managed Applications, Databricks, and AKS, the value of the property is the resource ID of the managing resource.

Resource group example

The following example scopes a module to a resource group.

```
param resourceGroupName string

module exampleModule 'rgModule.bicep' = {
  name: 'exampleModule'
  scope: resourceGroup(resourceGroupName)
}
```

The next example returns the properties of the resource group.

```
output resourceGroupOutput object = resourceGroup()
```

It returns an object in the following format:

```
{
  "id": "/subscriptions/{subscription-id}/resourceGroups/examplegroup",
  "name": "examplegroup",
  "type":"Microsoft.Resources/resourceGroups",
  "location": "southcentralus",
  "properties": {
    "provisioningState": "Succeeded"
  }
}
```

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 for a default parameter value.

```
param location string = resourceGroup().location
```

You can also use the resourceGroup function to apply tags from the resource group to a resource. For more information, see Apply tags from resource group.

subscription

`subscription()`

Returns details about the subscription for the current deployment.

`subscription(subscriptionId)`

Returns an object used for setting the scope to a subscription.

Namespace: az.

Remarks

The subscription function has two distinct uses. One usage is for setting the scope on a module or extension resource type. The other usage is for getting details about the current subscription. The placement of the function determines its usage. When used to set the `scope` property, it returns a scope object.

`subscription(subscriptionId)` can only be used for setting scope.

`subscription()` can be used for setting scope or getting details about the subscription.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
subscriptionId	No	string	The unique identifier for the subscription to deploy to. If you don't provide a value, the current subscription is returned.

Return value

When used for setting scope, the function returns an object that is valid for the `scope` property on a module or extension resource type.

When used for getting details about the subscription, the function returns the following format:

```
{
  "id": "/subscriptions/{subscription-id}",
  "subscriptionId": "{subscription-id}",
  "tenantId": "{tenant-id}",
  "displayName": "{name-of-subscription}"
}
```

Subscription example

The following example scopes a module to the subscription.

```
module exampleModule 'subModule.bicep' = {
  name: 'deployToSub'
  scope: subscription()
}
```

The next example returns the details for a subscription.

```
output subscriptionOutput object = subscription()
```

tenant

`tenant()`

Returns an object used for setting the scope to the tenant.

Or

Returns properties about the tenant for the current deployment.

Namespace: az.

Remarks

`tenant()` can be used with any deployment scope. It always returns the current tenant. You can use this function to set the scope for a resource, or to get properties for the current tenant.

Return value

An object used for setting the `scope` property on a module or extension resource type. Or, an object with properties about the current tenant.

Tenant example

The following example shows a module deployed to the tenant.

```
module exampleModule 'tenantModule.bicep' = {
  name: 'deployToTenant'
  scope: tenant()
}
```

The next example returns the properties for a tenant.

```
var tenantInfo = tenant()

output tenantResult object = tenantInfo
```

It returns:

```
"tenantResult": {
  "type": "Object",
  "value": {
    "countryCode": "US",
    "displayName": "Contoso",
    "id": "/tenants/00000000-0000-0000-0000-000000000000",
    "tenantId": "00000000-0000-0000-0000-000000000000"
  }
}
```

Some resources require setting the tenant ID for a property. Rather than passing the tenant ID as a parameter, you can retrieve it with the tenant function.

```
resource kv 'Microsoft.KeyVault/vaults@2021-06-01-preview' = {
  name: 'examplekeyvault'
  location: 'westus'
  properties: {
    tenantId: tenant().tenantId
    ...
  }
}
```

Next steps

To learn more about deployment scopes, see:

- Resource group deployments
- Subscription deployments
- Management group deployments
- Tenant deployments

String functions for Bicep

9/21/2022 • 24 minutes to read • Edit Online

This article describes the Bicep functions for working with strings.

base64

`base64(inputString)`

Returns the base64 representation of the input string.

Namespace: sys.

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 shows how to use the base64 function.

```
param stringData string = 'one, two, three'
param jsonFormattedData string = '{\'one\': \'a\', \'two\': \'b\'}'

var base64String = base64(stringData)
var base64Object = base64(jsonFormattedData)

output base64Output string = base64String
output toStringOutput string = base64ToString(base64String)
output toJsonOutput object = base64ToJson(base64Object)
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
base64Output	String	b25lLCB0d28sIHRocmVl
toStringOutput	String	one, two, three
toJsonOutput	Object	{"one": "a", "two": "b"}

base64ToJson

`base64ToJson(base64Value)`

Converts a base64 representation to a JSON object.

Namespace: sys.

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 uses the base64ToJson function to convert a base64 value:

```
param stringData string = 'one, two, three'
param jsonFormattedData string = '{\'one\': \'a\', \'two\': \'b\'}'

var base64String = base64(stringData)
var base64Object = base64(jsonFormattedData)

output base64Output string = base64String
output toStringOutput string = base64ToString(base64String)
output toJsonOutput object = base64ToJson(base64Object)
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
base64Output	String	b25lLCB0d28sIHRocmVl
toStringOutput	String	one, two, three
toJsonOutput	Object	{"one": "a", "two": "b"}

base64ToString

`base64ToString(base64Value)`

Converts a base64 representation to a string.

Namespace: sys.

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 uses the base64ToString function to convert a base64 value:

```
param stringData string = 'one, two, three'
param jsonFormattedData string = '{\'one\': \'a\', \'two\': \'b\'}'

var base64String = base64(stringData)
var base64Object = base64(jsonFormattedData)

output base64Output string = base64String
output toStringOutput string = base64ToString(base64String)
output toJsonOutput object = base64ToJson(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"}

concat

Instead of using the concat function, use string interpolation.

```
param prefix string = 'prefix'

output concatOutput string = '${prefix}And${uniqueString(resourceGroup().id)}'
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
concatOutput	String	prefixAnd5yj4yjf5mbg72

Namespace: sys.

contains

`contains(container, itemToFind)`

Checks whether an array contains a value, an object contains a key, or a string contains a substring. The string comparison is case-sensitive. However, when testing if an object contains a key, the comparison is case-insensitive.

Namespace: sys.

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 shows how to use contains with different types:

```
param stringToTest string = 'OneTwoThree'
param objectToTest object = {
  'one': 'a'
  'two': 'b'
  'three': 'c'
}
param arrayToTest array = [
  'one'
  'two'
  'three'
]

output stringTrue bool = contains(stringToTest, 'e')
output stringFalse bool = contains(stringToTest, 'z')
output objectTrue bool = contains(objectToTest, 'one')
output objectFalse bool = contains(objectToTest, 'a')
output arrayTrue bool = contains(arrayToTest, 'three')
output arrayFalse bool = contains(arrayToTest, 'four')
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
stringTrue	Bool	True
stringFalse	Bool	False

NAME	TYPE	VALUE
objectTrue	Bool	True
objectFalse	Bool	False
arrayTrue	Bool	True
arrayFalse	Bool	False

dataUri

`dataUri(stringToConvert)`

Converts a value to a data URI.

Namespace: sys.

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 converts a value to a data URI, and converts a data URI to a string:

```
param stringToTest string = 'Hello'
param dataFormattedString string = 'data:; base64,SGVsbG8sIFdvcmxkIQ=='

output dataUriOutput string = dataUri(stringToTest)
output toStringOutput string = dataUriToString(dataFormattedString)
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
dataUriOutput	String	data:text/plain;charset=utf8; base64,SGVsbG8=
toStringOutput	String	Hello, World!

dataUriToString

`dataUriToString(dataUriToConvert)`

Converts a data URI formatted value to a string.

Namespace: sys.

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 converts a value to a data URI, and converts a data URI to a string:

```
param stringToTest string = 'Hello'
param dataFormattedString string = 'data:; base64,SGVsbG8sIFdvcmxkIQ=='

output dataUriOutput string = dataUri(stringToTest)
output toStringOutput string = dataUriToString(dataFormattedString)
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
dataUriOutput	String	data:text/plain;charset=utf8; base64,SGVsbG8=
toStringOutput	String	Hello, World!

empty

`empty(itemToTest)`

Determines if an array, object, or string is empty.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
itemToTest	Yes	array, object, or string	The value to check if it's empty.

Return value

Returns **True** if the value is empty; otherwise, **False**.

Examples

The following example checks whether an array, object, and string are empty.

```
param testArray array = []
param testObject object = {}
param testString string = ''

output arrayEmpty bool = empty(testArray)
output objectEmpty bool = empty(testObject)
output stringEmpty bool = empty(testString)
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayEmpty	Bool	True
objectEmpty	Bool	True
stringEmpty	Bool	True

endsWith

`endsWith(stringToSearch, stringToFind)`

Determines whether a string ends with a value. The comparison is case-insensitive.

Namespace: sys.

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 shows how to use the startsWith and endsWith functions:

```
output startsTrue bool = startsWith('abcdef', 'ab')
output startsCapTrue bool = startsWith('abcdef', 'A')
output startsFalse bool = startsWith('abcdef', 'e')
output endsTrue bool = endsWith('abcdef', 'ef')
output endsCapTrue bool = endsWith('abcdef', 'F')
output endsFalse bool = endsWith('abcdef', 'e')
```

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

first

`first(arg1)`

Returns the first character of the string, or first element of the array.

Namespace: sys.

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 shows how to use the first function with an array and string.

```
param arrayToTest array = [
  'one'
  'two'
  'three'
]

output arrayOutput string = first(arrayToTest)
output stringOutput string = first('One Two Three')
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayOutput	String	one
stringOutput	String	O

format

`format(formatString, arg1, arg2, ...)`

Creates a formatted string from input values.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
formatString	Yes	string	The composite format string.
arg1	Yes	string, integer, or boolean	The value to include in the formatted string.
additional arguments	No	string, integer, or boolean	Additional values to include in the formatted string.

Remarks

Use this function to format a string in your Bicep file. It uses the same formatting options as the System.String.Format method in .NET.

Examples

The following example shows how to use the format function.

```
param greeting string = 'Hello'
param name string = 'User'
param numberToFormat int = 8175133

output formatTest string = format('{0}, {1}. Formatted number: {2:N0}', greeting, name, numberToFormat)
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
formatTest	String	Hello, User. Formatted number: 8,175,133

guid

`guid(baseString, ...)`

Creates a value in the format of a globally unique identifier based on the values provided as parameters.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
baseString	Yes	string	The value used in the hash function to create the GUID.

PARAMETER	REQUIRED	TYPE	DESCRIPTION
additional parameters as needed	No	string	You can add as many strings as needed to create the value that specifies the level of uniqueness.

Remarks

This function is helpful when you need to create a value in the format of a globally unique identifier. You provide parameter values that limit the scope of uniqueness for the result. You can specify whether the name is unique down to subscription, resource group, or deployment.

The returned value isn't a random string, but rather the result of a hash function on the parameters. The returned value is 36 characters long. It isn't globally unique. To create a new GUID that isn't based on that hash value of the parameters, use the newGuid function.

> **NOTE**
>
> The order of the parameters affects the returned value. For example:
>
> `guid('hello', 'world')` and `guid('world', 'hello')`
>
> don't return the same value.

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 returns results from guid:

```
output guidPerSubscription string = guid(subscription().subscriptionId)
output guidPerResourceGroup string = guid(resourceGroup().id)
output guidPerDeployment string = guid(resourceGroup().id, deployment().name)
```

indexOf

`indexOf(stringToSearch, stringToFind)`

Returns the first position of a value within a string. The comparison is case-insensitive.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
stringToSearch	Yes	string	The value that contains the item to find.

PARAMETER	REQUIRED	TYPE	DESCRIPTION
stringToFind	Yes	string	The value to find.

Return value

An integer that represents the position of the item to find. The value is zero-based. If the item isn't found, -1 is returned.

Examples

The following example shows how to use the indexOf and lastIndexOf functions:

```
output firstT int = indexOf('test', 't')
output lastT int = lastIndexOf('test', 't')
output firstString int = indexOf('abcdef', 'CD')
output lastString int = lastIndexOf('abcdef', 'AB')
output notFound int = indexOf('abcdef', 'z')
```

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

join

`join(inputArray, delimiter)`

Joins a string array into a single string, separated using a delimiter.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
inputArray	Yes	An array of string.	An array of strings to join.
delimiter	Yes	The delimiter to use for splitting the string.	

Return value

A string.

Examples

The following example joins the input string array into strings delimited by either a comma or a semi-colon.

```
var arrayString = [
  'one'
  'two'
  'three'
]

output firstOutput string = join(arrayString, ',')
output secondOutput string = join(arrayString, ';')
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
firstOutput	String	"one,two,three"
secondOutput	String	"one;two;three"

This function requires **Bicep version 0.8.2 or later**.

json

`json(arg1)`

Converts a valid JSON string into a JSON data type. For more information, see json function.

Namespace: sys.

last

`last(arg1)`

Returns last character of the string, or the last element of the array.

Namespace: sys.

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 shows how to use the last function with an array and string.

```
param arrayToTest array = [
  'one'
  'two'
  'three'
]

output arrayOutput string = last(arrayToTest)
output stringOutput string = last('One Two Three')
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayOutput	String	three
stringOutput	String	e

lastIndexOf

`lastIndexOf(stringToSearch, stringToFind)`

Returns the last position of a value within a string. The comparison is case-insensitive.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
stringToSearch	Yes	string	The value that contains the item to find.
stringToFind	Yes	string	The value to find.

Return value

An integer that represents the last position of the item to find. The value is zero-based. If the item isn't found, -1 is returned.

Examples

The following example shows how to use the `indexOf` and `lastIndexOf` functions:

```
output firstT int = indexOf('test', 't')
output lastT int = lastIndexOf('test', 't')
output firstString int = indexOf('abcdef', 'CD')
output lastString int = lastIndexOf('abcdef', 'AB')
output notFound int = indexOf('abcdef', 'z')
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
firstT	Int	0
lastT	Int	3
firstString	Int	2
lastString	Int	0
notFound	Int	-1

length

`length(string)`

Returns the number of characters in a string, elements in an array, or root-level properties in an object.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
arg1	Yes	array, string, or object	The array to use for getting the number of elements, the string to use for getting the number of characters, or the object to use for getting the number of root-level properties.

Return value

An int.

Examples

The following example shows how to use length with an array and string:

```
param arrayToTest array = [
  'one'
  'two'
  'three'
]
param stringToTest string = 'One Two Three'
param objectToTest object = {
  'propA': 'one'
  'propB': 'two'
  'propC': 'three'
  'propD': {
    'propD-1': 'sub'
    'propD-2': 'sub'
  }
}

output arrayLength int = length(arrayToTest)
output stringLength int = length(stringToTest)
output objectLength int = length(objectToTest)
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayLength	Int	3
stringLength	Int	13
objectLength	Int	4

newGuid

`newGuid()`

Returns a value in the format of a globally unique identifier. **This function can only be used in the default value for a parameter.**

Namespace: sys.

Remarks

You can only use this function within an expression for the default value of a parameter. Using this function anywhere else in a Bicep file returns an error. The function isn't allowed in other parts of the Bicep file because it returns a different value each time it's called. Deploying the same Bicep file with the same parameters wouldn't reliably produce the same results.

The newGuid function differs from the guid function because it doesn't take any parameters. When you call guid with the same parameters, it returns the same identifier each time. Use guid when you need to reliably generate the same GUID for a specific environment. Use newGuid when you need a different identifier each time, such as deploying resources to a test environment.

The newGuid function uses the Guid structure in the .NET Framework to generate the globally unique identifier.

If you use the option to redeploy an earlier successful deployment, and the earlier deployment includes a parameter that uses newGuid, the parameter isn't reevaluated. Instead, the parameter value from the earlier deployment is automatically reused in the rollback deployment.

In a test environment, you may need to repeatedly deploy resources that only live for a short time. Rather than constructing unique names, you can use newGuid with uniqueString to create unique names.

Be careful redeploying a Bicep file that relies on the newGuid function for a default value. When you redeploy and don't provide a value for the parameter, the function is reevaluated. If you want to update an existing resource rather than create a new one, pass in the parameter value from the earlier deployment.

Return value

A string containing 36 characters in the format of a globally unique identifier.

Examples

The following example shows a parameter with a new identifier.

```
param guidValue string = newGuid()

output guidOutput string = guidValue
```

The output from the preceding example varies for each deployment but will be similar to:

NAME	TYPE	VALUE
guidOutput	string	b76a51fc-bd72-4a77-b9a2-3c29e7d2e551

The following example uses the newGuid function to create a unique name for a storage account. This Bicep file might work for test environment where the storage account exists for a short time and isn't redeployed.

```
param guidValue string = newGuid()

var storageName = 'storage${uniqueString(guidValue)}'

resource myStorage 'Microsoft.Storage/storageAccounts@2018-07-01' = {
  name: storageName
  location: 'West US'
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
  properties: {}
}

output nameOutput string = storageName
```

The output from the preceding example varies for each deployment but will be similar to:

NAME	TYPE	VALUE
nameOutput	string	storagenziwvyru7uxie

padLeft

`padLeft(valueToPad, totalLength, paddingCharacter)`

Returns a right-aligned string by adding characters to the left until reaching the total specified length.

Namespace: sys.

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 shows how to pad the user-provided parameter value by adding the zero character until it reaches the total number of characters.

```
param testString string = '123'

output stringOutput string = padLeft(testString, 10, '0')
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
stringOutput	String	0000000123

replace

`replace(originalString, oldString, newString)`

Returns a new string with all instances of one string replaced by another string.

Namespace: sys.

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 shows how to remove all dashes from the user-provided string, and how to replace part of the string with another string.

```
param testString string = '123-123-1234'

output firstOutput string = replace(testString, '-', '')
output secondOutput string = replace(testString, '1234', 'xxxx')
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
firstOutput	String	1231231234
secondOutput	String	123-123-xxxx

skip

`skip(originalValue, numberToSkip)`

Returns a string with all the characters after the specified number of characters, or an array with all the elements after the specified number of elements.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
originalValue	Yes	array or string	The array or string to use for skipping.
numberToSkip	Yes	int	The number of elements or characters to skip. If this value is 0 or less, all the elements or characters in the value are returned. If it's larger than the length of the array or string, an empty array or string is returned.

Return value

An array or string.

Examples

The following example skips the specified number of elements in the array, and the specified number of characters in a string.

```
param testArray array = [
  'one'
  'two'
  'three'
]
param elementsToSkip int = 2
param testString string = 'one two three'
param charactersToSkip int = 4

output arrayOutput array = skip(testArray, elementsToSkip)
output stringOutput string = skip(testString, charactersToSkip)
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayOutput	Array	["three"]
stringOutput	String	two three

split

`split(inputString, delimiter)`

Returns an array of strings that contains the substrings of the input string that are delimited by the specified delimiters.

Namespace: sys.

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 splits the input string with a comma, and with either a comma or a semi-colon.

```
param firstString string = 'one,two,three'
param secondString string = 'one;two,three'

var delimiters = [
  ','
  ';'
]

output firstOutput array = split(firstString, ',')
output secondOutput array = split(secondString, delimiters)
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
firstOutput	Array	["one", "two", "three"]
secondOutput	Array	["one", "two", "three"]

startsWith

`startsWith(stringToSearch, stringToFind)`

Determines whether a string starts with a value. The comparison is case-insensitive.

Namespace: sys.

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 shows how to use the startsWith and endsWith functions:

```
output startsTrue bool = startsWith('abcdef', 'ab')
output startsCapTrue bool = startsWith('abcdef', 'A')
output startsFalse bool = startsWith('abcdef', 'e')
output endsTrue bool = endsWith('abcdef', 'ef')
output endsCapTrue bool = endsWith('abcdef', 'F')
output endsFalse bool = endsWith('abcdef', 'e')
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
startsTrue	Bool	True
startsCapTrue	Bool	True
startsFalse	Bool	False
endsTrue	Bool	True
endsCapTrue	Bool	True
endsFalse	Bool	False

string

`string(valueToConvert)`

Converts the specified value to a string.

Namespace: sys.

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 shows how to convert different types of values to strings:

```
param testObject object = {
  'valueA': 10
  'valueB': 'Example Text'
}
param testArray array = [
  'a'
  'b'
  'c'
]
param testInt int = 5

output objectOutput string = string(testObject)
output arrayOutput string = string(testArray)
output intOutput string = string(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

substring

`substring(stringToParse, startIndex, length)`

Returns a substring that starts at the specified character position and contains the specified number of characters.

Namespace: sys.

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.

PARAMETER	REQUIRED	TYPE	DESCRIPTION
length	No	int	The number of characters for the substring. Must refer to a location within the string. Must be zero or greater. If omitted, the remainder of the string from the start position will be returned.

Return value

The substring. Or, an empty string if the length is zero.

Remarks

The function fails when the substring extends beyond the end of the string, or when length is less than zero. The following example fails with the error "The index and length parameters must refer to a location within the string. The index parameter: '0', the length parameter: '11', the length of the string parameter: '10'.".

```
param inputString string = '1234567890'

var prefix = substring(inputString, 0, 11)
```

Examples

The following example extracts a substring from a parameter.

```
param testString string = 'one two three'
output substringOutput string = substring(testString, 4, 3)
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
substringOutput	String	two

take

`take(originalValue, numberToTake)`

Returns a string with the specified number of characters from the start of the string, or an array with the specified number of elements from the start of the array.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
originalValue	Yes	array or string	The array or string to take the elements from.
numberToTake	Yes	int	The number of elements or characters to take. If this value is 0 or less, an empty array or string is returned. If it's larger than the length of the given array or string, all the elements in the array or string are returned.

Return value

An array or string.

Examples

The following example takes the specified number of elements from the array, and characters from a string.

```
param testArray array = [
  'one'
  'two'
  'three'
]
param elementsToTake int = 2
param testString string = 'one two three'
param charactersToTake int = 2

output arrayOutput array = take(testArray, elementsToTake)
output stringOutput string = take(testString, charactersToTake)
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
arrayOutput	Array	["one", "two"]
stringOutput	String	on

toLower

`toLower(stringToChange)`

Converts the specified string to lower case.

Namespace: sys.

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 converts a parameter value to lower case and to upper case.

```
param testString string = 'One Two Three'

output toLowerOutput string = toLower(testString)
output toUpperOutput string = toUpper(testString)
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
toLowerOutput	String	one two three
toUpperOutput	String	ONE TWO THREE

toUpper

`toUpper(stringToChange)`

Converts the specified string to upper case.

Namespace: sys.

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 converts a parameter value to lower case and to upper case.

```
param testString string = 'One Two Three'

output toLowerOutput string = toLower(testString)
output toUpperOutput string = toUpper(testString)
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
toLowerOutput	String	one two three
toUpperOutput	String	ONE TWO THREE

trim

`trim(stringToTrim)`

Removes all leading and trailing white-space characters from the specified string.

Namespace: sys.

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 trims the white-space characters from the parameter.

```
param testString string = '    one two three   '

output return string = trim(testString)
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
return	String	one two three

uniqueString

`uniqueString(baseString, ...)`

Creates a deterministic hash string based on the values provided as parameters.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
baseString	Yes	string	The value used in the hash function to create a unique string.
additional parameters as needed	No	string	You can add as many strings as needed to create the value that specifies the level of uniqueness.

Remarks

This function is helpful when you need to create a unique name for a resource. You provide parameter values that limit the scope of uniqueness for the result. You can specify whether the name is unique down to subscription, resource group, or deployment.

The returned value isn't a random string, but rather the result of a hash function. The returned value is 13 characters long. It isn't globally unique. You may want to combine the value with a prefix from your naming convention to create a name that is meaningful. The following example shows the format of the returned value. The actual value varies by the provided parameters.

`tcvhiyu5h2o5o`

The following examples show how to use uniqueString to create a unique value for commonly used levels.

Unique scoped to subscription

```
uniqueString(subscription().subscriptionId)
```

Unique scoped to resource group

```
uniqueString(resourceGroup().id)
```

Unique scoped to deployment for a resource group

```
uniqueString(resourceGroup().id, deployment().name)
```

The following example shows how to create a unique name for a storage account based on your resource group. Inside the resource group, the name isn't unique if constructed the same way.

```
resource mystorage 'Microsoft.Storage/storageAccounts@2018-07-01' = {
  name: 'storage${uniqueString(resourceGroup().id)}'
  ...
}
```

If you need to create a new unique name each time you deploy a Bicep file, and don't intend to update the resource, you can use the utcNow function with uniqueString. You could use this approach in a test environment. For an example, see utcNow. Note the utcNow function can only be used within an expression for the default value of a parameter.

Return value

A string containing 13 characters.

Examples

The following example returns results from uniquestring:

```
output uniqueRG string = uniqueString(resourceGroup().id)
output uniqueDeploy string = uniqueString(resourceGroup().id, deployment().name)
```

uri

`uri(baseUri, relativeUri)`

Creates an absolute URI by combining the baseUri and the relativeUri string.

Namespace: sys.

Parameters

PARAMETER	REQUIRED	TYPE	DESCRIPTION
baseUri	Yes	string	The base uri string. Take care to observe the behavior regarding the handling of the trailing slash ('/'), as described following this table.
relativeUri	Yes	string	The relative uri string to add to the base uri string.

- If **baseUri** ends in a trailing slash, the result is simply **baseUri** followed by **relativeUri**.
- If **baseUri** does not end in a trailing slash one of two things happens.
  - If **baseUri** has no slashes at all (aside from the "//" near the front) the result is simply **baseUri** followed by **relativeUri**.
  - If **baseUri** has some slashes, but doesn't end with a slash, everything from the last slash onward is removed from **baseUri** and the result is **baseUri** followed by **relativeUri**.

Here are some examples:

```
uri('http://contoso.org/firstpath', 'myscript.sh') -> http://contoso.org/myscript.sh
uri('http://contoso.org/firstpath/', 'myscript.sh') -> http://contoso.org/firstpath/myscript.sh
uri('http://contoso.org/firstpath/azuredeploy.json', 'myscript.sh') ->
http://contoso.org/firstpath/myscript.sh
uri('http://contoso.org/firstpath/azuredeploy.json/', 'myscript.sh') ->
http://contoso.org/firstpath/azuredeploy.json/myscript.sh
```

For complete details, the **baseUri** and **relativeUri** parameters are resolved as specified in RFC 3986, section 5.

Return value

A string representing the absolute URI for the base and relative values.

Examples

The following example shows how to use uri, uriComponent, and uriComponentToString:

```
var uriFormat = uri('http://contoso.com/resources/', 'nested/azuredeploy.json')
var uriEncoded = uriComponent(uriFormat)

output uriOutput string = uriFormat
output componentOutput string = uriEncoded
output toStringOutput string = uriComponentToString(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%2Fazured`
toStringOutput	String	`http://contoso.com/resources/nested/azuredeploy.json`

uriComponent

`uricomponent(stringToEncode)`

Encodes a URI.

Namespace: sys.

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 shows how to use uri, uriComponent, and uriComponentToString:

```
var uriFormat = uri('http://contoso.com/resources/', 'nested/azuredeploy.json')
var uriEncoded = uriComponent(uriFormat)

output uriOutput string = uriFormat
output componentOutput string = uriEncoded
output toStringOutput string = uriComponentToString(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%2Fazured`
toStringOutput	String	`http://contoso.com/resources/nested/azuredeploy.json`

uriComponentToString

`uriComponentToString(uriEncodedString)`

Returns a string of a URI encoded value.

Namespace: sys.

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 shows how to use uri, uriComponent, and uriComponentToString:

```
var uriFormat = uri('http://contoso.com/resources/', 'nested/azuredeploy.json')
var uriEncoded = uriComponent(uriFormat)

output uriOutput string = uriFormat
output componentOutput string = uriEncoded
output toStringOutput string = uriComponentToString(uriEncoded)
```

The output from the preceding example with the default values is:

NAME	TYPE	VALUE
uriOutput	String	`http://contoso.com/resources/nested/azuredeploy.json`

NAME	TYPE	VALUE
componentOutput	String	`http%3A%2F%2Fcontoso.com%2Fresources%2Fnested%2Fazured`
toStringOutput	String	`http://contoso.com/resources/nested/azuredeploy.json`

Next steps

- For a description of the sections in a Bicep file, see Understand the structure and syntax of Bicep files.
- To iterate a specified number of times when creating a type of resource, see Iterative loops in Bicep.
- To see how to deploy the Bicep file you've created, see Deploy resources with Bicep and Azure PowerShell.

Bicep operators

9/21/2022 • 2 minutes to read • Edit Online

This article describes the Bicep operators. Operators are used to calculate values, compare values, or evaluate conditions. There are four types of Bicep operators:

- accessor
- comparison
- logical
- numeric

Operator precedence and associativity

The operators below are listed in descending order of precedence (the higher the position the higher the precedence). Operators listed at the same level have equal precedence.

SYMBOL	TYPE OF OPERATION	ASSOCIATIVITY
`(` `)` `[` `]` `.` `::`	Parentheses, array indexers, property accessors, and nested resource accessor	Left to right
`!` `-`	Unary	Right to left
`%` `*` `/`	Multiplicative	Left to right
`+` `-`	Additive	Left to right
`<=` `<` `>` `>=`	Relational	Left to right
`==` `!=` `=~` `!~`	Equality	Left to right
`&&`	Logical AND	Left to right
`||`	Logical OR	Left to right
`?` `:`	Conditional expression (ternary)	Right to left
`??`	Coalesce	Left to right

Parentheses

Enclosing an expression between parentheses allows you to override the default Bicep operator precedence. For example, the expression `x + y / z` evaluates the division first and then the addition. However, the expression `(x + y) / z` evaluates the addition first and division second.

Accessor

The accessor operators are used to access nested resources and properties on objects.

OPERATOR	NAME	DESCRIPTION
`[]`	Index accessor	Access an element of an array or property on an object.
`.`	Function accessor	Call a function on a resource.
`::`	Nested resource accessor	Access a nested resource from outside of the parent resource.
`.`	Property accessor	Access properties of an object.

Comparison

The comparison operators compare values and return either `true` or `false`.

OPERATOR	NAME	DESCRIPTION
`>=`	Greater than or equal	Evaluates if the first value is greater than or equal to the second value.
`>`	Greater than	Evaluates if the first value is greater than the second value.
`<=`	Less than or equal	Evaluates if the first value is less than or equal to the second value.
`<`	Less than	Evaluates if the first value is less than the second value.
`==`	Equals	Evaluates if two values are equal.
`!=`	Not equal	Evaluates if two values are **not** equal.
`=~`	Equal case-insensitive	Ignores case to determine if two values are equal.
`!~`	Not equal case-insensitive	Ignores case to determine if two values are **not** equal.

Logical

The logical operators evaluate boolean values, return non-null values, or evaluate a conditional expression.

OPERATOR	NAME	DESCRIPTION
`&&`	And	Returns `true` if all values are true.
`||`	Or	Returns `true` if either value is true.
`!`	Not	Negates a boolean value. Takes one operand.

OPERATOR	NAME	DESCRIPTION
`??`	Coalesce	Returns the first non-null value.
`?` `:`	Conditional expression	Evaluates a condition for true or false and returns a value.

Numeric

The numeric operators use integers to do calculations and return integer values.

OPERATOR	NAME	DESCRIPTION
`*`	Multiply	Multiplies two integers.
`/`	Divide	Divides an integer by an integer.
`%`	Modulo	Divides an integer by an integer and returns the remainder.
`+`	Add	Adds two integers.
`-`	Subtract	Subtracts one integer from another integer. Takes two operands.
`-`	Minus (unary)	Multiplies an integer by `-1`. Takes one operand.

> **NOTE**
> Subtract and minus use the same operator. The functionality is different because subtract uses two operands and minus uses one operand.

Next steps

- To create a Bicep file, see Quickstart: Create Bicep files with Visual Studio Code.
- For information about how to resolve Bicep type errors, see Any function for Bicep.
- To compare syntax for Bicep and JSON, see Comparing JSON and Bicep for templates.
- For examples of Bicep functions, see Bicep functions.

Bicep accessor operators

9/21/2022 • 3 minutes to read • Edit Online

The accessor operators are used to access child resources, properties on objects, and elements in an array. You can also use the property accessor to use some functions.

OPERATOR	NAME
`[]`	Index accessor
`.`	Function accessor
`::`	Nested resource accessor
`.`	Property accessor

Index accessor

`array[integerIndex]`

`object['stringIndex']`

Use the index accessor to get either an element from an array or a property from an object.

For an **array**, provide the index as an **integer**. The integer matches the zero-based position of the element to retrieve.

For an **object**, provide the index as a **string**. The string matches the name of the object to retrieve.

The following example gets an element in an array.

```
var arrayVar = [
  'Coho'
  'Contoso'
  'Fabrikan'
]

output accessorResult string = arrayVar[1]
```

Output from the example:

NAME	TYPE	VALUE
accessorResult	string	'Contoso'

The next example gets a property on an object.

```
var environmentSettings = {
  dev: {
    name: 'Development'
  }
  prod: {
    name: 'Production'
  }
}

output accessorResult string = environmentSettings['dev'].name
```

Output from the example:

NAME	TYPE	VALUE
accessorResult	string	'Development'

Function accessor

`resourceName.functionName()`

Two functions - getSecret and list* - support the accessor operator for calling the function. These two functions are the only functions that support the accessor operator.

Example

The following example references an existing key vault, then uses `getSecret` to pass a secret to a module.

```
resource kv 'Microsoft.KeyVault/vaults@2019-09-01' existing = {
  name: kvName
  scope: resourceGroup(subscriptionId, kvResourceGroup )
}

module sql './sql.bicep' = {
  name: 'deploySQL'
  params: {
    sqlServerName: sqlServerName
    adminLogin: adminLogin
    adminPassword: kv.getSecret('vmAdminPassword')
  }
}
```

Nested resource accessor

`parentResource::nestedResource`

A nested resource is a resource that is declared within another resource. Use the nested resource accessor `::` to access that nested resources from outside of the parent resource.

Within the parent resource, you reference the nested resource with just the symbolic name. You only need to use the nested resource accessor when referencing the nested resource from outside of the parent resource.

Example

The following example shows how to reference a nested resource from within the parent resource and from outside of the parent resource.

```
resource demoParent 'demo.Rp/parentType@2020-01-01' = {
  name: 'demoParent'
  location: 'West US'

  // Declare a nested resource within 'demoParent'
  resource demoNested 'childType' = {
    name: 'demoNested'
    properties: {
      displayName: 'The nested instance.'
    }
  }

  // Declare another nested resource
  resource demoSibling 'childType' = {
    name: 'demoSibling'
    properties: {
      // Use symbolic name to reference because this line is within demoParent
      displayName: 'Sibling of ${demoNested.properties.displayName}'
    }
  }
}

// Use nested accessor to reference because this line is outside of demoParent
output displayName string = demoParent::demoNested.properties.displayName
```

Property accessor

`objectName.propertyName`

Use property accessors to access properties of an object. Property accessors can be used with any object, including parameters and variables that are objects. You get an error when you use the property access on an expression that isn't an object.

Example

The following example shows an object variable and how to access the properties.

```
var x = {
  y: {
    z: 'Hello'
    a: true
  }
  q: 42
}

output outputZ string = x.y.z
output outputQ int = x.q
```

Output from the example:

NAME	TYPE	VALUE
`outputZ`	string	'Hello'
`outputQ`	integer	42

Typically, you use the property accessor with a resource deployed in the Bicep file. The following example creates a public IP address and uses property accessors to return a value from the deployed resource.

```
resource publicIp 'Microsoft.Network/publicIPAddresses@2020-06-01' = {
  name: publicIpResourceName
  location: location
  properties: {
    publicIPAllocationMethod: dynamicAllocation ? 'Dynamic' : 'Static'
    dnsSettings: {
      domainNameLabel: publicIpDnsLabel
    }
  }
}

// Use property accessor to get value
output ipFqdn string = publicIp.properties.dnsSettings.fqdn
```

Next steps

- To run the examples, use Azure CLI or Azure PowerShell to deploy a Bicep file.
- To create a Bicep file, see Quickstart: Create Bicep files with Visual Studio Code.
- For information about how to resolve Bicep type errors, see Any function for Bicep.

Bicep comparison operators

9/21/2022 • 6 minutes to read • Edit Online

The comparison operators compare values and return either `true` or `false`. To run the examples, use Azure CLI or Azure PowerShell to deploy a Bicep file.

OPERATOR	NAME
`>=`	Greater than or equal
`>`	Greater than
`<=`	Less than or equal
`<`	Less than
`==`	Equals
`!=`	Not equal
`=~`	Equal case-insensitive
`!~`	Not equal case-insensitive

Greater than or equal >=

`operand1 >= operand2`

Evaluates if the first value is greater than or equal to the second value.

Operands

OPERAND	TYPE	DESCRIPTION
`operand1`	integer, string	First value in the comparison.
`operand2`	integer, string	Second value in the comparison.

Return value

If the first value is greater than or equal to the second value, `true` is returned. Otherwise, `false` is returned.

Example

A pair of integers and pair of strings are compared.

```
param firstInt int = 10
param secondInt int = 5

param firstString string = 'A'
param secondString string = 'A'

output intGtE bool = firstInt >= secondInt
output stringGtE bool = firstString >= secondString
```

Output from the example:

NAME	TYPE	VALUE
`intGtE`	boolean	true
`stringGtE`	boolean	true

Greater than >

`operand1 > operand2`

Evaluates if the first value is greater than the second value.

Operands

OPERAND	TYPE	DESCRIPTION
`operand1`	integer, string	First value in the comparison.
`operand2`	integer, string	Second value in the comparison.

Return value

If the first value is greater than the second value, `true` is returned. Otherwise, `false` is returned.

Example

A pair of integers and pair of strings are compared.

```
param firstInt int = 10
param secondInt int = 5

param firstString string = 'bend'
param secondString string = 'band'

output intGt bool = firstInt > secondInt
output stringGt bool = firstString > secondString
```

Output from the example:

The **e** in **bend** makes the first string greater.

NAME	TYPE	VALUE
`intGt`	boolean	true
`stringGt`	boolean	true

Less than or equal <=

`operand1 <= operand2`

Evaluates if the first value is less than or equal to the second value.

Operands

OPERAND	TYPE	DESCRIPTION
`operand1`	integer, string	First value in the comparison.
`operand2`	integer, string	Second value in the comparison.

Return value

If the first value is less than or equal to the second value, `true` is returned. Otherwise, `false` is returned.

Example

A pair of integers and pair of strings are compared.

```
param firstInt int = 5
param secondInt int = 10

param firstString string = 'demo'
param secondString string = 'demo'

output intLtE bool = firstInt <= secondInt
output stringLtE bool = firstString <= secondString
```

Output from the example:

NAME	TYPE	VALUE
`intLtE`	boolean	true
`stringLtE`	boolean	true

Less than <

`operand1 < operand2`

Evaluates if the first value is less than the second value.

Operands

OPERAND	TYPE	DESCRIPTION
`operand1`	integer, string	First value in the comparison.
`operand2`	integer, string	Second value in the comparison.

Return value

If the first value is less than the second value, `true` is returned. Otherwise, `false` is returned.

Example

A pair of integers and pair of strings are compared.

```
param firstInt int = 5
param secondInt int = 10

param firstString string = 'demo'
param secondString string = 'Demo'

output intLt bool = firstInt < secondInt
output stringLt bool = firstString < secondString
```

Output from the example:

The string is `true` because lowercase letters are less than uppercase letters.

NAME	TYPE	VALUE
`intLt`	boolean	true
`stringLt`	boolean	true

Equals ==

`operand1 == operand2`

Evaluates if the values are equal.

Operands

OPERAND	TYPE	DESCRIPTION
`operand1`	string, integer, boolean, array, object	First value in the comparison.
`operand2`	string, integer, boolean, array, object	Second value in the comparison.

Return value

If the operands are equal, `true` is returned. If the operands are different, `false` is returned.

Example

Pairs of integers, strings, and booleans are compared.

```
param firstInt int = 5
param secondInt int = 5

param firstString string = 'demo'
param secondString string = 'demo'

param firstBool bool = true
param secondBool bool = true

output intEqual bool = firstInt == secondInt
output stringEqual bool = firstString == secondString
output boolEqual bool = firstBool == secondBool
```

Output from the example:

NAME	TYPE	VALUE
`intEqual`	boolean	true
`stringEqual`	boolean	true
`boolEqual`	boolean	true

When comparing arrays, the two arrays must have the same elements and order. The arrays don't need to be assigned to each other.

```
var array1 = [
  1
  2
  3
]

var array2 = [
  1
  2
  3
]

var array3 = array2

var array4 = [
  3
  2
  1
]

output sameElements bool = array1 == array2 // returns true because arrays are defined with same elements
output assignArray bool = array2 == array3 // returns true because one array was defined as equal to the
other array
output differentOrder bool = array4 == array1 // returns false because order of elements is different
```

Output from the example:

NAME	TYPE	VALUE
sameElements	bool	true
assignArray	bool	true
differentOrder	bool	false

When comparing objects, the property names and values must be the same. The properties don't need to be defined in the same order.

```
var object1 = {
  prop1: 'val1'
  prop2: 'val2'
}

var object2 = {
  prop1: 'val1'
  prop2: 'val2'
}

var object3 = {
  prop2: 'val2'
  prop1: 'val1'
}

var object4 = object3

var object5 = {
  prop1: 'valX'
  prop2: 'valY'
}

output sameObjects bool = object1 == object2 // returns true because both objects defined with same
properties
output differentPropertyOrder bool = object3 == object2 // returns true because both objects have same
properties even though order is different
output assignObject bool = object4 == object1 // returns true because one object was defined as equal to the
other object
output differentValues bool = object5 == object1 // returns false because values are different
```

Output from the example:

NAME	TYPE	VALUE
sameObjects	bool	true
differentPropertyOrder	bool	true
assignObject	bool	true
differentValues	bool	false

Not equal !=

`operand1 != operand2`

Evaluates if two values are **not** equal.

Operands

OPERAND	TYPE	DESCRIPTION
`operand1`	string, integer, boolean, array, object	First value in the comparison.
`operand2`	string, integer, boolean, array, object	Second value in the comparison.

Return value

If the operands are **not** equal, `true` is returned. If the operands are equal, `false` is returned.

Example

Pairs of integers, strings, and booleans are compared.

```
param firstInt int = 10
param secondInt int = 5

param firstString string = 'demo'
param secondString string = 'test'

param firstBool bool = false
param secondBool bool = true

output intNotEqual bool = firstInt != secondInt
output stringNotEqual bool = firstString != secondString
output boolNotEqual bool = firstBool != secondBool
```

Output from the example:

NAME	TYPE	VALUE
`intNotEqual`	boolean	true
`stringNotEqual`	boolean	true
`boolNotEqual`	boolean	true

For arrays and objects, see examples in equals.

Equal case-insensitive =~

`operand1 =~ operand2`

Ignores case to determine if the two values are equal.

Operands

OPERAND	TYPE	DESCRIPTION
`operand1`	string	First string in the comparison.
`operand2`	string	Second string in the comparison.

Return value

If the strings are equal, `true` is returned. Otherwise, `false` is returned.

Example

Compares strings that use mixed-case letters.

```
param firstString string = 'demo'
param secondString string = 'DEMO'

param thirdString string = 'demo'
param fourthString string = 'TEST'

output strEqual1 bool = firstString =~ secondString
output strEqual2 bool = thirdString =~ fourthString
```

Output from the example:

NAME	TYPE	VALUE
`strEqual1`	boolean	true
`strEqual2`	boolean	false

Not equal case-insensitive !~

`operand1 !~ operand2`

Ignores case to determine if the two values are **not** equal.

Operands

OPERAND	TYPE	DESCRIPTION
`operand1`	string	First string in the comparison.
`operand2`	string	Second string in the comparison.

Return value

If the strings are **not** equal, `true` is returned. Otherwise, `false` is returned.

Example

Compares strings that use mixed-case letters.

```
param firstString string = 'demo'
param secondString string = 'TEST'

param thirdString string = 'demo'
param fourthString string = 'DeMo'

output strNotEqual1 bool = firstString !~ secondString
output strEqual2 bool = thirdString !~ fourthString
```

Output from the example:

NAME	TYPE	VALUE
`strNotEqual1`	boolean	true
`strNotEqual2`	boolean	false

Next steps

- To create a Bicep file, see Quickstart: Create Bicep files with Visual Studio Code.
- For information about how to resolve Bicep type errors, see Any function for Bicep.
- To compare syntax for Bicep and JSON, see Comparing JSON and Bicep for templates.
- For examples of Bicep functions, see Bicep functions.

Bicep logical operators

9/21/2022 • 3 minutes to read • Edit Online

The logical operators evaluate boolean values, return non-null values, or evaluate a conditional expression. To run the examples, use Azure CLI or Azure PowerShell to deploy a Bicep file.

OPERATOR	NAME
`&&`	And
`||`	Or
`!`	Not
`??`	Coalesce
`?` `:`	Conditional expression

And &&

`operand1 && operand2`

Determines if both values are true.

Operands

OPERAND	TYPE	DESCRIPTION
`operand1`	boolean	The first value to check if true.
`operand2`	boolean	The second value to check if true.
More operands	boolean	More operands can be included.

Return value

`True` when both values are true, otherwise `false` is returned.

Example

Evaluates a set of parameter values and a set of expressions.

```
param operand1 bool = true
param operand2 bool = true

output andResultParm bool = operand1 && operand2
output andResultExp bool = bool(10 >= 10) && bool(5 > 2)
```

Output from the example:

NAME	TYPE	VALUE
`andResultParm`	boolean	true
`andResultExp`	boolean	true

Or ||

`operand1 || operand2`

Determines if either value is true.

Operands

OPERAND	TYPE	DESCRIPTION
`operand1`	boolean	The first value to check if true.
`operand2`	boolean	The second value to check if true.
More operands	boolean	More operands can be included.

Return value

`True` when either value is true, otherwise `false` is returned.

Example

Evaluates a set of parameter values and a set of expressions.

```
param operand1 bool = true
param operand2 bool = false

output orResultParm bool = operand1 || operand2
output orResultExp bool = bool(10 >= 10) || bool(5 < 2)
```

Output from the example:

NAME	TYPE	VALUE
`orResultParm`	boolean	true
`orResultExp`	boolean	true

Not !

`!boolValue`

Negates a boolean value.

Operand

OPERAND	TYPE	DESCRIPTION
`boolValue`	boolean	Boolean value that's negated.

Return value

Negates the initial value and returns a boolean. If the initial value is `true`, then `false` is returned.

Example

The `not` operator negates a value. The values can be wrapped with parentheses.

```
param initTrue bool = true
param initFalse bool = false

output startedTrue bool = !(initTrue)
output startedFalse bool = !initFalse
```

Output from the example:

NAME	TYPE	VALUE
`startedTrue`	boolean	false
`startedFalse`	boolean	true

Coalesce ??

`operand1 ?? operand2`

Returns first non-null value from operands.

Operands

OPERAND	TYPE	DESCRIPTION
`operand1`	string, integer, boolean, object, array	Value to test for `null`.
`operand2`	string, integer, boolean, object, array	Value to test for `null`.
More operands	string, integer, boolean, object, array	Value to test for `null`.

Return value

Returns the first non-null value. Empty strings, empty arrays, and empty objects aren't `null` and an <empty> value is returned.

Example

The output statements return the non-null values. The output type must match the type in the comparison or an error is generated.

```
param myObject object = {
  'isnull1': null
  'isnull2': null
  'string': 'demoString'
  'emptystr': ''
  'integer': 10
  }

output nonNullStr string = myObject.isnull1 ?? myObject.string ?? myObject.isnull2
output nonNullInt int = myObject.isnull1 ?? myObject.integer ?? myObject.isnull2
output nonNullEmpty string = myObject.isnull1 ?? myObject.emptystr ?? myObject.string ?? myObject.isnull2
```

Output from the example:

NAME	TYPE	VALUE
`nonNullStr`	string	demoString
`nonNullInt`	int	10
`nonNullEmpty`	string	<empty>

Conditional expression ? :

`condition ? true-value : false-value`

Evaluates a condition and returns a value whether the condition is true or false.

Operands

OPERAND	TYPE	DESCRIPTION
`condition`	boolean	Condition to evaluate as true or false.
`true-value`	string, integer, boolean, object, array	Value when condition is true.
`false-value`	string, integer, boolean, object, array	Value when condition is false.

Example

This example evaluates a parameter's initial and returns a value whether the condition is true or false.

```
param initValue bool = true

output outValue string = initValue ? 'true value' : 'false value'
```

Output from the example:

NAME	TYPE	VALUE
`outValue`	string	true value

Next steps

- To create a Bicep file, see Quickstart: Create Bicep files with Visual Studio Code.
- For information about how to resolve Bicep type errors, see Any function for Bicep.
- To compare syntax for Bicep and JSON, see Comparing JSON and Bicep for templates.
- For examples of Bicep functions, see Bicep functions.

Bicep numeric operators

9/21/2022 • 3 minutes to read • Edit Online

The numeric operators use integers to do calculations and return integer values. To run the examples, use Azure CLI or Azure PowerShell to deploy Bicep file.

OPERATOR	NAME
`*`	Multiply
`/`	Divide
`%`	Modulo
`+`	Add
`-`	Subtract
`-`	Minus

> **NOTE**
> Subtract and minus use the same operator. The functionality is different because subtract uses two operands and minus uses one operand.

Multiply *

`operand1 * operand2`

Multiplies two integers.

Operands

OPERAND	TYPE	DESCRIPTION
`operand1`	integer	Number to multiply.
`operand2`	integer	Multiplier of the number.

Return value

The multiplication returns the product as an integer.

Example

Two integers are multiplied and return the product.

```
param firstInt int = 5
param secondInt int = 2

output product int = firstInt * secondInt
```

Output from the example:

NAME	TYPE	VALUE
`product`	integer	10

Divide /

`operand1 / operand2`

Divides an integer by an integer.

Operands

OPERAND	TYPE	DESCRIPTION
`operand1`	integer	Integer that's divided.
`operand2`	integer	Integer that's used for division. Can't be zero.

Return value

The division returns the quotient as an integer.

Example

Two integers are divided and return the quotient.

```
param firstInt int = 10
param secondInt int = 2

output quotient int = firstInt / secondInt
```

Output from the example:

NAME	TYPE	VALUE
`quotient`	integer	5

Modulo %

`operand1 % operand2`

Divides an integer by an integer and returns the remainder.

Operands

OPERAND	TYPE	DESCRIPTION
`operand1`	integer	The integer that's divided.
`operand2`	integer	The integer that's used for division. Can't be 0.

Return value

The remainder is returned as an integer. If the division doesn't produce a remainder, 0 is returned.

Example

Two pairs of integers are divided and return the remainders.

```
param firstInt int = 10
param secondInt int = 3

param thirdInt int = 8
param fourthInt int = 4

output remainder int = firstInt % secondInt
output zeroRemainder int = thirdInt % fourthInt
```

Output from the example:

NAME	TYPE	VALUE
`remainder`	integer	1
`zeroRemainder`	integer	0

Add +

`operand1 + operand2`

Adds two integers.

Operands

OPERAND	TYPE	DESCRIPTION
`operand1`	integer	Number to add.
`operand2`	integer	Number that's added to a number.

Return value

The addition returns the sum as an integer.

Example

Two integers are added and return the sum.

```
param firstInt int = 10
param secondInt int = 2

output sum int = firstInt + secondInt
```

Output from the example:

NAME	TYPE	VALUE
`sum`	integer	12

Subtract -

`operand1 - operand2`

Subtracts an integer from an integer.

Operands

OPERAND	TYPE	DESCRIPTION
`operand1`	integer	Larger number that's subtracted from.
`operand2`	integer	Number that's subtracted from the larger number.

Return value

The subtraction returns the difference as an integer.

Example

An integer is subtracted and returns the difference.

```
param firstInt int = 10
param secondInt int = 4

output difference int = firstInt - secondInt
```

Output from the example:

NAME	TYPE	VALUE
`difference`	integer	6

Minus -

`-integerValue`

Multiplies an integer by `-1`.

Operand

OPERAND	TYPE	DESCRIPTION
`integerValue`	integer	Integer multiplied by `-1`.

Return value

An integer is multiplied by `-1`. A positive integer returns a negative integer and a negative integer returns a positive integer. The values can be wrapped with parentheses.

Example

```
param posInt int = 10
param negInt int = -20

output startedPositive int = -posInt
output startedNegative int = -(negInt)
```

Output from the example:

NAME	TYPE	VALUE
`startedPositive`	integer	-10
`startedNegative`	integer	20

Next steps

- To create a Bicep file, see Quickstart: Create Bicep files with Visual Studio Code.
- For information about how to resolve Bicep type errors, see Any function for Bicep.
- To compare syntax for Bicep and JSON, see Comparing JSON and Bicep for templates.
- For examples of Bicep functions, see Bicep functions.

Best practices for Bicep

9/21/2022 • 5 minutes to read • Edit Online

This article recommends practices to follow when developing your Bicep files. These practices make your Bicep file easier to understand and use.

Training resources

If you would rather learn about Bicep best practices through step-by-step guidance, see Structure your Bicep code for collaboration.

Parameters

- Use good naming for parameter declarations. Good names make your templates easy to read and understand. Make sure you're using clear, descriptive names, and be consistent in your naming.
- Think carefully about the parameters your template uses. Try to use parameters for settings that change between deployments. Variables and hard-coded values can be used for settings that don't change between deployments.
- Be mindful of the default values you use. Make sure the default values are safe for anyone to deploy. For example, consider using low-cost pricing tiers and SKUs so that someone deploying the template to a test environment doesn't incur a large cost unnecessarily.
- Use the `@allowed` decorator sparingly. If you use this decorator too broadly, you might block valid deployments. As Azure services add SKUs and sizes, your allowed list might not be up to date. For example, allowing only Premium v3 SKUs might make sense in production, but it prevents you from using the same template in non-production environments.
- It's a good practice to provide descriptions for your parameters. Try to make the descriptions helpful, and provide any important information about what the template needs the parameter values to be.

  You can also use `//` comments to add notes within your Bicep files.
- You can put parameter declarations anywhere in the template file, although it's usually a good idea to put them at the top of the file so your Bicep code is easy to read.
- It's a good practice to specify the minimum and maximum character length for parameters that control naming. These limitations help avoid errors later during deployment.

For more information about Bicep parameters, see Parameters in Bicep.

Variables

- When you define a variable, the data type isn't needed. Variables infer the type from the resolve value.
- You can use Bicep functions to create a variable.
- After a variable is defined in your Bicep file, you reference the value using the variable's name.

For more information about Bicep variables, see Variables in Bicep.

Names

- Use lower camel case for names, such as `myVariableName` or `myResource`.

- The uniqueString() function is useful for creating unique resource names. When you provide the same parameters, it returns the same string every time. Passing in the resource group ID means the string is the same on every deployment to the same resource group, but different when you deploy to different resource groups or subscriptions.
- It's a good practice to use template expressions to create resource names, like in this example:

  ```
  param shortAppName string = 'toy'
  param shortEnvironmentName string = 'prod'
  param appServiceAppName string = '${shortAppName}-${shortEnvironmentName}-
  ${uniqueString(resourceGroup().id)}'
  ```

  Using template expressions to create resource names gives you several benefits:

  - Strings generated by `uniqueString()` aren't meaningful. It's helpful to use a template expression to create a name that includes meaningful information, such as a short descriptor of the project or environment name, as well as a random component to make the name more likely to be unique.
  - The `uniqueString()` function doesn't guarantee globally unique names. By adding additional text to your resource names, you reduce the likelihood of reusing an existing resource name.
  - Sometimes the `uniqueString()` function creates strings that start with a number. Some Azure resources, like storage accounts, don't allow their names to start with numbers. This requirement means it's a good idea to use string interpolation to create resource names. You can add a prefix to the unique string.
  - Many Azure resource types have rules about the allowed characters and length of their names. Embedding the creation of resource names in the template means that anyone who uses the template doesn't have to remember to follow these rules themselves.
- Avoid using `name` in a symbolic name. The symbolic name represents the resource, not the resource's name. For example, instead of the following syntax:

  ```
  resource cosmosDBAccountName 'Microsoft.DocumentDB/databaseAccounts@2021-04-15' = {
  ```

  Use:

  ```
  resource cosmosDBAccount 'Microsoft.DocumentDB/databaseAccounts@2021-04-15' = {
  ```

- Avoid distinguishing variables and parameters by the use of suffixes.

Resource definitions

- Instead of embedding complex expressions directly into resource properties, use variables to contain the expressions. This approach makes your Bicep file easier to read and understand. It avoids cluttering your resource definitions with logic.
- Try to use resource properties as outputs, rather than making assumptions about how resources will behave. For example, if you need to output the URL to an App Service app, use the defaultHostname property of the app instead of creating a string for the URL yourself. Sometimes these assumptions aren't correct in different environments, or the resources change the way they work. It's safer to have the resource tell you its own properties.
- It's a good idea to use a recent API version for each resource. New features in Azure services are sometimes available only in newer API versions.

- When possible, avoid using the reference and resourceId functions in your Bicep file. You can access any resource in Bicep by using the symbolic name. For example, if you define a storage account with the symbolic name toyDesignDocumentsStorageAccount, you can access its resource ID by using the expression `toyDesignDocumentsStorageAccount.id`. By using the symbolic name, you create an implicit dependency between resources.
- Prefer using implicit dependencies over explicit dependencies. Although the `dependsOn` resource property enables you to declare an explicit dependency between resources, it's usually possible to use the other resource's properties by using its symbolic name. This approach creates an implicit dependency between the two resources, and enables Bicep to manage the relationship itself.
- If the resource isn't deployed in the Bicep file, you can still get a symbolic reference to the resource using the `existing` keyword.

Child resources

- Avoid nesting too many layers deep. Too much nesting makes your Bicep code harder to read and work with.
- Avoid constructing resource names for child resources. You lose the benefits that Bicep provides when it understands the relationships between your resources. Use the `parent` property or nesting instead.

Outputs

- Make sure you don't create outputs for sensitive data. Output values can be accessed by anyone who has access to the deployment history. They're not appropriate for handling secrets.
- Instead of passing property values around through outputs, use the existing keyword to look up properties of resources that already exist. It's a best practice to look up keys from other resources in this way instead of passing them around through outputs. You'll always get the most up-to-date data.

For more information about Bicep outputs, see Outputs in Bicep.

Tenant scopes

You can't create policies or role assignments at the tenant scope. However, if you need to grant access or apply policies across your whole organization, you can deploy these resources to the root management group.

Next steps

- For an introduction to Bicep, see Bicep quickstart.
- For information about the parts of a Bicep file, see Understand the structure and syntax of Bicep files.

Create Bicep files by using Visual Studio Code

9/21/2022 • 2 minutes to read • Edit Online

This article shows you how to use Visual Studio Code to create Bicep files.

Install VS Code

To set up your environment for Bicep development, see Install Bicep tools. After completing those steps, you'll have Visual Studio Code and the Bicep extension. You also have either the latest Azure CLI or the latest Azure PowerShell module.

Bicep commands

Visual Studio Code comes with several Bicep commands.

Open or create a Bicep file in VS Code, select the **View** menu and then select **Command Palette**. You can also use the key combination **[CTRL]+[SHIFT]+P** to bring up the command palette. Type **Bicep** to list the Bicep commands.

These commands include:

- Build ARM Template
- Create Bicep Configuration File
- Deploy Bicep File
- Generate Parameters File
- Insert Resource
- Open Bicep Visualizer
- Open Bicep Visualizer to the side
- Restore Bicep Modules (Force)

These commands are also shown in the context menu when you right-click a Bicep file:

Build ARM template

The `build` command converts a Bicep file to an Azure Resource Manager template (ARM template). The new JSON template is stored in the same folder with the same file name. If a file with the same file name exists, it overwrites the old file. For more information, see Bicep CLI commands.

Create Bicep configuration file

The Bicep configuration file (bicepconfig.json) can be used to customize your Bicep development experience. You can add `bicepconfig.json` in multiple directories. The configuration file closest to the bicep file in the directory hierarchy is used. When you select this command, the extension opens a dialog for you to select a folder. The default folder is where you store the Bicep file. If a `bicepconfig.json` file already exists in the folder, you can overwrite the existing file.

Deploy Bicep file

You can deploy Bicep files directly from Visual Studio Code. Select **Deploy Bicep file** from the command palette or from the context menu. The extension prompts you to sign in Azure, select subscription, create/select resource group, and enter parameter values.

Generate parameters file

This command creates a parameter file in the same folder as the Bicep file. The new parameter file name is `<bicep-file-name>.parameters.json`.

Insert resource

The `insert resource` command adds a resource declaration in the Bicep file by providing the resource ID of an existing resource. After you select **Insert Resource**, enter the resource ID in the command palette. It takes a few moments to insert the resource.

You can find the resource ID from the Azure portal, or by using:

- CLI
- PowerShell

```
az resource list
```

Similar to exporting templates, the process tries to create a usable resource. However, most of the inserted resources require some modification before they can be used to deploy Azure resources.

For more information, see Decompiling ARM template JSON to Bicep.

Open Bicep visualizer

The visualizer shows the resources defined in the Bicep file with the resource dependency information. The diagram is the visualization of a Linux virtual machine Bicep file.

You can also open the visualizer side-by-side with the Bicep file.

Restore Bicep modules

When your Bicep file uses modules that are published to a registry, the restore command gets copies of all the required modules from the registry. It stores those copies in a local cache. For more information, see restore.

View type document

From Visual Studio Code, you can easily open the template reference for the resource type you're working on. To do so, hover your cursor over the resource symbolic name, and then select **View type document**.

Next steps

To walk through a quickstart, see Quickstart: Create Bicep files with Visual Studio Code.

Create private registry for Bicep modules

9/21/2022 • 3 minutes to read • Edit Online

To share modules within your organization, you can create a private module registry. You publish modules to that registry and give read access to users who need to deploy the modules. After the modules are shared in the registries, you can reference them from your Bicep files. To contribute to the public module registry, see the contribution guide.

To work with module registries, you must have Bicep CLI version **0.4.1008 or later**. To use with Azure CLI, you must also have version **2.31.0 or later**; to use with Azure PowerShell, you must also have version **7.0.0** or later.

Training resources

If you would rather learn about parameters through step-by-step guidance, see Share Bicep modules by using private registries.

Configure private registry

A Bicep registry is hosted on Azure Container Registry (ACR). Use the following steps to configure your registry for modules.

1. If you already have a container registry, you can use it. If you need to create a container registry, see Quickstart: Create a container registry by using a Bicep file.

   You can use any of the available registry SKUs for the module registry. Registry geo-replication provides users with a local presence or as a hot-backup.

2. Get the login server name. You need this name when linking to the registry from your Bicep files. The format of the login server name is: `<registry-name>.azurecr.io`.

   - PowerShell
   - Azure CLI

   To get the login server name, use Get-AzContainerRegistry.

   ```
   Get-AzContainerRegistry -ResourceGroupName "<resource-group-name>" -Name "<registry-name>"  | Select-Object LoginServer
   ```

3. To publish modules to a registry, you must have permission to **push** an image. To deploy a module from a registry, you must have permission to **pull** the image. For more information about the roles that grant adequate access, see Azure Container Registry roles and permissions.

4. Depending on the type of account you use to deploy the module, you may need to customize which credentials are used. These credentials are needed to get the modules from the registry. By default, credentials are obtained from Azure CLI or Azure PowerShell. You can customize the precedence for getting the credentials in the **bicepconfig.json** file. For more information, see Credentials for restoring modules.

> **IMPORTANT**
> The private container registry is only available to users with the required access. However, it's accessed through the public internet. For more security, you can require access through a private endpoint. See Connect privately to an Azure container registry using Azure Private Link.

Publish files to registry

After setting up the container registry, you can publish files to it. Use the publish command and provide any Bicep files you intend to use as modules. Specify the target location for the module in your registry.

- PowerShell
- Azure CLI

```
Publish-AzBicepModule -FilePath ./storage.bicep -Target
br:exampleregistry.azurecr.io/bicep/modules/storage:v1
```

View files in registry

To see the published module in the portal:

1. Sign in to the Azure portal.
2. Search for **container registries**.
3. Select your registry.
4. Select **Repositories** from the left menu.
5. Select the module path (repository). In the preceding example, the module path name is **bicep/modules/storage**.
6. Select the tag. In the preceding example, the tag is **v1**.
7. The **Artifact reference** matches the reference you'll use in the Bicep file.

You're now ready to reference the file in the registry from a Bicep file. For examples of the syntax to use for referencing an external module, see Bicep modules.

Next steps

- To learn about modules, see Bicep modules.
- To configure aliases for a module registry, see Add module settings in the Bicep config file.
- For more information about publishing and restoring modules, see Bicep CLI commands.

Use deployment scripts in Bicep

9/21/2022 • 23 minutes to read • Edit Online

Learn how to use deployment scripts in Bicep. With Microsoft.Resources/deploymentScripts, users can execute scripts in Bicep deployments and review execution results. These scripts can be used for performing custom steps such as:

- add users to a directory
- perform data plane operations, for example, copy blobs or seed database
- look up and validate a license key
- create a self-signed certificate
- create an object in Azure AD
- look up IP Address blocks from custom system

The benefits of deployment script:

- Easy to code, use, and debug. You can develop deployment scripts in your favorite development environments. The scripts can be embedded in Bicep files or in external script files.
- You can specify the script language and platform. Currently, Azure PowerShell and Azure CLI deployment scripts on the Linux environment are supported.
- Allow passing command-line arguments to the script.
- Can specify script outputs and pass them back to the deployment.

The deployment script resource is only available in the regions where Azure Container Instance is available. See Resource availability for Azure Container Instances in Azure regions.

> **IMPORTANT**
>
> A storage account and a container instance are needed for script execution and troubleshooting. You have the options to specify an existing storage account, otherwise the storage account along with the container instance are automatically created by the script service. The two automatically created resources are usually deleted by the script service when the deployment script execution gets in a terminal state. You are billed for the resources until the resources are deleted. To learn more, see Clean-up deployment script resources.

> **NOTE**
>
> Retry logic for Azure sign in is now built in to the wrapper script. If you grant permissions in the same Bicep file as your deployment scripts, the deployment script service retries sign in for 10 minutes with 10-second interval until the managed identity role assignment is replicated.

Training resources

If you would rather learn about the ARM template test toolkit through step-by-step guidance, see Extend ARM templates by using deployment scripts.

Configure the minimum permissions

For deployment script API version 2020-10-01 or later, there are two principals involved in deployment script execution:

- **Deployment principal** (the principal used to deploy the Bicep file): this principal is used to create underlying resources required for the deployment script resource to execute — a storage account and an Azure container instance. To configure the least-privilege permissions, assign a custom role with the following properties to the deployment principal:

  ```
  {
    "roleName": "deployment-script-minimum-privilege-for-deployment-principal",
    "description": "Configure least privilege for the deployment principal in deployment script",
    "type": "customRole",
    "IsCustom": true,
    "permissions": [
      {
        "actions": [
          "Microsoft.Storage/storageAccounts/*",
          "Microsoft.ContainerInstance/containerGroups/*",
          "Microsoft.Resources/deployments/*",
          "Microsoft.Resources/deploymentScripts/*"
        ],
      }
    ],
    "assignableScopes": [
      "[subscription().id]"
    ]
  }
  ```

  If the Azure Storage and the Azure Container Instance resource providers haven't been registered, you also need to add `Microsoft.Storage/register/action` and `Microsoft.ContainerInstance/register/action`.

- **Deployment script principal**: This principal is only required if the deployment script needs to authenticate to Azure and call Azure CLI/PowerShell. There are two ways to specify the deployment script principal:

  - Specify a user-assigned managed identity in the `identity` property (see Sample Bicep files). When specified, the script service calls `Connect-AzAccount -Identity` before invoking the deployment script. The managed identity must have the required access to complete the operation in the script. Currently, only user-assigned managed identity is supported for the `identity` property. To login with a different identity, use the second method in this list.
  - Pass the service principal credentials as secure environment variables, and then can call Connect-AzAccount or az login in the deployment script.

  If a managed identity is used, the deployment principal needs the **Managed Identity Operator** role (a built-in role) assigned to the managed identity resource.

Sample Bicep files

The following Bicep file is an example. For more information, see the latest Bicep schema.

```
resource runPowerShellInline 'Microsoft.Resources/deploymentScripts@2020-10-01' = {
  name: 'runPowerShellInline'
  location: resourceGroup().location
  kind: 'AzurePowerShell'
  identity: {
    type: 'UserAssigned'
    userAssignedIdentities: {
      '/subscriptions/01234567-89AB-CDEF-0123-
456789ABCDEF/resourceGroups/myResourceGroup/providers/Microsoft.ManagedIdentity/userAssignedIdentities/myID'
: {}
    }
  }
  properties: {
    forceUpdateTag: '1'
    containerSettings: {
      containerGroupName: 'mycustomaci'
    }
    storageAccountSettings: {
      storageAccountName: 'myStorageAccount'
      storageAccountKey: 'myKey'
    }
    azPowerShellVersion: '6.4' // or azCliVersion: '2.28.0'
    arguments: '-name \\"John Dole\\"'
    environmentVariables: [
      {
        name: 'UserName'
        value: 'jdole'
      }
      {
        name: 'Password'
        secureValue: 'jDolePassword'
      }
    ]
    scriptContent: '''
      param([string] $name)
      $output = \'Hello {0}. The username is {1}, the password is {2}.\' -f
$name,\${Env:UserName},\${Env:Password}
      Write-Output $output
      $DeploymentScriptOutputs = @{}
      $DeploymentScriptOutputs[\'text\'] = $output
    ''' // or primaryScriptUri: 'https://raw.githubusercontent.com/Azure/azure-docs-bicep-
samples/main/samples/deployment-script/inlineScript.ps1'
    supportingScriptUris: []
    timeout: 'PT30M'
    cleanupPreference: 'OnSuccess'
    retentionInterval: 'P1D'
  }
}
```

NOTE

The example is for demonstration purposes. The properties `scriptContent` and `primaryScriptUri` can't coexist in a Bicep file.

NOTE

The *scriptContent* shows a script with multiple lines. The Azure portal and Azure DevOps pipeline can't parse a deployment script with multiple lines. You can either chain the PowerShell commands (by using semicolons or *\r\n* or *\n*) into one line, or use the `primaryScriptUri` property with an external script file. There are many free JSON string escape/unescape tools available. For example, https://www.freeformatter.com/json-escape.html.

Property value details:

- `identity`: For deployment script API version 2020-10-01 or later, a user-assigned managed identity is optional unless you need to perform any Azure-specific actions in the script. For the API version 2019-10-01-preview, a managed identity is required as the deployment script service uses it to execute the scripts. When the identity property is specified, the script service calls `Connect-AzAccount -Identity` before invoking the user script. Currently, only user-assigned managed identity is supported. To login with a different identity, you can call Connect-AzAccount in the script.

- `kind`: Specify the type of script. Currently, Azure PowerShell and Azure CLI scripts are supported. The values are **AzurePowerShell** and **AzureCLI**.

- `forceUpdateTag`: Changing this value between Bicep file deployments forces the deployment script to re-execute. If you use the `newGuid()` or the `utcNow()` functions, both functions can only be used in the default value for a parameter. To learn more, see Run script more than once.

- `containerSettings`: Specify the settings to customize Azure Container Instance. Deployment script requires a new Azure Container Instance. You can't specify an existing Azure Container Instance. However, you can customize the container group name by using `containerGroupName`. If not specified, the group name is automatically generated.

- `storageAccountSettings`: Specify the settings to use an existing storage account. If `storageAccountName` is not specified, a storage account is automatically created. See Use an existing storage account.

- `azPowerShellVersion` / `azCliVersion`: Specify the module version to be used. See a list of supported Azure PowerShell versions. See a list of supported Azure CLI versions.

  > **IMPORTANT**
  >
  > Deployment script uses the available CLI images from Microsoft Container Registry (MCR). It takes about one month to certify a CLI image for deployment script. Don't use the CLI versions that were released within 30 days. To find the release dates for the images, see Azure CLI release notes. If an unsupported version is used, the error message lists the supported versions.

- `arguments`: Specify the parameter values. The values are separated by spaces.

  Deployment Scripts splits the arguments into an array of strings by invoking the CommandLineToArgvW system call. This step is necessary because the arguments are passed as a command property to Azure Container Instance, and the command property is an array of string.

  If the arguments contain escaped characters, double escaped the characters. For example, in the previous sample Bicep, The argument is `-name \"John Dole\"`. The escaped string is `-name \\"John Dole\\"`.

  To pass a Bicep parameter of type object as an argument, convert the object to a string by using the string() function, and then use the replace() function to replace any `"` into `\\"`. For example:

  ```
  replace(string(parameters('tables')), '"', '\\"')
  ```

  For more information, see the sample Bicep file.

- `environmentVariables`: Specify the environment variables to pass over to the script. For more information, see Develop deployment scripts.

- `scriptContent`: Specify the script content. To run an external script, use `primaryScriptUri` instead. For examples, see Use inline script and Use external script.

- `primaryScriptUri`: Specify a publicly accessible URL to the primary deployment script with supported file extensions. For more information, see Use external scripts.
- `supportingScriptUris`: Specify an array of publicly accessible URLs to supporting files that are called in either `scriptContent` or `primaryScriptUri`. For more information, see Use external scripts.
- `timeout`: Specify the maximum allowed script execution time specified in the ISO 8601 format. Default value is **P1D**.
- `cleanupPreference`. Specify the preference of cleaning up deployment resources when the script execution gets in a terminal state. Default setting is **Always**, which means deleting the resources despite the terminal state (Succeeded, Failed, Canceled). To learn more, see Clean up deployment script resources.
- `retentionInterval`: Specify the interval for which the service retains the deployment script resources after the deployment script execution reaches a terminal state. The deployment script resources will be deleted when this duration expires. Duration is based on the ISO 8601 pattern. The retention interval is between 1 and 26 hours (PT26H). This property is used when `cleanupPreference` is set to **OnExpiration**. To learn more, see Clean up deployment script resources.

Additional samples

- Sample 1: create a key vault and use deployment script to assign a certificate to the key vault.
- Sample 2: create a resource group at the subscription level, create a key vault in the resource group, and then use deployment script to assign a certificate to the key vault.
- Sample 3: create a user-assigned managed identity, assign the contributor role to the identity at the resource group level, create a key vault, and then use deployment script to assign a certificate to the key vault.

Use inline scripts

The following Bicep file has one resource defined with the `Microsoft.Resources/deploymentScripts` type. The highlighted part is the inline script.

```
param name string = '\\"John Dole\\"'
param utcValue string = utcNow()

resource runPowerShellInlineWithOutput 'Microsoft.Resources/deploymentScripts@2020-10-01' = {
  name: 'runPowerShellInlineWithOutput'
  location: resourceGroup().location
  kind: 'AzurePowerShell'
  properties: {
    forceUpdateTag: utcValue
    azPowerShellVersion: '6.4'
    scriptContent: '''
      param([string] $name)
      $output = "Hello {0}" -f $name
      Write-Output $output
      $DeploymentScriptOutputs = @{}
      $DeploymentScriptOutputs["text"] = $output
    '''
    arguments: '-name ${name}'
    timeout: 'PT1H'
    cleanupPreference: 'OnSuccess'
    retentionInterval: 'P1D'
  }
}

output result string = runPowerShellInlineWithOutput.properties.outputs.text
```

The script takes a parameter, and output the parameter value. `DeploymentScriptOutputs` is used for storing outputs. The output line shows how to access the stored values. `Write-Output` is used for debugging purpose. To

learn how to access the output file, see Monitor and troubleshoot deployment scripts. For the property descriptions, see Sample Bicep files.

Save the preceding content into a Bicep file called **inlineScript.bicep**, and use the following PowerShell script to deploy the Bicep file.

```
$resourceGroupName = Read-Host -Prompt "Enter the name of the resource group to be created"
$location = Read-Host -Prompt "Enter the location (i.e. centralus)"

New-AzResourceGroup -Name $resourceGroupName -Location $location

New-AzResourceGroupDeployment -ResourceGroupName $resourceGroupName -TemplateFile "inlineScript.bicep"

Write-Host "Press [ENTER] to continue ..."
```

The output looks like:

```
Outputs                 :
                          Name             Type                       Value
                          ===============  =========================  ==========
                          result           String                     Hello John Dole
```

Load script file

You can use the loadTextContent function to load a script file as a string. This function enables you to keep the script in a separate file and retrieve it as a deployment script. The path you provide to the script file is relative to the Bicep file.

The following example loads a script from a file and uses it for a deployment script.

```
resource exampleScript 'Microsoft.Resources/deploymentScripts@2020-10-01' = {
  name: 'exampleScript'
  location: resourceGroup().location
  kind: 'AzurePowerShell'
  identity: {
    type: 'UserAssigned'
    userAssignedIdentities: {
      '/subscriptions/{sub-id}/resourcegroups/{rg-
name}/providers/Microsoft.ManagedIdentity/userAssignedIdentities/{id-name}': {}
    }
  }
  properties: {
    azPowerShellVersion: '3.0'
    scriptContent: loadTextContent('myscript.ps1')
    retentionInterval: 'P1D'
  }
}
```

Use external scripts

In addition to inline scripts, you can also use external script files. Only primary PowerShell scripts with the *ps1* file extension are supported. For CLI scripts, the primary scripts can have any extensions (or without an extension), as long as the scripts are valid bash scripts. To use external script files, replace `scriptContent` with `primaryScriptUri`. For example:

```
"primaryScriptUri": "https://raw.githubusercontent.com/Azure/azure-docs-bicep-
samples/master/samples/deployment-script/inlineScript.ps1",
```

For a usage example, see the external script.

The external script files must be accessible. To secure your script files that are stored in Azure storage accounts, 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. For more information, see Deploy private ARM template with SAS token.

You're responsible for ensuring the integrity of the scripts that are referenced by deployment script, either `primaryScriptUri` or `supportingScriptUris`. Reference only scripts that you trust.

Use supporting scripts

You can separate complicated logics into one or more supporting script files. The `supportingScriptUris` property allows you to provide an array of URIs to the supporting script files if needed:

```
scriptContent: '''
    ...
    ./Create-Cert.ps1
    ...
'''

supportingScriptUris: [
  'https://raw.githubusercontent.com/Azure/azure-docs-bicep-samples/master/samples/deployment-script/create-cert.ps1'
],
```

Supporting script files can be called from both inline scripts and primary script files. Supporting script files have no restrictions on the file extension.

The supporting files are copied to `azscripts/azscriptinput` at the runtime. Use relative path to reference the supporting files from inline scripts and primary script files.

Work with outputs from PowerShell script

The following Bicep file shows how to pass values between two `deploymentScripts` resources:

```
param name string = 'John Dole'
param utcValue string = utcNow()

resource scriptInTemplate1 'Microsoft.Resources/deploymentScripts@2020-10-01' = {
  name: 'scriptInTemplate1'
  location: resourceGroup().location
  kind: 'AzurePowerShell'
  properties: {
    forceUpdateTag: utcValue
    azPowerShellVersion: '6.4'
    timeout: 'PT1H'
    arguments: '-name \\"${name}\\"'
    scriptContent: '''
      param([string] $name)
      $output = \'Hello {0}\' -f $name
      Write-Output $output
      $DeploymentScriptOutputs = @{}
      $DeploymentScriptOutputs[\'text\'] = $output
    '''
    cleanupPreference: 'Always'
    retentionInterval: 'P1D'
  }
}

resource scriptInTemplate2 'Microsoft.Resources/deploymentScripts@2020-10-01' = {
  name: 'scriptInTemplate2'
  location: resourceGroup().location
  kind: 'AzurePowerShell'
  properties: {
    forceUpdateTag: utcValue
    azPowerShellVersion: '6.4'
    timeout: 'PT1H'
    arguments: '-textToEcho \\"${scriptInTemplate1.properties.outputs.text}\\"'
    scriptContent: '''
      param([string] $textToEcho)
      Write-Output $textToEcho
      $DeploymentScriptOutputs = @{}
      $DeploymentScriptOutputs[\'text\'] = $textToEcho
    '''
    cleanupPreference: 'Always'
    retentionInterval: 'P1D'
  }
}

output result string = scriptInTemplate2.properties.outputs.text
```

In the first resource, you define a variable called `$DeploymentScriptOutputs`, and use it to store the output values. Use resource symbolic name to access the output values.

Work with outputs from CLI script

Different from the PowerShell deployment script, CLI/bash support doesn't expose a common variable to store script outputs, instead, there's an environment variable called `AZ_SCRIPTS_OUTPUT_PATH` that stores the location where the script outputs file resides. If a deployment script is run from a Bicep file, this environment variable is set automatically for you by the Bash shell. The value of `AZ_SCRIPTS_OUTPUT_PATH` is */mnt/azscripts/azscriptoutput/scriptoutputs.json*.

Deployment script outputs must be saved in the `AZ_SCRIPTS_OUTPUT_PATH` location, and the outputs must be a valid JSON string object. The contents of the file must be saved as a key-value pair. For example, an array of strings is stored as `{ "MyResult": [ "foo", "bar"] }`. Storing just the array results, for example `[ "foo", "bar" ]`, is invalid.

```
param identity string
param utcValue string = utcNow()

resource runBashWithOutputs 'Microsoft.Resources/deploymentScripts@2020-10-01' = {
  name: 'runBashWithOutputs'
  location: resourceGroup().location
  kind: 'AzureCLI'
  identity: {
    type: 'UserAssigned'
    userAssignedIdentities: {
      '${identity}': {}
    }
  }
  properties: {
    forceUpdateTag: utcValue
    azCliVersion: '2.28.0'
    timeout: 'PT30M'
    arguments: '\'foo\' \'bar\''
    environmentVariables: [
      {
        name: 'UserName'
        value: 'jdole'
      }
      {
        name: 'Password'
        secureValue: 'jDolePassword'
      }
    ]
    scriptContent: 'result=$(az keyvault list); echo "arg1 is: $1"; echo "arg2 is: $2"; echo "Username is
:$Username"; echo "Password is: $Password"; echo $result | jq -c \'{Result: map({id: .id})}\' >
$AZ_SCRIPTS_OUTPUT_PATH'
    cleanupPreference: 'OnSuccess'
    retentionInterval: 'P1D'
  }
}

output result object = runBashWithOutputs.properties.outputs
```

jq is used in the previous sample. It comes with the container images. See Configure development environment.

Use existing storage account

A storage account and a container instance are needed for script execution and troubleshooting. You have the options to specify an existing storage account, otherwise the storage account along with the container instance are automatically created by the script service. The requirements for using an existing storage account:

- Supported storage account kinds are:

SKU	SUPPORTED KIND
Premium_LRS	FileStorage
Premium_ZRS	FileStorage
Standard_GRS	Storage, StorageV2
Standard_GZRS	StorageV2
Standard_LRS	Storage, StorageV2

SKU	SUPPORTED KIND
Standard_RAGRS	Storage, StorageV2
Standard_RAGZRS	StorageV2
Standard_ZRS	StorageV2

These combinations support file shares. For more information, see Create an Azure file share and Types of storage accounts.

- Storage account firewall rules aren't supported yet. For more information, see Configure Azure Storage firewalls and virtual networks.
- Deployment principal must have permissions to manage the storage account, which includes read, create, delete file shares.

To specify an existing storage account, add the following Bicep to the property element of `Microsoft.Resources/deploymentScripts`:

```
storageAccountSettings: {
  storageAccountName: 'myStorageAccount'
  storageAccountKey: 'myKey'
}
```

- `storageAccountName`: specify the name of the storage account.
- `storageAccountKey`: specify one of the storage account keys. You can use the listKeys() function to retrieve the key. For example:

```
storageAccountSettings: {
  storageAccountName: 'myStorageAccount'
  storageAccountKey: listKeys(resourceId('Microsoft.Storage/storageAccounts', storageAccountName),
'2019-06-01').keys[0].value
}
```

See Sample Bicep file for a complete `Microsoft.Resources/deploymentScripts` definition sample.

When an existing storage account is used, the script service creates a file share with a unique name. See Clean up deployment script resources for how the script service cleans up the file share.

Develop deployment scripts

Handle non-terminating errors

You can control how PowerShell responds to non-terminating errors by using the `$ErrorActionPreference` variable in your deployment script. If the variable isn't set in your deployment script, the script service uses the default value **Continue**.

The script service sets the resource provisioning state to **Failed** when the script encounters an error despite the setting of `$ErrorActionPreference`.

Use environment variables

Deployment script uses these environment variables:

ENVIRONMENT VARIABLE	DEFAULT VALUE	SYSTEM RESERVED
AZ_SCRIPTS_AZURE_ENVIRONMENT	AzureCloud	N
AZ_SCRIPTS_CLEANUP_PREFERENCE	OnExpiration	N
AZ_SCRIPTS_OUTPUT_PATH	<AZ_SCRIPTS_PATH_OUTPUT_DIRECTORY>/<AZ_SCRIPTS_PATH_SCRIPT_OUTPUT_FILE_NAME>	Y
AZ_SCRIPTS_PATH_INPUT_DIRECTORY	/mnt/azscripts/azscriptinput	Y
AZ_SCRIPTS_PATH_OUTPUT_DIRECTORY	/mnt/azscripts/azscriptoutput	Y
AZ_SCRIPTS_PATH_USER_SCRIPT_FILE_NAME	Azure PowerShell: userscript.ps1; Azure CLI: userscript.sh	Y
AZ_SCRIPTS_PATH_PRIMARY_SCRIPT_URI_FILE_NAME	primaryscripturi.config	Y
AZ_SCRIPTS_PATH_SUPPORTING_SCRIPT_URI_FILE_NAME	supportingscripturi.config	Y
AZ_SCRIPTS_PATH_SCRIPT_OUTPUT_FILE_NAME	scriptoutputs.json	Y
AZ_SCRIPTS_PATH_EXECUTION_RESULTS_FILE_NAME	executionresult.json	Y
AZ_SCRIPTS_USER_ASSIGNED_IDENTITY	/subscriptions/	N

For more information about using `AZ_SCRIPTS_OUTPUT_PATH`, see Work with outputs from CLI script.

Pass secured strings to deployment script

Setting environment variables (EnvironmentVariable) in your container instances allows you to provide dynamic configuration of the application or script run by the container. Deployment script handles non-secured and secured environment variables in the same way as Azure Container Instance. For more information, see Set environment variables in container instances. For an example, see Sample Bicep file.

The max allowed size for environment variables is 64 KB.

Monitor and troubleshoot deployment scripts

The script service creates a storage account and a container instance for script execution (unless you specify an existing storage account and/or an existing container instance). If these resources are automatically created by the script service, both resources have the `azscripts` suffix in the resource names.

The user script, the execution results, and the stdout file are stored in the files shares of the storage account. There's a folder called `azscripts`. In the folder, there are two more folders for the input and the output files: `azscriptinput` and `azscriptoutput`.

The output folder contains a *executionresult.json* and the script output file. You can see the script execution error message in *executionresult.json*. The output file is created only when the script is executed successfully. The input folder contains a system PowerShell script file and the user deployment script files. You can replace the user deployment script file with a revised one, and rerun the deployment script from the Azure container instance.

Use the Azure portal

After you deploy a deployment script resource, the resource is listed under the resource group in the Azure portal. The following screenshot shows the **Overview** page of a deployment script resource:

The overview page displays some important information of the resource, such as **Provisioning state**, **Storage account**, **Container instance**, and **Logs**.

From the left menu, you can view the deployment script content, the arguments passed to the script, and the output. You can also export the JSON template for the deployment script including the deployment script.

Use PowerShell

Using Azure PowerShell, you can manage deployment scripts at subscription or resource group scope:

- Get-AzDeploymentScript: Gets or lists deployment scripts.
- Get-AzDeploymentScriptLog: Gets the log of a deployment script execution.
- Remove-AzDeploymentScript: Removes a deployment script and its associated resources.
- Save-AzDeploymentScriptLog: Saves the log of a deployment script execution to disk.

The `Get-AzDeploymentScript` output is similar to:

```
Name                : runPowerShellInlineWithOutput
Id                  : /subscriptions/01234567-89AB-CDEF-0123-
456789ABCDEF/resourceGroups/myds0618rg/providers/Microsoft.Resources/deploymentScripts/runPowerShellInlineWi
thOutput
ResourceGroupName   : myds0618rg
Location            : centralus
SubscriptionId      : 01234567-89AB-CDEF-0123-456789ABCDEF
ProvisioningState   : Succeeded
Identity            : /subscriptions/01234567-89AB-CDEF-0123-
456789ABCDEF/resourceGroups/mydentity1008rg/providers/Microsoft.ManagedIdentity/userAssignedIdentities/myuam
i
ScriptKind          : AzurePowerShell
AzPowerShellVersion : 3.0
StartTime           : 6/18/2020 7:46:45 PM
EndTime             : 6/18/2020 7:49:45 PM
ExpirationDate      : 6/19/2020 7:49:45 PM
CleanupPreference   : OnSuccess
StorageAccountId    : /subscriptions/01234567-89AB-CDEF-0123-
456789ABCDEF/resourceGroups/myds0618rg/providers/Microsoft.Storage/storageAccounts/ftnlvo6rlrvo2azscripts
ContainerInstanceId : /subscriptions/01234567-89AB-CDEF-0123-
456789ABCDEF/resourceGroups/myds0618rg/providers/Microsoft.ContainerInstance/containerGroups/ftnlvo6rlrvo2az
scripts
Outputs             :
                      Key                 Value
                      ==================  ==================
                      text                Hello John Dole

RetentionInterval   : P1D
Timeout             : PT1H
```

Use Azure CLI

Using Azure CLI, you can manage deployment scripts at subscription or resource group scope:

- az deployment-scripts delete: Delete a deployment script.
- az deployment-scripts list: List all deployment scripts.
- az deployment-scripts show: Retrieve a deployment script.
- az deployment-scripts show-log: Show deployment script logs.

The list command output is similar to:

```
[
  {
    "arguments": "-name \\\"John Dole\\\"",
    "azPowerShellVersion": "3.0",
    "cleanupPreference": "OnSuccess",
    "containerSettings": {
      "containerGroupName": null
    },
    "environmentVariables": null,
    "forceUpdateTag": "20200625T025902Z",
    "id": "/subscriptions/01234567-89AB-CDEF-0123-
456789ABCDEF/resourceGroups/myds0624rg/providers/Microsoft.Resources/deploymentScripts/runPowerShellInlineWi
thOutput",
    "identity": {
      "tenantId": "01234567-89AB-CDEF-0123-456789ABCDEF",
      "type": "userAssigned",
      "userAssignedIdentities": {
        "/subscriptions/01234567-89AB-CDEF-0123-
456789ABCDEF/resourceGroups/myidentity1008rg/providers/Microsoft.ManagedIdentity/userAssignedIdentities/myua
mi": {
          "clientId": "01234567-89AB-CDEF-0123-456789ABCDEF",
          "principalId": "01234567-89AB-CDEF-0123-456789ABCDEF"
        }
      }
    },
    "kind": "AzurePowerShell",
    "location": "centralus",
    "name": "runPowerShellInlineWithOutput",
    "outputs": {
      "text": "Hello John Dole"
    },
    "primaryScriptUri": null,
    "provisioningState": "Succeeded",
    "resourceGroup": "myds0624rg",
    "retentionInterval": "1 day, 0:00:00",
    "scriptContent": "\r\n          param([string] $name)\r\n          $output = \"Hello {0}\" -f $name\r\n
Write-Output $output\r\n          $DeploymentScriptOutputs = @{}\r\n
$DeploymentScriptOutputs['text'] = $output\r\n        ",
    "status": {
      "containerInstanceId": "/subscriptions/01234567-89AB-CDEF-0123-
456789ABCDEF/resourceGroups/myds0624rg/providers/Microsoft.ContainerInstance/containerGroups/64lxews2qfa5uaz
scripts",
      "endTime": "2020-06-25T03:00:16.796923+00:00",
      "error": null,
      "expirationTime": "2020-06-26T03:00:16.796923+00:00",
      "startTime": "2020-06-25T02:59:07.595140+00:00",
      "storageAccountId": "/subscriptions/01234567-89AB-CDEF-0123-
456789ABCDEF/resourceGroups/myds0624rg/providers/Microsoft.Storage/storageAccounts/64lxews2qfa5uazscripts"
    },
    "storageAccountSettings": null,
    "supportingScriptUris": null,
    "systemData": {
      "createdAt": "2020-06-25T02:59:04.750195+00:00",
      "createdBy": "someone@contoso.com",
      "createdByType": "User",
      "lastModifiedAt": "2020-06-25T02:59:04.750195+00:00",
      "lastModifiedBy": "someone@contoso.com",
      "lastModifiedByType": "User"
    },
    "tags": null,
    "timeout": "1:00:00",
    "type": "Microsoft.Resources/deploymentScripts"
  }
]
```


Use REST API

You can get the deployment script resource deployment information at the resource group level and the subscription level by using REST API:

```
/subscriptions/<SubscriptionID>/resourcegroups/<ResourceGroupName>/providers/microsoft.resources/deploymentS
cripts/<DeploymentScriptResourceName>?api-version=2020-10-01
```

```
/subscriptions/<SubscriptionID>/providers/microsoft.resources/deploymentScripts?api-version=2020-10-01
```

The following example uses ARMClient:

```
armclient login
armclient get /subscriptions/01234567-89AB-CDEF-0123-
456789ABCDEF/resourcegroups/myrg/providers/microsoft.resources/deploymentScripts/myDeployementScript?api-
version=2020-10-01
```

The output is similar to:

```
{
  "kind": "AzurePowerShell",
  "identity": {
    "type": "userAssigned",
    "tenantId": "01234567-89AB-CDEF-0123-456789ABCDEF",
    "userAssignedIdentities": {
      "/subscriptions/01234567-89AB-CDEF-0123-
456789ABCDEF/resourceGroups/myidentity1008rg/providers/Microsoft.ManagedIdentity/userAssignedIdentities/myua
mi": {
        "principalId": "01234567-89AB-CDEF-0123-456789ABCDEF",
        "clientId": "01234567-89AB-CDEF-0123-456789ABCDEF"
      }
    }
  },
  "location": "centralus",
  "systemData": {
    "createdBy": "someone@contoso.com",
    "createdByType": "User",
    "createdAt": "2020-06-25T02:59:04.7501955Z",
    "lastModifiedBy": "someone@contoso.com",
    "lastModifiedByType": "User",
    "lastModifiedAt": "2020-06-25T02:59:04.7501955Z"
  },
  "properties": {
    "provisioningState": "Succeeded",
    "forceUpdateTag": "20200625T025902Z",
    "azPowerShellVersion": "3.0",
    "scriptContent": "\r\n          param([string] $name)\r\n          $output = \"Hello {0}\" -f $name\r\n
Write-Output $output\r\n          $DeploymentScriptOutputs = @{}\r\n
$DeploymentScriptOutputs['text'] = $output\r\n        ",
    "arguments": "-name \\\"John Dole\\\"",
    "retentionInterval": "P1D",
    "timeout": "PT1H",
    "containerSettings": {},
    "status": {
      "containerInstanceId": "/subscriptions/01234567-89AB-CDEF-0123-
456789ABCDEF/resourceGroups/myds0624rg/providers/Microsoft.ContainerInstance/containerGroups/64lxews2qfa5uaz
scripts",
      "storageAccountId": "/subscriptions/01234567-89AB-CDEF-0123-
456789ABCDEF/resourceGroups/myds0624rg/providers/Microsoft.Storage/storageAccounts/64lxews2qfa5uazscripts",
      "startTime": "2020-06-25T02:59:07.5951401Z",
      "endTime": "2020-06-25T03:00:16.7969234Z",
      "expirationTime": "2020-06-26T03:00:16.7969234Z"
    },
    "outputs": {
      "text": "Hello John Dole"
    },
    "cleanupPreference": "OnSuccess"
  },
  "id": "/subscriptions/01234567-89AB-CDEF-0123-
456789ABCDEF/resourceGroups/myds0624rg/providers/Microsoft.Resources/deploymentScripts/runPowerShellInlineWi
thOutput",
  "type": "Microsoft.Resources/deploymentScripts",
  "name": "runPowerShellInlineWithOutput"
}
```


The following REST API returns the log:

```
/subscriptions/<SubscriptionID>/resourcegroups/<ResourceGroupName>/providers/microsoft.resources/deploymentS
cripts/<DeploymentScriptResourceName>/logs?api-version=2020-10-01
```

It only works before the deployment script resources are deleted.

To see the deploymentScripts resource in the portal, select **Show hidden types**:

Clean up deployment script resources

A storage account and a container instance are needed for script execution and troubleshooting. You have the options to specify an existing storage account, otherwise a storage account along with a container instance are automatically created by the script service. The two automatically created resources are deleted by the script service when the deployment script execution gets in a terminal state. You're billed for the resources until the resources are deleted. For the price information, see Container Instances pricing and Azure Storage pricing.

The life cycle of these resources is controlled by the following properties in the Bicep file:

- `cleanupPreference`: Clean up preference when the script execution gets in a terminal state. The supported values are:
  - **Always**: Delete the automatically created resources once script execution gets in a terminal state. If an existing storage account is used, the script service deletes the file share created in the storage account. Because the `deploymentScripts` resource may still be present after the resources are cleaned up, the script service persists the script execution results, for example, stdout, outputs, and return value before the resources are deleted.
  - **OnSuccess**: Delete the automatically created resources only when the script execution is successful. If an existing storage account is used, the script service removes the file share only when the script execution is successful. You can still access the resources to find the debug information.
  - **OnExpiration**: Delete the automatically created resources only when the `retentionInterval` setting is expired. If an existing storage account is used, the script service removes the file share, but retains the storage account.
- `retentionInterval`: Specify the time interval that a script resource will be retained and after which will be expired and deleted.

> **NOTE**
> It is not recommended to use the storage account and the container instance that are generated by the script service for other purposes. The two resources might be removed depending on the script life cycle.

The container instance and storage account are deleted according to the `cleanupPreference`. However, if the script fails and `cleanupPreference` isn't set to **Always**, the deployment process automatically keeps the container running for one hour. You can use this hour to troubleshoot the script. If you want to keep the container running after successful deployments, add a sleep step to your script. For example, add Start-Sleep to the end of your script. If you don't add the sleep step, the container is set to a terminal state and can't be accessed even if it hasn't been deleted yet.

Run script more than once

Deployment script execution is an idempotent operation. If none of the `deploymentScripts` resource properties (including the inline script) are changed, the script doesn't execute when you redeploy the Bicep file. The deployment script service compares the resource names in the Bicep file with the existing resources in the same resource group. There are two options if you want to execute the same deployment script multiple times:

- Change the name of your `deploymentScripts` resource. For example, use the utcNow Bicep function as the resource name or as a part of the resource name. Changing the resource name creates a new `deploymentScripts` resource. It's good for keeping a history of script execution.

  > **NOTE**
  > The `utcNow` function can only be used in the default value for a parameter.

- Specify a different value in the `forceUpdateTag` property. For example, use `utcNow` as the value.

> **NOTE**
> Write the deployment scripts that are idempotent. This ensures that if they run again accidentally, it will not cause system changes. For example, if the deployment script is used to create an Azure resource, verify the resource doesn't exist before creating it, so the script will succeed or you don't create the resource again.

Configure development environment

You can use a pre-configured container image as your deployment script development environment. For more information, see Configure development environment for deployment scripts.

After the script is tested successfully, you can use it as a deployment script in your Bicep files.

Deployment script error codes

ERROR CODE	DESCRIPTION
DeploymentScriptInvalidOperation	The deployment script resource definition in the Bicep file contains invalid property names.

ERROR CODE	DESCRIPTION
DeploymentScriptResourceConflict	Can't delete a deployment script resource that is in non-terminal state and the execution hasn't exceeded 1 hour. Or can't rerun the same deployment script with the same resource identifier (same subscription, resource group name, and resource name) but different script body content at the same time.
DeploymentScriptOperationFailed	The deployment script operation failed internally. Contact Microsoft support.
DeploymentScriptStorageAccountAccessKeyNotSpecified	The access key hasn't been specified for the existing storage account.
DeploymentScriptContainerGroupContainsInvalidContainers	A container group created by the deployment script service got externally modified, and invalid containers got added.
DeploymentScriptContainerGroupInNonterminalState	Two or more deployment script resources use the same Azure container instance name in the same resource group, and one of them hasn't finished its execution yet.
DeploymentScriptStorageAccountInvalidKind	The existing storage account of the BlobBlobStorage or BlobStorage type doesn't support file shares, and can't be used.
DeploymentScriptStorageAccountInvalidKindAndSku	The existing storage account doesn't support file shares. For a list of supported storage account kinds, see Use existing storage account.
DeploymentScriptStorageAccountNotFound	The storage account doesn't exist or has been deleted by an external process or tool.
DeploymentScriptStorageAccountWithServiceEndpointEnabled	The storage account specified has a service endpoint. A storage account with a service endpoint isn't supported.
DeploymentScriptStorageAccountInvalidAccessKey	Invalid access key specified for the existing storage account.
DeploymentScriptStorageAccountInvalidAccessKeyFormat	Invalid storage account key format. See Manage storage account access keys.
DeploymentScriptExceededMaxAllowedTime	Deployment script execution time exceeded the timeout value specified in the deployment script resource definition.
DeploymentScriptInvalidOutputs	The deployment script output isn't a valid JSON object.
DeploymentScriptContainerInstancesServiceLoginFailure	The user-assigned managed identity wasn't able to sign in after 10 attempts with 1-minute interval.
DeploymentScriptContainerGroupNotFound	A Container group created by deployment script service got deleted by an external tool or process.
DeploymentScriptDownloadFailure	Failed to download a supporting script. See Use supporting script.
DeploymentScriptError	The user script threw an error.

ERROR CODE	DESCRIPTION
DeploymentScriptBootstrapScriptExecutionFailed	The bootstrap script threw an error. Bootstrap script is the system script that orchestrates the deployment script execution.
DeploymentScriptExecutionFailed	Unknown error during the deployment script execution.
DeploymentScriptContainerInstancesServiceUnavailable	When creating the Azure container instance (ACI), ACI threw a service unavailable error.
DeploymentScriptContainerGroupInNonterminalState	When creating the Azure container instance (ACI), another deployment script is using the same ACI name in the same scope (same subscription, resource group name, and resource name).
DeploymentScriptContainerGroupNameInvalid	The Azure container instance name (ACI) specified doesn't meet the ACI requirements. See Troubleshoot common issues in Azure Container Instances.

Next steps

In this article, you learned how to use deployment scripts. To walk through a Learn module:

Extend ARM templates by using deployment scripts

Azure Resource Manager template specs in Bicep

9/21/2022 • 9 minutes to read • Edit Online

A template spec is a resource type for storing an Azure Resource Manager template (ARM template) for later deployment. This resource type enables you to share ARM templates with other users in your organization. Just like any other Azure resource, you can use Azure role-based access control (Azure RBAC) to share the template spec. You can use Azure CLI or Azure PowerShell to create template specs by providing Bicep files. The Bicep files are transpiled into ARM JSON templates before they're stored. Currently, you can't import a Bicep file from the Azure portal to create a template spec resource.

Microsoft.Resources/templateSpecs is the resource type for template specs. It consists of a main template and any number of linked templates. Azure securely stores template specs in resource groups. Both the main template and the linked templates must be in JSON. Template Specs support versioning.

To deploy the template spec, you use standard Azure tools like PowerShell, Azure CLI, Azure portal, REST, and other supported SDKs and clients. You use the same commands as you would for the template or the Bicep file.

> **NOTE**
>
> To use template specs in Bicep with Azure PowerShell, you must install version 6.3.0 or later. To use it with Azure CLI, use version 2.27.0 or later.

When designing your deployment, always consider the lifecycle of the resources and group the resources that share similar lifecycle into a single template spec. For instance, your deployments include multiple instances of Cosmos DB with each instance containing its own databases and containers. Given the databases and the containers don't change much, you want to create one template spec to include a Cosmo DB instance and its underlying databases and containers. You can then use conditional statements in your Bicep along with copy loops to create multiple instances of these resources.

> **TIP**
>
> The choice between module registry and template specs is mostly a matter of preference. There are a few things to consider when you choose between the two:
>
> - Module registry is only supported by Bicep. If you are not yet using Bicep, use template specs.
> - Content in the Bicep module registry can only be deployed from another Bicep file. Template specs can be deployed directly from the API, Azure PowerShell, Azure CLI, and the Azure portal. You can even use `UiFormDefinition` to customize the portal deployment experience.
> - Bicep has some limited capabilities for embedding other project artifacts (including non-Bicep and non-ARM-template files. For example, PowerShell scripts, CLI scripts and other binaries) by using the `loadTextContent` and `loadFileAsBase64` functions. Template specs can't package these artifacts.

Training resources

To learn more about template specs, and for hands-on guidance, see Publish libraries of reusable infrastructure code by using template specs.

Required permissions

To create a template spec, you need **write** access to `Microsoft.Resources/templateSpecs` and `Microsoft.Resources/templateSpecs/versions`.

To deploy a template spec, you need **read** access to `Microsoft.Resources/templateSpecs` and `Microsoft.Resources/templateSpecs/versions`. You also need **write** access to any resources deployed by the template spec, and access to `Microsoft.Resources/deployments/*`.

Why use template specs?

Template specs provide the following benefits:

- You use standard ARM templates or Bicep files for your template spec.
- You manage access through Azure RBAC, rather than SAS tokens.
- Users can deploy the template spec without having write access to the Bicep file.
- You can integrate the template spec into existing deployment process, such as PowerShell script or DevOps pipeline.

Template specs enable you to create canonical templates and share them with teams in your organization. The template specs are secure because they're available to Azure Resource Manager for deployment, but not accessible to users without the correct permission. Users only need read access to the template spec to deploy its template, so you can share the template without allowing others to modify it.

If you currently have your templates in a GitHub repo or storage account, you run into several challenges when trying to share and use the templates. To deploy the template, you need to either make the template publicly accessible or manage access with SAS tokens. To get around this limitation, users might create local copies, which eventually diverge from your original template. Template specs simplify sharing templates.

The templates you include in a template spec should be verified by administrators in your organization to follow the organization's requirements and guidance.

Create template spec

The following example shows a simple Bicep file for creating a storage account in Azure.

```
@allowed([
  'Standard_LRS'
  'Standard_GRS'
  'Standard_ZRS'
  'Premium_LRS'
])
param storageAccountType string = 'Standard_LRS'

resource stg 'Microsoft.Storage/storageAccounts@2021-04-01' = {
  name:  'store${uniqueString(resourceGroup().id)}'
  location: resourceGroup().location
  sku: {
    name: storageAccountType
  }
  kind:'StorageV2'
}
```

Create a template spec by using:

- PowerShell
- CLI

```
New-AzTemplateSpec -Name storageSpec -Version 1.0a -ResourceGroupName templateSpecsRg -Location westus2 -
TemplateFile ./mainTemplate.bicep
```

You can also create template specs by using Bicep files. However the content of `mainTemplate` must be in JSON. The following template creates a template spec to deploy a storage account:

```
param templateSpecName string = 'CreateStorageAccount'
param templateSpecVersionName string = '0.1'
param location string = resourceGroup().location

resource createTemplateSpec 'Microsoft.Resources/templateSpecs@2021-05-01' = {
  name: templateSpecName
  location: location
  properties: {
    description: 'A basic templateSpec - creates a storage account.'
    displayName: 'Storage account (Standard_LRS)'
  }
}

resource createTemplateSpecVersion 'Microsoft.Resources/templateSpecs/versions@2021-05-01' = {
  parent: createTemplateSpec
  name: templateSpecVersionName
  location: location
  properties: {
    mainTemplate: {
      '$schema': 'https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#'
      'contentVersion': '1.0.0.0'
      'parameters': {
        'storageAccountType': {
          'type': 'string'
          'defaultValue': 'Standard_LRS'
          'allowedValues': [
            'Standard_LRS'
            'Standard_GRS'
            'Standard_ZRS'
            'Premium_LRS'
          ]
        }
      }
      'resources': [
        {
          'type': 'Microsoft.Storage/storageAccounts'
          'apiVersion': '2019-06-01'
          'name': 'store$uniquestring(resourceGroup().id)'
          'location': resourceGroup().location
          'kind': 'StorageV2'
          'sku': {
            'name': '[parameters(\'storageAccountType\')]'
          }
        }
      ]
    }
  }
}
```

The JSON template embedded in the Bicep file needs to make these changes:

- Remove the commas at the end of the lines.
- Replace double quotes to single quotes.
- Escape the single quotes within the expressions. For example, **`'name': '[parameters(\'storageAccountType\')]'`**.
- To access the parameters and variables defined in the Bicep file, you can directly use the parameter names and the variable names. To access the parameters and variables defined in `mainTemplate`, you still need to use the ARM JSON template syntax. For example, **`'name': '[parameters(\'storageAccountType\')]'`**.
- Use the Bicep syntax to call Bicep functions. For example, **`'location': resourceGroup().location`**.

The size of a template spec is limited to approximated 2 MB. If a template spec size exceeds the limit, you'll get the **TemplateSpecTooLarge** error code. The error message says:

```
The size of the template spec content exceeds the maximum limit. For large template specs with many
artifacts, the recommended course of action is to split it into multiple template specs and reference them
modularly via TemplateLinks.
```

You can view all template specs in your subscription by using:

- PowerShell
- CLI

```
Get-AzTemplateSpec
```

You can view details of a template spec, including its versions with:

- PowerShell
- CLI

```
Get-AzTemplateSpec -ResourceGroupName templateSpecsRG -Name storageSpec
```

Deploy template spec

After you've created the template spec, users with **read** access to the template spec can deploy it. For information about granting access, see Tutorial: Grant a group access to Azure resources using Azure PowerShell.

Template specs can be deployed through the portal, PowerShell, Azure CLI, or as a Bicep module in a larger template deployment. Users in an organization can deploy a template spec to any scope in Azure (resource group, subscription, management group, or tenant).

Instead of passing in a path or URI for a Bicep file, you deploy a template spec by providing its resource ID. The resource ID has the following format:

/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.Resources/templateSpecs/{template-spec-name}/versions/{template-spec-version}

Notice that the resource ID includes a version name for the template spec.

For example, you deploy a template spec with the following command.

- PowerShell
- CLI

```
$id = "/subscriptions/11111111-1111-1111-1111-
111111111111/resourceGroups/templateSpecsRG/providers/Microsoft.Resources/templateSpecs/storageSpec/versions
/1.0a"

New-AzResourceGroupDeployment `
  -TemplateSpecId $id `
  -ResourceGroupName demoRG
```

In practice, you'll typically run `Get-AzTemplateSpec` or `az ts show` to get the ID of the template spec you want to deploy.

- PowerShell
- CLI

```
$id = (Get-AzTemplateSpec -Name storageSpec -ResourceGroupName templateSpecsRg -Version 1.0a).Versions.Id

New-AzResourceGroupDeployment `
  -ResourceGroupName demoRG `
  -TemplateSpecId $id
```

You can also open a URL in the following format to deploy a template spec:

```
https://portal.azure.com/#create/Microsoft.Template/templateSpecVersionId/%2fsubscriptions%2f{subscription-id}%2fresourceGroups%2f{resource-group-name}%2fproviders%2fMicrosoft.Resources%2ftemplateSpecs%2f{template-spec-name}%2fversions%2f{template-spec-version}
```

Parameters

Passing in parameters to template spec is exactly like passing parameters to a Bicep file. Add the parameter values either inline or in a parameter file.

To pass a parameter inline, use:

- PowerShell
- CLI

```
New-AzResourceGroupDeployment `
  -TemplateSpecId $id `
  -ResourceGroupName demoRG `
  -StorageAccountType Standard_GRS
```

To create a local parameter file, use:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "StorageAccountType": {
      "value": "Standard_GRS"
    }
  }
}
```

And, pass that parameter file with:

- PowerShell
- CLI

```
New-AzResourceGroupDeployment `
  -TemplateSpecId $id `
  -ResourceGroupName demoRG `
  -TemplateParameterFile ./mainTemplate.parameters.json
```

Versioning

When you create a template spec, you provide a version name for it. As you iterate on the template code, you can either update an existing version (for hotfixes) or publish a new version. The version is a text string. You can choose to follow any versioning system, including semantic versioning. Users of the template spec can provide the version name they want to use when deploying it.

Use tags

Tags help you logically organize your resources. You can add tags to template specs by using Azure PowerShell and Azure CLI. The following example shows how to specify tags when creating the template spec:

- PowerShell
- CLI

```
New-AzTemplateSpec `
  -Name storageSpec `
  -Version 1.0a `
  -ResourceGroupName templateSpecsRg `
  -Location westus2 `
  -TemplateFile ./mainTemplate.bicep `
  -Tag @{Dept="Finance";Environment="Production"}
```

The next example shows how to apply tags when updating an existing template spec:

- PowerShell
- CLI

```
Set-AzTemplateSpec `
  -Name storageSpec `
  -Version 1.0a `
  -ResourceGroupName templateSpecsRg `
  -Location westus2 `
  -TemplateFile ./mainTemplate.bicep `
  -Tag @{Dept="Finance";Environment="Production"}
```

Both the template and its versions can have tags. The tags are applied or inherited depending on the parameters you specify.

TEMPLATE SPEC	VERSION	VERSION PARAMETER	TAG PARAMETER	TAG VALUES
Exists	N/A	Not specified	Specified	applied to the template spec
Exists	New	Specified	Not specified	inherited from the template spec to the version

TEMPLATE SPEC	VERSION	VERSION PARAMETER	TAG PARAMETER	TAG VALUES
New	New	Specified	Specified	applied to both template spec and version
Exists	New	Specified	Specified	applied to the version
Exists	Exists	Specified	Specified	applied to the version

Link to template specs

After creating a template spec, you can link to that template spec in a Bicep module. For more information, see File in template spec.

Next steps

To learn more about template specs, and for hands-on guidance, see Publish libraries of reusable infrastructure code by using template specs.

Configuration set pattern

9/21/2022 • 3 minutes to read • Edit Online

Rather than define lots of individual parameters, create predefined sets of values. During deployment, select the set of values to use.

Context and problem

A single Bicep file often defines many resources. Each resource might need to use a different configuration depending on the environment you're deploying it to. For example, you might build a Bicep file that deploys an App Service plan and app, and a storage account. Each of these resources has multiple options that affect its cost, availability, and resiliency. For production environments, you want to use one set of configuration that prioritize high availability and resiliency. For non-production environments, you want to use a different set of configuration that prioritizes cost reduction.

You could create parameters for each configuration setting, but this has some drawbacks:

- This approach creates a burden on your template users, since they need to understand the values to use for each resource, and the impact of setting each parameter.
- The number of parameters in your template increases with each new resource you define.
- Users may select combinations of parameter values that haven't been tested or that won't work correctly.

Solution

Create a single parameter to specify the environment type. Use a variable to automatically select the configuration for each resource based on the value of the parameter.

> **NOTE**
>
> This approach is sometimes called *t-shirt sizing*. When you buy a t-shirt, you don't get lots of options for its length, width, sleeves, and so forth. You simply choose between small, medium, and large sizes, and the t-shirt designer has predefined those measurements based on that size.

Example

Suppose you have a template that can be deployed to two types of environment: non-production and production. Depending on the environment type, the configuration you need is different:

PROPERTY	NON-PRODUCTION ENVIRONMENTS	PRODUCTION ENVIRONMENTS
App Service plan		
SKU name	S2	P2V3
Capacity (number of instances)	1	3
App Service app		
Always On	Disabled	Enabled

PROPERTY	NON-PRODUCTION ENVIRONMENTS	PRODUCTION ENVIRONMENTS
Storage account		
SKU name	Standard_LRS	Standard_ZRS

You could use the configuration set pattern for this template.

Accept a single parameter that indicates the environment type, such as production or non-production. Use the `@allowed` parameter decorator to ensure that your template's users only provide values that you expect:

```
@allowed([
  'Production'
  'NonProduction'
])
param environmentType string = 'NonProduction'
```

Then create a *map variable*, which is an object that defines the specific configuration depending on the environment type. Notice that the variable has two objects named `Production` and `NonProduction`. These names match the allowed values for the parameter in the preceding example:

```
var environmentConfigurationMap = {
  Production: {
    appServicePlan: {
      sku: {
        name: 'P2V3'
        capacity: 3
      }
    }
    appServiceApp: {
      alwaysOn: false
    }
    storageAccount: {
      sku: {
        name: 'Standard_ZRS'
      }
    }
  }
  NonProduction: {
    appServicePlan: {
      sku: {
        name: 'S2'
        capacity: 1
      }
    }
    appServiceApp: {
      alwaysOn: false
    }
    storageAccount: {
      sku: {
        name: 'Standard_LRS'
      }
    }
  }
}
```

When you define the resources, use the configuration map to define the resource properties:

```
resource appServicePlan 'Microsoft.Web/serverfarms@2020-06-01' = {
  name: appServicePlanName
  location: location
  sku: environmentConfigurationMap[environmentType].appServicePlan.sku
}

resource appServiceApp 'Microsoft.Web/sites@2020-06-01' = {
  name: appServiceAppName
  location: location
  properties: {
    serverFarmId: appServicePlan.id
    httpsOnly: true
    siteConfig: {
      alwaysOn: environmentConfigurationMap[environmentType].appServiceApp.alwaysOn
    }
  }
}

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-02-01' = {
  name: storageAccountName
  location: location
  kind: 'StorageV2'
  sku: environmentConfigurationMap[environmentType].storageAccount.sku
}
```

Considerations

- In your map variable, consider grouping the properties by resource to simplify their definition.
- In your map variable, you can define both individual property values (like the `alwaysOn` property in the example), or object variables that set an object property (like the SKU properties in the example).
- Consider using a configuration set with resource conditions. This enables your Bicep code to deploy certain resources for specific environments, and not in others.

Next steps

Learn about the shared variable file pattern.

Logical parameter pattern

9/21/2022 • 3 minutes to read • Edit Online

Use parameters to specify the logical definition of a resource, or even of multiple resources. The Bicep file converts the logical parameter to deployable resource definitions. By following this pattern, you can separate *what's* deployed from *how* it's deployed.

Context and problem

In complex deployments, array parameters and loops are used to create a set of resources or resource properties. If you create a parameter that defines the details of a resource, you create parameters that are difficult to understand and work with.

Solution

Define parameters that accept simplified values, or values that are business domain-specific instead of Azure resource-specific. Build logic into your Bicep file to convert the simplified values to Azure resource definitions.

Often, you'll use a loop to translate the logical values defined in parameters to the property values expected by the resource definition.

When you use this pattern, you can easily apply default values for properties that don't need to change between resources. You can also use variables and functions to determine the values of resource properties automatically based on your own business rules.

By using the Logical parameter pattern, you can deploy complex sets of resources while keeping your template's parameters simple and easy to work with.

Example 1: Virtual network subnets

This example illustrates how you can use the pattern to simplify the definition of new subnets, and to add business logic that determines the resource's configuration.

In this example, you define a parameter that specifies the list of subnets that should be created in a virtual network. Each subnet's definition also includes a property called `allowRdp`. This property indicates whether the subnet should be associated with a network security group that allows inbound remote desktop traffic:

```
param subnets array = [
  {
    name: 'Web'
    addressPrefix: '10.0.0.0/24'
    allowRdp: false
  }
  {
    name: 'JumpBox'
    addressPrefix: '10.0.1.0/24'
    allowRdp: true
  }
]
```

The Bicep file then defines a variable to convert each of the logical subnet definitions to the subnet definition required by Azure. The network security group is assigned to the subnet when the `allowRdp` property is set to `true`.

```
var subnetsToCreate = [for item in subnets: {
  name: item.name
  properties: {
    addressPrefix: item.addressPrefix
    networkSecurityGroup: item.allowRdp ? {
       id: nsgAllowRdp.id
    } : null
  }
}]
```

Finally, the Bicep file defines the virtual network and uses the variable to configure the subnets:

```
resource virtualNetwork 'Microsoft.Network/virtualNetworks@2019-11-01' = {
  name: virtualNetworkName
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        virtualNetworkAddressPrefix
      ]
    }
    subnets: subnetsToCreate
  }
}
```

Refer to the complete example.

Example 2: Service Bus queue

This example demonstrates how you can use the pattern to apply a consistent set of configuration across multiple resources that are defined based on a parameter.

In this example, you define a parameter with a list of queue names for an Azure Service Bus queue:

```
param queueNames array = [
  'queue1'
  'queue2'
]
```

You then define the queue resources by using a loop, and configure every queue to automatically forward dead-lettered messages to another centralized queue:

```
resource queues 'Microsoft.ServiceBus/namespaces/queues@2018-01-01-preview' = [for queueName in queueNames:
{
  parent: serviceBusNamespace
  name: queueName
  properties: {
    forwardDeadLetteredMessagesTo: deadLetterFirehoseQueueName
  }
}]
```

Refer to the complete example.

Example 3: Resources for a multitenant solution

This example illustrates how you might use the pattern when building a multitenant solution. The Bicep deployment creates complex sets of resources based on a logical list of tenants, and uses modules to simplify the creation of shared and tenant-specific resources. Every tenant gets their own database in Azure SQL, and

their own custom domain configured in Azure Front Door.

In this example, you define a parameter that specifies the list of tenants. The definition of a tenant is simply the tenant identifier and their custom domain name:

```
param tenants array = [
  {
    id: 'fabrikam'
    domainName: 'fabrikam.com'
  }
  {
    id: 'contoso'
    domainName: 'contoso.com'
  }
]
```

The main Bicep file defines the shared resources, and then uses a module to loop through the tenants:

```
module tenantResources 'tenant-resources.bicep' = [for tenant in tenants: {
  name: 'tenant-${tenant.id}'
  params: {
    location: location
    tenant: tenant
    sqlServerName: sqlServerName
    frontDoorProfileName: frontDoorProfileName
  }
}]
```

Within the module, the tenant-specific resources are deployed.

Refer to the complete example.

Considerations

- When you're developing or debugging your Bicep file, consider creating the mapping variable without defining a resource. Then, define an output with the variable's value so that you can see the result of the mapping.
- You can use conditions to selectively deploy resources based on logical parameter values.
- When creating resource names dynamically, ensure the names meet the requirements for the resource that you're deploying. You might need to generate a name for some resources rather than accepting names directly through parameters.

Next steps

Learn about the shared variable file pattern.

Name generation pattern

9/21/2022 • 3 minutes to read • Edit Online

Within your Bicep files, use string interpolation and Bicep functions to create resource names that are unique, deterministic, meaningful, and different for each environment that you deploy to.

Context and problem

In Azure, the name you give a resource is important. Names help you and your team to identify the resource. For many services, the resource name forms part of the DNS name you use to access the resource. Names can't easily be changed after the resource is created.

When planning a resource's name, you need to ensure it is:

- **Unique:** Azure resource names need to be unique, but the scope of uniqueness depends on the resource.
- **Deterministic:** It's important that your Bicep file can be repeatedly deployed without recreating existing resources. When you redeploy your Bicep file with the same parameters, resources should maintain the same names.
- **Meaningful:** Names give you and your team important information about the purpose of the resource. Where possible, use names that provide some indication of the purpose of the resource.
- **Distinct for each environment:** It's common to deploy to multiple environments, such as test, staging, and production, as part of your rollout process.
- **Valid for the specific resource:** Each Azure resource has a set of guidelines you must follow when creating a valid resource name. These include maximum lengths, allowed characters, and whether the resource name must start with a letter.

> **NOTE**
>
> Your organization may also have its own naming convention that you need to follow. The Azure Cloud Adoption Framework provide guidance about how to create a naming strategy for your organization. If you have a strict naming convention to follow and the names it generates are distinctive and unique, you might not need to follow this pattern.

Solution

Use Bicep's string interpolation to generate resource names as variables. If the resource requires a globally unique name, use the uniqueString() function to generate part of the resource name. Prepend or append meaningful information to ensure your resources are easily identifiable.

> **NOTE**
>
> Some Azure resources, such as Azure RBAC role definitions and role assignments, need to have globally unique identifiers (GUIDs) as their names. Use the guid() function to generate names for these resources.

If you're creating reusable Bicep code, you should consider defining names as parameters. Use a default parameter value to define a default name that can be overridden. Default values help to make your Bicep files more reusable, ensuring that users of the file can define their own names if they need to follow a different naming convention.

Example 1: Organizational naming convention

The following example generates the names for an App Service app and plan. It follows an organizational convention that includes a resource type code (`app` or `plan`), the application or workload name (`contoso`), the environment name (specified by a parameter), and a string that ensures uniqueness by using the `uniqueString()` function with a seed value of the resource group's ID.

Although App Service plans don't require globally unique names, the plan name is constructed using the same format to ensure compliance with the organization's policy.

```
param location string = resourceGroup().location
param environmentName string
param appServiceAppName string = 'app-contoso-${environmentName}-${uniqueString(resourceGroup().id)}'
param appServicePlanName string = 'plan-contoso-${environmentName}-${uniqueString(resourceGroup().id)}'

resource appServiceApp 'Microsoft.Web/sites@2018-11-01' = {
  name: appServiceAppName
  // ...
}

resource appServicePlan 'Microsoft.Web/serverfarms@2020-12-01' = {
  name: appServicePlanName
  // ...
}
```

Example 2

The following example generates the names for two storage accounts for a different organization without a naming convention. This example again uses the `uniqueString()` function with the resource group's ID. A short string is prepended to the generated names to ensure that each of the two storage accounts has a distinct name. This also helps to ensure that the names begin with a letter, which is a requirement for storage accounts.

```
param primaryStorageAccountName string = 'contosopri${uniqueString(resourceGroup().id)}'
param secondaryStorageAccountName string = 'contososec${uniqueString(resourceGroup().id)}'

resource primaryStorageAccount 'Microsoft.Storage/storageAccounts@2021-02-01' = {
  name: primaryStorageAccountName
  // ...
}

resource secondaryStorageAccount 'Microsoft.Storage/storageAccounts@2021-02-01' = {
  name: secondaryStorageAccountName
  // ...
}
```

Considerations

- Ensure you verify the scope of the uniqueness of your resource names. Use appropriate seed values for the uniqueString() function to ensure that you can reuse the Bicep file across Azure resource groups and subscriptions.

TIP

In most situations, the fully qualified resource group ID is a good option for the seed value for the `uniqueString` function:

```
var uniqueNameComponent = uniqueString(resourceGroup().id)
```

The name of the resource group (`resourceGroup().name`) may not be sufficiently unique to enable you to reuse the file across subscriptions.

- Avoid changing the seed values for the `uniqueString()` function after resources have been deployed. Changing the seed value results in new names, and might affect your production resources.

Next steps

Learn about the shared variable file pattern.

Shared variable file pattern

9/21/2022 • 3 minutes to read • Edit Online

Reduce the repetition of shared values in your Bicep files. Instead, load those values from a shared JSON file within your Bicep file. When using arrays, concatenate the shared values with deployment-specific values in your Bicep code.

Context and problem

When you write your Bicep code, you might have common variables that you reuse across a set of Bicep files. You could duplicate the values each time you declare the resource, such as by copying and pasting the values between your Bicep files. However, this approach is error-prone, and when you need to make changes you need to update each resource definition to keep it in sync with the others.

Furthermore, when you work with variables defined as arrays, you might have a set of common values across multiple Bicep files and also need to add specific values for the resource that you're deploying. When you mix the shared variables with the resource-specific variables, it's harder for someone to understand the distinction between the two categories of variables.

Solution

Create a JSON file that includes the variables you need to share. Use the `loadJsonContent()` function to load the file and access the variables. For array variables, use the `concat()` function to combine the shared values with any custom values for the specific resource.

Example 1: Naming prefixes

Suppose you have multiple Bicep files that define resources. You need to use a consistent naming prefix for all of your resources.

Define a JSON file that includes the common naming prefixes that apply across your company:

```
{
    "storageAccountPrefix": "stg",
    "appServicePrefix": "app"
}
```

In your Bicep file, declare a variable that imports the shared naming prefixes:

```
var sharedNamePrefixes = loadJsonContent('./shared-prefixes.json')
```

When you define your resource names, use string interpolation to concatenate the shared name prefixes with unique name suffixes:

```
var appServiceAppName = '${sharedNamePrefixes.appServicePrefix}-myapp-${uniqueString(resourceGroup().id)}'
var storageAccountName =
'${sharedNamePrefixes.storageAccountPrefix}myapp${uniqueString(resourceGroup().id)}'
```

Example 2: Network security group rules

Suppose you have multiple Bicep file that define their own network security groups (NSG). You have a common set of security rules that must be applied to each NSG, and then you have application-specific rules that must be added.

Define a JSON file that includes the common security rules that apply across your company:

```
{
  "securityRules": [
    {
      "name": "Allow_RDP_from_company_IP_address",
      "properties": {
        "description": "Allow inbound RDP from the company's IP address range.",
        "protocol": "Tcp",
        "sourceAddressPrefix": "203.0.113.0/24",
        "sourcePortRange": "*",
        "destinationAddressPrefix": "VirtualNetwork",
        "destinationPortRange": "3389",
        "access": "Allow",
        "priority": 100,
        "direction": "Inbound"
      }
    },
    {
      "name": "Allow_VirtualNetwork_to_Storage",
      "properties": {
        "description": "Allow outbound connections to the Azure Storage service tag.",
        "protocol": "Tcp",
        "sourceAddressPrefix": "VirtualNetwork",
        "sourcePortRange": "*",
        "destinationAddressPrefix": "Storage",
        "destinationPortRange": "*",
        "access": "Allow",
        "priority": 100,
        "direction": "Outbound"
      }
    }
    // other rules here
  ]
}
```

In your Bicep file, declare a variable that imports the shared security rules:

```
var sharedRules = loadJsonContent('./shared-rules.json', 'securityRules')
```

Create a variable array that represents the custom rules for this specific NSG:

```
var customRules = [
  {
    name: 'Allow_Internet_HTTPS_Inbound'
    properties: {
      description: 'Allow inbound internet connectivity for HTTPS only.'
      protocol: 'Tcp'
      sourcePortRange: '*'
      destinationPortRange: '443'
      sourceAddressPrefix: 'Internet'
      destinationAddressPrefix: 'VirtualNetwork'
      access: 'Allow'
      priority: 400
      direction: 'Inbound'
    }
  }
]
```

Define the NSG resource. Use the `concat()` function to combine the two arrays together and set the `securityRules` property:

```
resource nsg 'Microsoft.Network/networkSecurityGroups@2021-08-01' = {
  name: nsgName
  location: location
  properties: {
    securityRules: concat(sharedRules, customRules)
  }
}
```

Considerations

- When you use this approach, the JSON file will be included inside the ARM template generated by Bicep. The JSON ARM templates generated by Bicep have a file limit of 4MB, so it's important to avoid using large shared variable files.
- Ensure your shared variable arrays don't conflict with the array values specified in each Bicep file. For example, when using the configuration set pattern to define network security groups, ensure you don't have multiple rules that define the same priority and direction.

Next steps

Learn about the configuration set pattern.

Create Azure RBAC resources by using Bicep

9/21/2022 • 5 minutes to read • Edit Online

Azure has a powerful role-based access control (RBAC) system. For more information on Azure RBAC, see What is Azure Role-based access control (Azure RBAC)? By using Bicep, you can programmatically define your RBAC role assignments and role definitions.

Role assignments

Role assignments enable you to grant a principal (such as a user, a group, or a service principal) access to a specific Azure resource.

To define a role assignment, create a resource with type `Microsoft.Authorization/roleAssignments`. A role definition has multiple properties, including a scope, a name, a role definition ID, a principal ID, and a principal type.

Scope

Role assignments apply at a specific *scope*, which defines the resource or set of resources that you're granting access to. For more information, see Understand scope for Azure RBAC.

Role assignments are extension resources, which means they apply to another resource. The following example shows how to create a storage account and a role assignment scoped to that storage account:

```
param location string = resourceGroup().location
param storageAccountName string = 'stor${uniqueString(resourceGroup().id)}'
param storageSkuName string = 'Standard_LRS'
param roleDefinitionResourceId string
param principalId string

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-04-01' = {
  name: storageAccountName
  location: location
  kind: 'StorageV2'
  sku: {
   name: storageSkuName
  }
}

resource roleAssignment 'Microsoft.Authorization/roleAssignments@2020-04-01-preview' = {
  scope: storageAccount
  name: guid(storageAccount.id, principalId, roleDefinitionResourceId)
  properties: {
    roleDefinitionId: roleDefinitionResourceId
    principalId: principalId
    principalType: 'ServicePrincipal'
  }
}
```

If you don't explicitly specify the scope, Bicep uses the file's `targetScope`. In the following example, no `scope` property is specified, so the role assignment is scoped to the subscription:

```
param roleDefinitionResourceId string
param principalId string

targetScope = 'subscription'

resource roleAssignment 'Microsoft.Authorization/roleAssignments@2020-04-01-preview' = {
  name: guid(subscription().id, principalId, roleDefinitionResourceId)
  properties: {
    roleDefinitionId: roleDefinitionResourceId
    principalId: principalId
    principalType: 'ServicePrincipal'
  }
}
```

TIP

Ensure you use the smallest scope required for your requirements.

For example, if you need to grant a managed identity access to a single storage account, it's good security practice to create the role assignment at the scope of the storage account, not at the resource group or subscription scope.

Name

A role assignment's resource name must be a globally unique identifier (GUID).

Role assignment resource names must be unique within the Azure Active Directory tenant, even if the scope is narrower.

For your Bicep deployment to be repeatable, it's important for the name to be deterministic - in other words, to use the same name every time you deploy. It's a good practice to create a GUID that uses the scope, principal ID, and role ID together. It's a good idea to use the `guid()` function to help you to create a deterministic GUID for your role assignment names, like in this example:

```
name: guid(subscription().id, principalId, roleDefinitionResourceId)
```

Role definition ID

The role you assign can be a built-in role definition or a custom role definition. To use a built-in role definition, find the appropriate role definition ID. For example, the *Contributor* role has a role definition ID of `b24988ac-6180-42a0-ab88-20f7382dd24c`.

When you create the role assignment resource, you need to specify a fully qualified resource ID. Built-in role definition IDs are subscription-scoped resources. It's a good practice to use an `existing` resource to refer to the built-in role, and to access its fully qualified resource ID by using the `.id` property:

```
param principalId string

@description('This is the built-in Contributor role. See https://docs.microsoft.com/azure/role-based-access-control/built-in-roles#contributor')
resource contributorRoleDefinition 'Microsoft.Authorization/roleDefinitions@2018-01-01-preview' existing = {
  scope: subscription()
  name: 'b24988ac-6180-42a0-ab88-20f7382dd24c'
}

resource roleAssignment 'Microsoft.Authorization/roleAssignments@2020-04-01-preview' = {
  name: guid(resourceGroup().id, principalId, contributorRoleDefinition.id)
  properties: {
    roleDefinitionId: contributorRoleDefinition.id
    principalId: principalId
    principalType: 'ServicePrincipal'
  }
}
```

Principal

The `principalId` property must be set to a GUID that represents the Azure Active Directory (Azure AD) identifier for the principal. In Azure AD, this is sometimes referred to as the *object ID*.

The `principalType` property specifies whether the principal is a user, a group, or a service principal. Managed identities are a form of service principal.

> **TIP**
> It's important to set the `principalType` property when you create a role assignment in Bicep. Otherwise, you might get intermittent deployment errors, especially when you work with service principals and managed identities.

The following example shows how to create a user-assigned managed identity and a role assignment:

```
param location string = resourceGroup().location
param roleDefinitionResourceId string

var managedIdentityName = 'MyManagedIdentity'

resource managedIdentity 'Microsoft.ManagedIdentity/userAssignedIdentities@2018-11-30' = {
  name: managedIdentityName
  location: location
}

resource roleAssignment 'Microsoft.Authorization/roleAssignments@2020-04-01-preview' = {
  name: guid(resourceGroup().id, managedIdentity.id, roleDefinitionResourceId)
  properties: {
    roleDefinitionId: roleDefinitionResourceId
    principalId: managedIdentity.properties.principalId
    principalType: 'ServicePrincipal'
  }
}
```

Resource deletion behavior

When you delete a user, group, service principal, or managed identity from Azure AD, it's a good practice to delete any role assignments. They aren't deleted automatically.

Any role assignments that refer to a deleted principal ID become invalid. If you try to reuse a role assignment's name for another role assignment, the deployment will fail. To work around this behavior, you should either remove the old role assignment before you recreate it, or ensure that you use a unique name when you deploy a new role assignment. This quickstart template illustrates how you can define a role assignment in a Bicep

module and use a principal ID as a seed value for the role assignment name.

Custom role definitions

Custom role definitions enable you to define a set of permissions that can then be assigned to a principal by using a role assignment. For more information on role definitions, see Understand Azure role definitions.

To create a custom role definition, define a resource of type `Microsoft.Authorization/roleDefinitions`. See the Create a new role def via a subscription level deployment quickstart for an example.

Role definition resource names must be unique within the Azure Active Directory tenant, even if the assignable scopes are narrower.

> **NOTE**
>
> Some services manage their own role definitions and assignments. For example, Azure Cosmos DB maintains its own `Microsoft.DocumentDB/databaseAccounts/sqlRoleAssignments` and `Microsoft.DocumentDB/databaseAccounts/sqlRoleDefinitions` resources. For more information, see the specific service's documentation.

Related resources

- Resource documentation
  - `Microsoft.Authorization/roleAssignments`
  - `Microsoft.Authorization/roleDefinitions`
- Extension resources
- Scopes
  - Resource group
  - Subscription
  - Management group
  - Tenant
- Quickstart templates
  - Create a new role def via a subscription level deployment
  - Assign a role at subscription scope
  - Assign a role at tenant scope
  - Create a resourceGroup, apply a lock and RBAC
  - Create key vault, managed identity, and role assignment
- Community blog posts
  - Create role assignments for different scopes with Bicep, by Barbara Forbes

Create monitoring resources by using Bicep

9/21/2022 • 8 minutes to read • Edit Online

Azure has a comprehensive suite of tools that can monitor your applications and services. You can programmatically create your monitoring resources using Bicep to automate the creation of rules, diagnostic settings, and alerts when provisioning your Azure infrastructure.

Bringing your monitoring configuration into your Bicep code might seem unusual, considering that there are tools available inside the Azure portal to set up alert rules, diagnostic settings and dashboards.

However, alerts and diagnostic settings are essentially the same as your other infrastructure resources. By including them in your Bicep code, you can deploy and test your alerting resources as you would for other Azure resources.

If you use Git or another version control tool to manage your Bicep files, you also gain the benefit of having a history of your monitoring configuration so that you can see how alerts were set up and configured.

Log Analytics and Application Insights workspaces

You can create Log Analytics workspaces with the resource type Microsoft.OperationalInsights/workspaces and Application Insights workspaces with the type Microsoft.Insights/components. Both of these components are deployed to resource groups.

Diagnostic settings

Diagnostic settings enable you to configure Azure Monitor to export your logs and metrics to a number of destinations, including Log Analytics and Azure Storage.

When creating diagnostic settings in Bicep, remember that this resource is an extension resource, which means it's applied to another resource. You can create diagnostic settings in Bicep by using the resource type Microsoft.Insights/diagnosticSettings.

When creating diagnostic settings in Bicep, you need to apply the scope of the diagnostic setting. The diagnostic setting can be applied at the management, subscription, or resource group level. Use the scope property on this resource to set the scope for this resource.

Consider the following example:

```
param location string = resourceGroup().location
param appPlanName string = '${uniqueString(resourceGroup().id)}asp'
param logAnalyticsWorkspace string = '${uniqueString(resourceGroup().id)}la'

var appPlanSkuName = 'S1'

resource logAnalytics 'Microsoft.OperationalInsights/workspaces@2021-12-01-preview' existing = {
  name: logAnalyticsWorkspace
}

resource appServicePlan 'Microsoft.Web/serverfarms@2021-03-01' = {
  name: appPlanName
  location: location
  sku: {
    name: appPlanSkuName
    capacity: 1
  }
}

resource diagnosticLogs 'Microsoft.Insights/diagnosticSettings@2021-05-01-preview' = {
  name: appServicePlan.name
  scope: appServicePlan
  properties: {
    workspaceId: logAnalytics.id
    logs: [
      {
        category: 'AllMetrics'
        enabled: true
        retentionPolicy: {
          days: 30
          enabled: true
        }
      }
    ]
  }
}
```

In the preceding example, you create a diagnostic setting for the App Service plan and send those diagnostics to Log Analytics. You can use the `scope` property to define your App Service plan as the scope for your diagnostic setting, and use the `workspaceId` property to define the Log Analytics workspace to send the diagnostic logs to. You can also export diagnostic settings to Event Hubs and Azure Storage Accounts.

Log types differ between resources, so ensure that the logs you want to export are applicable for the resource you're using.

Activity log diagnostic settings

To use Bicep to configure diagnostic settings to export the Azure activity log, deploy a diagnostic setting resource at the subscription scope.

The following example shows how to export several activity log types to a Log Analytics workspace:

```
targetScope = 'subscription'

param logAnalyticsWorkspaceId string

var activityLogDiagnosticSettingsName = 'export-activity-log'

resource subscriptionActivityLog 'Microsoft.Insights/diagnosticSettings@2021-05-01-preview' = {
  name: activityLogDiagnosticSettingsName
  properties: {
    workspaceId: logAnalyticsWorkspaceId
    logs: [
      {
        category: 'Administrative'
        enabled: true
      }
      {
        category: 'Security'
        enabled: true
      }
      {
        category: 'ServiceHealth'
        enabled: true
      }
      {
        category: 'Alert'
        enabled: true
      }
      {
        category: 'Recommendation'
        enabled: true
      }
      {
        category: 'Policy'
        enabled: true
      }
      {
        category: 'Autoscale'
        enabled: true
      }
      {
        category: 'ResourceHealth'
        enabled: true
      }
    ]
  }
}
```

Alerts

Alerts proactively notify you when issues are found within your Azure infrastructure and applications by monitoring data within Azure Monitor. By configuring your monitoring and alerting configuration within your Bicep code, you can automate the creation of these alerts alongside the infrastructure that you are provisioning in Azure.

For more information about how alerts work in Azure see Overview of alerts in Microsoft Azure.

The following sections demonstrate how you can configure different types of alerts using Bicep code.

Action groups

To be notified when alerts have been triggered, you need to create an action group. An action group is a collection of notification preferences that are defined by the owner of an Azure subscription. Action groups are used to notify users that an alert has been triggered, or to trigger automated responses to alerts.

To create action groups in Bicep, you can use the type Microsoft.Insights/actionGroups. Here is an example:

```
param actionGroupName string = 'On-Call Team'
param location string = resourceGroup().location

var actionGroupEmail = 'oncallteam@contoso.com'

resource supportTeamActionGroup 'Microsoft.Insights/actionGroups@2021-09-01' = {
  name: actionGroupName
  location: location
  properties: {
    enabled: true
    groupShortName: actionGroupName
    emailReceivers: [
      {
        name: actionGroupName
        emailAddress: actionGroupEmail
        useCommonAlertSchema: true
      }
    ]
  }
}
```

The preceding example creates an action group that sends alerts to an email address, but you can also define action groups that send alerts to Event Hubs, Azure Functions, Logic Apps and more.

Alert processing rules

Alert processing rules (previously referred to as action rules) allow you to apply processing on alerts that have fired. You can create alert processing rules in Bicep using the type Microsoft.AlertsManagement/actionRules.

Each alert processing rule has a scope, which could be a list of one or more specific resources, a specific resource group or your entire Azure subscription. When you define alert processing rules in Bicep, you define a list of resource IDs in the *scope* property, which targets those resources for the alert processing rule.

```
param alertRuleName string = 'AlertRuleName'
param actionGroupName string = 'On-Call Team'
param location string = resourceGroup().location

resource actionGroup 'Microsoft.Insights/actionGroups@2021-09-01' existing = {
  name: actionGroupName
}

resource alertProcessingRule 'Microsoft.AlertsManagement/actionRules@2021-08-08' = {
  name: alertRuleName
  location: location
  properties: {
    actions: [
      {
        actionType: 'AddActionGroups'
        actionGroupIds: [
          actionGroup.id
        ]
      }
    ]
    conditions: [
      {
        field: 'MonitorService'
        operator: 'Equals'
        values: [
          'Azure Backup'
        ]
      }
    ]
    enabled: true
    scopes: [
      subscription().id
    ]
  }
}
```

In the preceding example, the `MonitorService` alert processing rule on Azure Backup Vault is defined, which is applied to the existing action group. This rule triggers alerts to the action group.

Log alert rules

Log alerts automatically run a Log Analytics query. The query which is used to evaluate resource logs at an interval that you define, determines if the results meet some criteria that you specify, and then fires an alert.

You can create log alert rules in Bicep by using the type Microsoft.Insights/scheduledQueryRules.

Metric alert rules

Metric alerts notify you when one of your metrics crosses a defined threshold. You can define a metric alert rule in your Bicep code by using the type Microsoft.Insights/metricAlerts.

Activity log alerts

The Azure activity log is a platform log in Azure that provides insights into events at the subscription level. This includes information such as when a resource in Azure is modified.

Activity log alerts are alerts that are activated when a new activity log event occurs that matches the conditions that are specified in the alert.

You can use the `scope` property within the type Microsoft.Insights/activityLogAlerts to create activity log alerts on a specific resource or a list of resources using the resource IDs as a prefix.

You define your alert rule conditions within the `condition` property and then configure the alert group to trigger these alerts to by using the `actionGroup` array. Here you can pass a single or multiple action groups to send activity log alerts to, depending on your requirements.

```
param activityLogAlertName string = '${uniqueString(resourceGroup().id)}-alert'
param actionGroupName string = 'adminactiongroup'

resource actionGroup 'Microsoft.Insights/actionGroups@2021-09-01' existing = {
  name: actionGroupName
}

resource activityLogAlert 'Microsoft.Insights/activityLogAlerts@2020-10-01' = {
  name: activityLogAlertName
  location: 'Global'
  properties: {
    condition: {
      allOf: [
        {
          field: 'category'
          equals: 'Administrative'
        }
        {
          field: 'operationName'
          equals: 'Microsoft.Resources/deployments/write'
        }
        {
          field: 'resourceType'
          equals: 'Microsoft.Resources/deployments'
        }
      ]
    }
    actions: {
      actionGroups: [
        {
          actionGroupId: actionGroup.id
        }
      ]
    }
    scopes: [
      subscription().id
    ]
  }
}
```

Resource health alerts

Azure Resource Health keeps you informed about the current and historical health status of your Azure resources. By creating your resource health alerts using Bicep, you can create and customize these alerts in bulk.

In Bicep, you can create resource health alerts with the type Microsoft.Insights/activityLogAlerts.

Resource health alerts can be configured to monitor events at the level of a subscription, resource group, or individual resource.

Consider the following example, where you create a resource health alert that reports on service health alerts. The alert is applied at the subscription level (using the `scope` property), and sends alerts to an existing action group:

```
param activityLogAlertName string = uniqueString(resourceGroup().id)
param actionGroupName string = 'oncallactiongroup'

resource actionGroup 'Microsoft.Insights/actionGroups@2021-09-01' existing = {
  name: actionGroupName
}

resource resourceHealthAlert 'Microsoft.Insights/activityLogAlerts@2020-10-01' = {
  name: activityLogAlertName
  location: 'global'
  properties: {
    condition: {
      allOf: [
        {
          field: 'category'
          equals: 'ServiceHealth'
        }
      ]
    }
    scopes: [
      subscription().id
    ]
    actions: {
      actionGroups: [
        {
          actionGroupId: actionGroup.id
        }
      ]
    }
  }
}
```

Smart detection alerts

Smart detection alerts warn you of potential performance problems and failure anomalies in your web application. You can create smart detection alerts in Bicep using the type Microsoft.AlertsManagement/smartDetectorAlertRules.

Dashboards

In Bicep, you can create portal dashboards by using the resource type Microsoft.Portal/dashboards.

For more information about creating dashboards with code, see Programmatically create an Azure Dashboard.

Autoscale rules

To create an autoscaling setting, you define these using the resource type Microsoft.Insights/autoscaleSettings.

To target the resource that you want to apply the autoscaling setting to, you need to provide the target resource identifier of the resource that the setting should be added to.

In this example, a *scale out* condition for the App Service plan based on the average CPU percentage over a 10 minute time period. If the App Service plan exceeds 70% average CPU consumption over 10 minutes, the autoscale engine scales out the plan by adding one instance.

```
param location string = resourceGroup().location
param appPlanName string = '${uniqueString(resourceGroup().id)}asp'

var appPlanSkuName = 'S1'

resource appServicePlan 'Microsoft.Web/serverfarms@2022-03-01' = {
  name: appPlanName
  location: location
  properties: {}
  sku: {
    name: appPlanSkuName
    capacity: 1
  }
}

resource scaleOutRule 'Microsoft.Insights/autoscalesettings@2022-10-01' = {
  name: appServicePlan.name
  location: location
  properties: {
    enabled: true
    profiles: [
      {
        name: 'Scale out condition'
        capacity: {
          maximum: '3'
          default: '1'
          minimum: '1'
        }
        rules: [
          {
            scaleAction: {
              type: 'ChangeCount'
              direction: 'Increase'
              cooldown: 'PT5M'
              value: '1'
            }
            metricTrigger: {
              metricName: 'CpuPercentage'
              operator: 'GreaterThan'
              timeAggregation: 'Average'
              threshold: 70
              metricResourceUri: appServicePlan.id
              timeWindow: 'PT10M'
              timeGrain: 'PT1M'
              statistic: 'Average'
            }
          }
        ]
      }
    ]
    targetResourceUri: appServicePlan.id
  }
}
```

NOTE

When defining autoscaling rules, keep best practices in mind to avoid issues when attempting to autoscale, such as flapping. For more information, see the following documentation on best practices for Autoscale.

Related resources

- Resource documentation
  - Microsoft.OperationalInsights/workspaces

- Microsoft.Insights/components
- Microsoft.Insights/diagnosticSettings
- Microsoft.Insights/actionGroups
- Microsoft.Insights/scheduledQueryRules
- Microsoft.Insights/metricAlerts
- Microsoft.Portal/dashboards
- Microsoft.Insights/activityLogAlerts
- Microsoft.AlertsManagement/smartDetectorAlertRules.
- Microsoft.Insights/autoscaleSettings

Manage secrets by using Bicep

9/21/2022 • 4 minutes to read • Edit Online

Deployments often require secrets to be stored and propagated securely throughout your Azure environment. Bicep and Azure provide many features to assist you with managing secrets in your deployments.

Avoid secrets where you can

In many situations, it's possible to avoid using secrets at all. Many Azure resources support managed identities, which enable them to authenticate and be authorized to access other resources within Azure, without you needing to handle or manage any credentials. Additionally, some Azure services can generate HTTPS certificates for you automatically, avoiding you handling certificates and private keys. Use managed identities and service-managed certificates wherever possible.

Use secure parameters

When you need to provide secrets to your Bicep deployments as parameters, use the `@secure()` decorator. When you mark a parameter as secure, Azure Resource Manager avoids logging the value or displaying it in the Azure portal, Azure CLI, or Azure PowerShell.

Avoid outputs for secrets

Don't use Bicep outputs for secure data. Outputs are logged to the deployment history, and anyone with access to the deployment can view the values of a deployment's outputs.

If you need to generate a secret within a Bicep deployment and make it available to the caller or to other resources, consider using one of the following approaches.

Look up secrets dynamically

Sometimes, you need to access a secret from one resource to configure another resource.

For example, you might have created a storage account in another deployment, and need to access its primary key to configure an Azure Functions app. You can use the `existing` keyword to obtain a strongly typed reference to the pre-created storage account, and then use the storage account's `listKeys()` method to create a connection string with the primary key:

> The following example is part of a larger example. For a Bicep file that you can deploy, see the complete file.

```
resource storageAccount 'Microsoft.Storage/storageAccounts@2021-06-01' existing = {
  name: storageAccountName
}

var storageAccountConnectionString =
'DefaultEndpointsProtocol=https;AccountName=${storageAccount.name};EndpointSuffix=${environment().suffixes.s
torage};AccountKey=${listKeys(storageAccount.id, storageAccount.apiVersion).keys[0].value}'

resource functionApp 'Microsoft.Web/sites@2021-02-01' = {
  name: functionAppName
  location: location
  kind: 'functionapp'
  properties: {
    httpsOnly: true
    serverFarmId: appServicePlan.id
    siteConfig: {
      appSettings: [
        {
          name: 'APPINSIGHTS_INSTRUMENTATIONKEY'
          value: applicationInsights.properties.InstrumentationKey
        }
        {
          name: 'AzureWebJobsStorage'
          value: storageAccountConnectionString
        }
        {
          name: 'FUNCTIONS_EXTENSION_VERSION'
          value: '~3'
        }
        {
          name: 'FUNCTIONS_WORKER_RUNTIME'
          value: 'dotnet'
        }
        {
          name: 'WEBSITE_CONTENTAZUREFILECONNECTIONSTRING'
          value: storageAccountConnectionString
        }
      ]
    }
  }
}
```

By using this approach, you avoid passing secrets into or out of your Bicep file.

You can also use this approach to store secrets in a key vault.

Use Key Vault

Azure Key Vault is designed to store and manage secure data. Use a key vault to manage your secrets, certificates, keys, and other data that needs to be protected and shared.

You can create and manage vaults and secrets by using Bicep. Define your vaults by creating a resource with the type `Microsoft.KeyVault/vaults`.

When you create a vault, you need to determine who and what can access its data. If you plan to read the vault's secrets from within a Bicep file, set the `enabledForTemplateDeployment` property to `true`.

Add secrets to a key vault

Secrets are a child resource and can be created by using the type `Microsoft.KeyVault/vaults/secrets`. The following example demonstrates how to create a vault and a secret:

> The following example is part of a larger example. For a Bicep file that you can deploy, see the complete file.

```
resource keyVault 'Microsoft.KeyVault/vaults@2019-09-01' = {
  name: keyVaultName
  location: location
  properties: {
    enabledForTemplateDeployment: true
    tenantId: tenant().tenantId
    accessPolicies: [
    ]
    sku: {
      name: 'standard'
      family: 'A'
    }
  }
}

resource keyVaultSecret 'Microsoft.KeyVault/vaults/secrets@2019-09-01' = {
  parent: keyVault
  name: 'MySecretName'
  properties: {
    value: 'MyVerySecretValue'
  }
}
```

TIP

When you use automated deployment pipelines, it can sometimes be challenging to determine how to bootstrap key vault secrets for your deployments. For example, if you've been provided with an API key to use when communicating with an external API, then the secret needs to be added to a vault before it can be used in your deployments.

When you work with secrets that come from a third party, you may need to manually add them to your vault, and then you can reference the secret for all subsequent uses.

Use a key vault with modules

When you use Bicep modules, you can provide secure parameters by using the `getSecret` function.

You can also reference a key vault defined in another resource group by using the `existing` and `scope` keywords together. In the following example, the Bicep file is deployed to a resource group named *Networking*. The value for the module's parameter *mySecret* is defined in a key vault named *contosonetworkingsecrets* located in the *Secrets* resource group:

```
resource networkingSecretsKeyVault 'Microsoft.KeyVault/vaults@2019-09-01' existing = {
  scope: resourceGroup('Secrets')
  name: 'contosonetworkingsecrets'
}

module exampleModule 'module.bicep' = {
  name: 'exampleModule'
  params: {
    mySecret: networkingSecretsKeyVault.getSecret('mySecret')
  }
}
```

Work with secrets in pipelines

When you deploy your Azure resources by using a pipeline, you need to take care to handle your secrets appropriately.

- Avoid storing secrets in your code repository. For example, don't add secrets to parameter files, or to your pipeline definition YAML files.

- In GitHub Actions, use encrypted secrets to store secure data. Use secret scanning to detect any accidental commits of secrets.
- In Azure Pipelines, use secret variables to store secure data.

Related resources

- Resource documentation
  - `Microsoft.KeyVault/vaults`
  - `Microsoft.KeyVault/vaults/secrets`
- Azure features
  - Managed identities
  - Azure Key Vault
- Bicep features
  - Secure parameters
  - Referencing existing resources
  - `getSecret` function
- Quickstart templates
  - Create a user-assigned managed identity and role assignments
  - Create an Azure Key Vault and a secret
  - Create a Key Vault and a list of secrets
  - Onboard a custom domain and managed TLS certificate with Front Door
- Azure Pipelines
  - Secret variables
- GitHub Actions
  - Encrypted secrets
  - Secret scanning

Create virtual network resources by using Bicep

9/21/2022 • 3 minutes to read • Edit Online

Many Azure deployments require networking resources to be deployed and configured. You can use Bicep to define your Azure networking resources.

Virtual networks and subnets

Define your virtual networks by creating a resource with the type `Microsoft.Network/virtualNetworks`.

Configure subnets by using the subnets property

Virtual networks contain subnets, which are logical groups of IP addresses within the virtual network. There are two ways to define subnets in Bicep: by using the `subnets` property on the virtual network resource, and by creating a child resource with type `Microsoft.Network/virtualNetworks/subnets`.

> **WARNING**
> Avoid defining subnets as child resources. This approach can result in downtime for your resources during subsequent deployments, or failed deployments.

It's best to define your subnets within the virtual network definition, as in this example:

> The following example is part of a larger example. For a Bicep file that you can deploy, see the complete file.

```
resource virtualNetwork 'Microsoft.Network/virtualNetworks@2019-11-01' = {
  name: virtualNetworkName
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        '10.0.0.0/16'
      ]
    }
    subnets: [
      {
        name: subnet1Name
        properties: {
          addressPrefix: '10.0.0.0/24'
        }
      }
      {
        name: subnet2Name
        properties: {
          addressPrefix: '10.0.1.0/24'
        }
      }
    ]
  }
}
```

Although both approaches enable you to define and create your subnets, there is an important difference. When you define subnets by using child resources, the first time your Bicep file is deployed, the virtual network is deployed. Then, after the virtual network deployment is complete, each subnet is deployed. This sequencing occurs because Azure Resource Manager deploys each individual resource separately.

When you redeploy the same Bicep file, the same deployment sequence occurs. However, the virtual network is deployed without any subnets configured on it because the `subnets` property is effectively empty. Then, after the virtual network is reconfigured, the subnet resources are redeployed, which re-establishes each subnet. In some situations, this behavior causes the resources within your virtual network to lose connectivity during your deployment. In other situations, Azure prevents you from modifying the virtual network and your deployment fails.

Access subnet resource IDs

You often need to refer to a subnet's resource ID. When you use the `subnets` property to define your subnet, you can use the `existing` keyword to also obtain a strongly typed reference to the subnet, and then access the subnet's `id` property:

> The following example is part of a larger example. For a Bicep file that you can deploy, see the complete file.

```
resource virtualNetwork 'Microsoft.Network/virtualNetworks@2019-11-01' = {
  name: virtualNetworkName
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        '10.0.0.0/16'
      ]
    }
    subnets: [
      {
        name: subnet1Name
        properties: {
          addressPrefix: '10.0.0.0/24'
        }
      }
      {
        name: subnet2Name
        properties: {
          addressPrefix: '10.0.1.0/24'
        }
      }
    ]
  }

  resource subnet1 'subnets' existing = {
    name: subnet1Name
  }

  resource subnet2 'subnets' existing = {
    name: subnet2Name
  }
}

output subnet1ResourceId string = virtualNetwork::subnet1.id
output subnet2ResourceId string = virtualNetwork::subnet2.id
```

Because this example uses the `existing` keyword to access the subnet resource, instead of defining the complete subnet resource, it doesn't have the risks outlined in the previous section.

You can also combine the `existing` and `scope` keywords to refer to a virtual network or subnet resource in another resource group.

Network security groups

Network security groups are frequently used to apply rules controlling the inbound and outbound flow of traffic from a subnet or network interface. It can become cumbersome to define large numbers of rules within a Bicep file, and to share rules across multiple Bicep files. Consider using the Shared variable file pattern when you work with complex or large network security groups.

Private endpoints

Private endpoints must be approved. In some situations, approval happens automatically. But in other scenarios, you need to approve the endpoint before it's usable.

Private endpoint approval is an operation, so you can't perform it directly within your Bicep code. However, you can use a deployment script to invoke the operation. Alternatively, you can invoke the operation outside of your Bicep file, such as in a pipeline script.

Related resources

- Resource documentation
  - `Microsoft.Network/virtualNetworks`
  - `Microsoft.Network/networkSecurityGroups`
- Child resources
- Quickstart templates
  - Create a Virtual Network with two Subnets
  - Virtual Network with diagnostic logs

Create Bicep parameter file

9/21/2022 • 4 minutes to read • Edit Online

Rather than passing parameters as inline values in your script, you can use a JSON file that contains the parameter values. This article shows how to create a parameter file that you use with a Bicep file.

Parameter file

A parameter file uses the following format:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "<first-parameter-name>": {
      "value": "<first-value>"
    },
    "<second-parameter-name>": {
      "value": "<second-value>"
    }
  }
}
```

Notice that the parameter file stores parameter values as plain text. This approach works for values that aren't sensitive, such as a resource SKU. Plain text doesn't work for sensitive values, such as passwords. If you need to pass a parameter that contains a sensitive value, store the value in a key vault. Instead of adding the sensitive value to your parameter file, retrieve it with the getSecret function. For more information, see Use Azure Key Vault to pass secure parameter value during Bicep deployment.

Define parameter values

To determine how to define the parameter names and values, open your Bicep file. Look at the parameters section of the Bicep file. The following examples show the parameters from a Bicep file.

```
@maxLength(11)
param storagePrefix string

@allowed([
  'Standard_LRS'
  'Standard_GRS'
  'Standard_ZRS'
  'Premium_LRS'
])
param storageAccountType string = 'Standard_LRS'
```

In the parameter file, the first detail to notice is the name of each parameter. The parameter names in your parameter file must match the parameter names in your Bicep file.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "storagePrefix": {
    },
    "storageAccountType": {
    }
  }
}
```

Notice the parameter type. The parameter types in your parameter file must use the same types as your Bicep file. In this example, both parameter types are strings.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "storagePrefix": {
      "value": ""
    },
    "storageAccountType": {
      "value": ""
    }
  }
}
```

Check the Bicep file for parameters with a default value. If a parameter has a default value, you can provide a value in the parameter file but it's not required. The parameter file value overrides the Bicep file's default value.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "storagePrefix": {
      "value": "" // This value must be provided.
    },
    "storageAccountType": {
      "value": "" // This value is optional. Bicep will use default value if not provided.
    }
  }
}
```

> **NOTE**
>
> For inline comments, you can use either // or /* ... */. In Visual Studio Code, save the parameter files with the **JSONC** file type, otherwise you will get an error message saying "Comments not permitted in JSON".

Check the Bicep's allowed values and any restrictions such as maximum length. Those values specify the range of values you can provide for a parameter. In this example, `storagePrefix` can have a maximum of 11 characters and `storageAccountType` must specify an allowed value.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "storagePrefix": {
      "value": "storage"
    },
    "storageAccountType": {
      "value": "Standard_ZRS"
    }
  }
}
```

> **NOTE**
> Your parameter file can only contain values for parameters that are defined in the Bicep file. If your parameter file contains extra parameters that don't match the Bicep file's parameters, you receive an error.

Parameter type formats

The following example shows the formats of different parameter types: string, integer, boolean, array, and object.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "exampleString": {
      "value": "test string"
    },
    "exampleInt": {
      "value": 4
    },
    "exampleBool": {
      "value": true
    },
    "exampleArray": {
      "value": [
        "value 1",
        "value 2"
      ]
    },
    "exampleObject": {
      "value": {
        "property1": "value1",
        "property2": "value2"
      }
    }
  }
}
```

Deploy Bicep file with parameter file

From Azure CLI, pass a local parameter file using `@` and the parameter file name. For example, `@storage.parameters.json`.

```
az deployment group create \
  --name ExampleDeployment \
  --resource-group ExampleGroup \
  --template-file storage.bicep \
  --parameters @storage.parameters.json
```

For more information, see Deploy resources with Bicep and Azure CLI. To deploy *.bicep* files you need Azure CLI version 2.20 or higher.

From Azure PowerShell, pass a local parameter file using the `TemplateParameterFile` parameter.

```
New-AzResourceGroupDeployment -Name ExampleDeployment -ResourceGroupName ExampleResourceGroup `
  -TemplateFile C:\MyTemplates\storage.bicep `
  -TemplateParameterFile C:\MyTemplates\storage.parameters.json
```

For more information, see Deploy resources with Bicep and Azure PowerShell. To deploy *.bicep* files you need Azure PowerShell version 5.6.0 or higher.

File name

The general naming convention for the parameter file is to include *parameters* in the Bicep file name. For example, if your Bicep file is named *azuredeploy.bicep*, your parameter file is named *azuredeploy.parameters.json*. This naming convention helps you see the connection between the Bicep file and the parameters.

To deploy to different environments, you create more than one parameter file. When you name the parameter files, identify their use such as development and production. For example, use *azuredeploy.parameters-dev.json* and *azuredeploy.parameters-prod.json* to deploy resources.

Parameter precedence

You can use inline parameters and a local parameter file in the same deployment operation. For example, you can specify some values in the local parameter file and add other values inline during deployment. If you provide values for a parameter in both the local parameter file and inline, the inline value takes precedence.

It's possible to use an external parameter file, by providing the URI to the file. When you use an external parameter file, you can't pass other values either inline or from a local file. All inline parameters are ignored. Provide all parameter values in the external file.

Parameter name conflicts

If your Bicep file includes a parameter with the same name as one of the parameters in the PowerShell command, PowerShell presents the parameter from your Bicep file with the postfix `FromTemplate`. For example, a parameter named `ResourceGroupName` in your Bicep file conflicts with the `ResourceGroupName` parameter in the New-AzResourceGroupDeployment cmdlet. You're prompted to provide a value for `ResourceGroupNameFromTemplate`. To avoid this confusion, use parameter names that aren't used for deployment commands.

Next steps

- For more information about how to define parameters in a Bicep file, see Parameters in Bicep.
- To get sensitive values, see Use Azure Key Vault to pass secure parameter value during deployment.

Use Azure Key Vault to pass secure parameter value during Bicep deployment

9/21/2022 • 5 minutes to read • Edit Online

Instead of putting a secure value (like a password) directly in your Bicep file or parameter file, you can retrieve the value from an Azure Key Vault during a deployment. When a module expects a `string` parameter with `secure:true` modifier, you can use the getSecret function to obtain a key vault secret. The value is never exposed because you only reference its key vault ID.

> **IMPORTANT**
>
> This article focuses on how to pass a sensitive value as a template parameter. When the secret is passed as a parameter, the key vault can exist in a different subscription than the resource group you're deploying to.
>
> This article doesn't cover how to set a virtual machine property to a certificate's URL in a key vault. For a quickstart template of that scenario, see Install a certificate from Azure Key Vault on a Virtual Machine.

Deploy key vaults and secrets

To access a key vault during Bicep deployment, set `enabledForTemplateDeployment` on the key vault to `true`.

If you already have a key vault, make sure it allows template deployments.

- Azure CLI
- PowerShell

```
az keyvault update  --name ExampleVault --enabled-for-template-deployment true
```

To create a new key vault and add a secret, use:

- Azure CLI
- PowerShell

```
az group create --name ExampleGroup --location centralus
az keyvault create \
  --name ExampleVault \
  --resource-group ExampleGroup \
  --location centralus \
  --enabled-for-template-deployment true
az keyvault secret set --vault-name ExampleVault --name "ExamplePassword" --value "hVFkk965BuUv"
```

As the owner of the key vault, you automatically have access to create secrets. If the user working with secrets isn't the owner of the key vault, grant access with:

- Azure CLI
- PowerShell

```
az keyvault set-policy \
  --upn <user-principal-name> \
  --name ExampleVault \
  --secret-permissions set delete get list
```

For more information about creating key vaults and adding secrets, see:

- Set and retrieve a secret by using CLI
- Set and retrieve a secret by using PowerShell
- Set and retrieve a secret by using the portal
- Set and retrieve a secret by using .NET
- Set and retrieve a secret by using Node.js

Grant access to the secrets

The user who deploys the Bicep file must have the `Microsoft.KeyVault/vaults/deploy/action` permission for the scope of the resource group and key vault. The Owner and Contributor roles both grant this access. If you created the key vault, you're the owner and have the permission.

The following procedure shows how to create a role with the minimum permission, and how to assign the user.

1. Create a custom role definition JSON file:

   ```
   {
     "Name": "Key Vault Bicep deployment operator",
     "IsCustom": true,
     "Description": "Lets you deploy a Bicep file with the access to the secrets in the Key Vault.",
     "Actions": [
       "Microsoft.KeyVault/vaults/deploy/action"
     ],
     "NotActions": [],
     "DataActions": [],
     "NotDataActions": [],
     "AssignableScopes": [
       "/subscriptions/00000000-0000-0000-0000-000000000000"
     ]
   }
   ```

   Replace "00000000-0000-0000-0000-000000000000" with the subscription ID.

2. Create the new role using the JSON file:

   - Azure CLI
   - PowerShell

   ```
   az role definition create --role-definition "<path-to-role-file>"
   az role assignment create \
     --role "Key Vault Bicep deployment operator" \
     --scope /subscriptions/<Subscription-id>/resourceGroups/<resource-group-name> \
     --assignee <user-principal-name>
   ```

   The samples assign the custom role to the user on the resource group level.

When using a key vault with the Bicep file 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.

Use getSecret function

You can use the getSecret function to obtain a key vault secret and pass the value to a `string` parameter of a module. The `getSecret` function can only be called on a `Microsoft.KeyVault/vaults` resource and can be used only with parameter with `@secure()` decorator.

The following Bicep file creates an Azure SQL server. The `adminPassword` parameter has a `@secure()` decorator.

```
param sqlServerName string
param adminLogin string

@secure()
param adminPassword string

resource sqlServer 'Microsoft.Sql/servers@2020-11-01-preview' = {
  name: sqlServerName
  location: resourceGroup().location
  properties: {
    administratorLogin: adminLogin
    administratorLoginPassword: adminPassword
    version: '12.0'
  }
}
```

Let's use the preceding Bicep file as a module given the file name is *sql.bicep* in the same directory as the main Bicep file.

The following Bicep file consumes the sql.bicep as a module. The Bicep file references an existing key vault, and calls the `getSecret` function to retrieve the key vault secret, and then passes the value as a parameter to the module.

```
param sqlServerName string
param adminLogin string

param subscriptionId string
param kvResourceGroup string
param kvName string

resource kv 'Microsoft.KeyVault/vaults@2019-09-01' existing = {
  name: kvName
  scope: resourceGroup(subscriptionId, kvResourceGroup )
}

module sql './sql.bicep' = {
  name: 'deploySQL'
  params: {
    sqlServerName: sqlServerName
    adminLogin: adminLogin
    adminPassword: kv.getSecret('vmAdminPassword')
  }
}
```

Reference secrets in parameter file

If you don't want to use a module, you can reference the key vault directly in the parameter file. The following image shows how the parameter file references the secret and passes that value to the Bicep file.

The following Bicep file deploys a SQL server that includes an administrator password. The password parameter is set to a secure string. But the Bicep doesn't specify where that value comes from.

```
param adminLogin string

@secure()
param adminPassword string

param sqlServerName string

resource sqlServer 'Microsoft.Sql/servers@2020-11-01-preview' = {
  name: sqlServerName
  location: resourceGroup().location
  properties: {
    administratorLogin: adminLogin
    administratorLoginPassword: adminPassword
    version: '12.0'
  }
}
```

Now, create a parameter file for the preceding Bicep file. In the parameter file, specify a parameter that matches the name of the parameter in the Bicep file. For the parameter value, reference the secret from the key vault. You reference the secret by passing the resource identifier of the key vault and the name of the secret:

In the following parameter file, the key vault secret must already exist, and you provide a static value for its resource ID.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "adminLogin": {
      "value": "exampleadmin"
    },
    "adminPassword": {
      "reference": {
        "keyVault": {
          "id": "/subscriptions/<subscription-id>/resourceGroups/<rg-
name>/providers/Microsoft.KeyVault/vaults/<vault-name>"
        },
        "secretName": "ExamplePassword"
      }
    },
    "sqlServerName": {
      "value": "<your-server-name>"
    }
  }
}
```

If you need to use a version of the secret other than the current version, include the `secretVersion` property.

```
"secretName": "ExamplePassword",
"secretVersion": "cd91b2b7e10e492ebb870a6ee0591b68"
```

Deploy the template and pass in the parameter file:

- Azure CLI
- PowerShell

```
az group create --name SqlGroup --location westus2
az deployment group create \
  --resource-group SqlGroup \
  --template-file <Bicep-file> \
  --parameters <parameter-file>
```

Next steps

- For general information about key vaults, see What is Azure Key Vault?
- For complete examples of referencing key secrets, see key vault examples on GitHub.
- For a Learn module that covers passing a secure value from a key vault, see Manage complex cloud deployments by using advanced ARM template features.

Use Bicep linter

9/21/2022 • 2 minutes to read • Edit Online

The Bicep linter checks Bicep files for syntax errors and best practice violations. The linter helps enforce coding standards by providing guidance during development. You can customize the best practices to use for checking the file.

Linter requirements

The linter is integrated into the Bicep CLI and the Bicep extension for Visual Studio Code. To use it, you must have version **0.4 or later**.

Default rules

The default set of linter rules is minimal and taken from arm-ttk test cases. The extension and Bicep CLI check the following rules, which are set to the warning level.

- adminusername-should-not-be-literal
- artifacts-parameters
- max-outputs
- max-params
- max-resources
- max-variables
- no-hardcoded-env-urls
- no-unnecessary-dependson
- no-unused-existing-resources
- no-unused-params
- no-unused-vars
- outputs-should-not-contain-secrets
- prefer-interpolation
- prefer-unquoted-property-names
- secure-parameter-default
- secure-secrets-in-params
- simplify-interpolation
- use-protectedsettings-for-commandtoexecute-secrets
- use-stable-resource-identifiers
- use-stable-vm-image

You can customize how the linter rules are applied. To overwrite the default settings, add a **bicepconfig.json** file and apply custom settings. For more information about applying those settings, see Add custom settings in the Bicep config file.

Use in Visual Studio Code

The following screenshot shows the linter in Visual Studio Code:

In the **PROBLEMS** pane, there are four errors, one warning, and one info message shown in the screenshot. The info message shows the Bicep configuration file that is used. It only shows this piece of information when you set **verbose** to **true** in the configuration file.

Hover your mouse cursor to one of the problem areas. Linter gives the details about the error or warning. Select the area, it also shows a blue light bulb:

Select either the light bulb or the **Quick fix** link to see the solution:

Select the solution to fix the issue automatically.

Use in Bicep CLI

The following screenshot shows the linter in the command line. The output from the build command shows any rule violations.

```
Windows PowerShell
PS D:\bicep\linter> bicep build ./linter.bicep
D:\bicep\linter\linter.bicep(1,1) : Info Bicep Linter Configuration: Custom bicepconfig.json file found (D:\bicep
\bicepconfig.json).
D:\bicep\linter\linter.bicep(4,7) : Error no-unused-params: Parameter "adminPassword" is declared but never used.
 [https://aka.ms/bicep/linter/no-unused-params]
D:\bicep\linter\linter.bicep(4,28) : Error secure-parameter-default: Secure parameters should not have hardcoded
defaults (except for empty or newGuid()). [https://aka.ms/bicep/linter/secure-parameter-default]
D:\bicep\linter\linter.bicep(6,5) : Error no-unused-vars: Variable "storeageAccountName" is declared but never us
ed. [https://aka.ms/bicep/linter/no-unused-vars]
D:\bicep\linter\linter.bicep(6,27) : Warning prefer-interpolation: Use string interpolation instead of the concat
 function. [https://aka.ms/bicep/linter/prefer-interpolation]
D:\bicep\linter\linter.bicep(9,27) : Error BCP082: The name "StoreageAccountName" does not exist in the current c
ontext. Did you mean "storeageAccountName"?
PS D:\bicep\linter>
```

You can integrate these checks as a part of your CI/CD pipelines. You can use a GitHub action to attempt a bicep build. Errors will fail the pipelines.

Silencing false positives

Sometimes a rule can have false positives. For example you may need to include a link to a blob storage directly without using the environment() function. In this case you can disable the warning for one line only, not the entire document, by adding `#disable-next-line <rule name>` before the line with the warning.

```
#disable-next-line no-hardcoded-env-urls //Direct download link to my toolset
scriptDownloadUrl: 'https://mytools.blob.core.windows.net/...'
```

It is good practice to add a comment explaining why the rule does not apply to this line.

Next steps

- For more information about customizing the linter rules, see Add custom settings in the Bicep config file.
- For more information about using Visual Studio Code and the Bicep extension, see Quickstart: Create Bicep files with Visual Studio Code.

Linter rule - admin user name should not be literal

9/21/2022 • 2 minutes to read • Edit Online

This rule finds when an admin user name is set to a literal value.

Linter rule code

Use the following value in the Bicep configuration file to customize rule settings:

`adminusername-should-not-be-literal`

Solution

Don't use a literal value or an expression that evaluates to a literal value. Instead, create a parameter for the user name and assign it to the admin user name.

The following example fails this test because the user name is a literal value.

```
resource vm 'Microsoft.Compute/virtualMachines@2020-12-01' = {
  name: 'name'
  location: location
  properties: {
    osProfile: {
      adminUsername: 'adminUsername'
    }
  }
}
```

The next example fails this test because the expression evaluates to a literal value when the default value is used.

```
var defaultAdmin = 'administrator'
resource vm 'Microsoft.Compute/virtualMachines@2020-12-01' = {
  name: 'name'
  location: location
  properties: {
    osProfile: {
      adminUsername: defaultAdmin
    }
  }
}
```

This example passes this test.

```
@secure()
param adminUsername string
param location string
resource vm 'Microsoft.Compute/virtualMachines@2020-12-01' = {
  name: 'name'
  location: location
  properties: {
    osProfile: {
      adminUsername: adminUsername
    }
  }
}
```

Next steps

For more information about the linter, see Use Bicep linter.

Linter rule - artifacts parameters

9/21/2022 • 2 minutes to read • Edit Online

This rule verifies whether the artifacts parameters are defined correctly. The following conditions must be met to pass the test:

- If you provide one parameter (either `_artifactsLocation` or `_artifactsLocationSasToken`), you must provide the other.
- `_artifactsLocation` must be a string.
- If `_artifactsLocation` has a default value, it must be either `deployment().properties.templateLink.uri` or a raw URL for its default value.
- `_artifactsLocationSasToken` must be a secure string.
- If `_artifactsLocationSasToken` has a default value, it must be an empty string.
- If a referenced module has an `_artifactsLocation` or `_artifactsLocationSasToken` parameter, a value must be passed in for those parameters, even if they have default values in the module.

Linter rule code

Use the following value in the Bicep configuration file to customize rule settings:

`artifacts-parameters`

Solution

The following example fails this test because `_artifactsLocationSasToken` is missing:

```
@description('The base URI where artifacts required by this template are located including a trailing
\'/\'')
param _artifactsLocation string = deployment().properties.templateLink.uri

...
```

The next example fails this test because `_artifactsLocation` must be either `deployment().properties.templateLink.uri` or a raw URL when the default value is provided, and the default value of `_artifactsLocationSasToken` is not an empty string.

```
@description('The base URI where artifacts required by this template are located including a trailing
\'/\'')
param _artifactsLocation string = 'something'

@description('SAS Token for accessing script path')
@secure()
param _artifactsLocationSasToken string = 'something'

...
```

This example passes this test.

```
@description('The base URI where artifacts required by this template are located including a trailing
\'/\'')
param _artifactsLocation string = deployment().properties.templateLink.uri

@description('SAS Token for accessing script path')
@secure()
param _artifactsLocationSasToken string = ''

...
```

Next steps

For more information about the linter, see Use Bicep linter.

Linter rule - max outputs

9/21/2022 • 2 minutes to read • Edit Online

This rule checks that the number of outputs does not exceed the ARM template limits.

Linter rule code

Use the following value in the Bicep configuration file to customize rule settings:

```
max-outputs
```

Solution

Reduce the number of outputs in your template.

Next steps

For more information about the linter, see Use Bicep linter.

Linter rule - max parameters

9/21/2022 • 2 minutes to read • Edit Online

This rule checks that the number of parameters does not exceed the ARM template limits.

Linter rule code

Use the following value in the Bicep configuration file to customize rule settings:

```
max-params
```

Solution

Reduce the number of parameters in your template.

Next steps

For more information about the linter, see Use Bicep linter.

Linter rule - max resources

9/21/2022 • 2 minutes to read • Edit Online

This rule checks that the number of resources does not exceed the ARM template limits.

Linter rule code

Use the following value in the Bicep configuration file to customize rule settings:

```
max-resources
```

Solution

Reduce the number of outputs in your template.

Next steps

For more information about the linter, see Use Bicep linter.

Linter rule - max variables

9/21/2022 • 2 minutes to read • Edit Online

This rule checks that the number of variables does not exceed the ARM template limits.

Linter rule code

Use the following value in the Bicep configuration file to customize rule settings:

```
max-variables
```

Solution

Reduce the number of variables in your template.

Next steps

For more information about the linter, see Use Bicep linter.

Linter rule - no hardcoded environment URL

9/21/2022 • 2 minutes to read • Edit Online

This rule finds any hard-coded URLs that vary by the cloud environment.

Linter rule code

Use the following value in the Bicep configuration file to customize rule settings:

`no-hardcoded-env-urls`

Solution

Instead of hard-coding URLs in your Bicep file, use the environment function to dynamically get these URLs during deployment. The environment function returns different URLs based on the cloud environment you're deploying to.

The following example fails this test because the URL is hardcoded.

```
var managementURL = 'https://management.azure.com'
```

The test also fails when used with concat or uri.

```
var galleryURL1 = concat('https://','gallery.azure.com')
var galleryURL2 = uri('gallery.azure.com','test')
```

You can fix it by replacing the hard-coded URL with the `environment()` function.

```
var galleryURL = environment().gallery
```

In some cases, you can fix it by getting a property from a resource you've deployed. For example, instead of constructing the endpoint for your storage account, retrieve it with `.properties.primaryEndpoints`.

```
param storageAccountName string

resource sa 'Microsoft.Storage/storageAccounts@2021-04-01' = {
  name: storageAccountName
  location: 'westus'
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
  properties: {
    accessTier: 'Hot'
  }
}

output endpoint string = sa.properties.primaryEndpoints.web
```

Configuration

By default, this rule uses the following settings for determining which URLs are disallowed.

```
"analyzers": {
  "core": {
    "verbose": false,
    "enabled": true,
    "rules": {
      "no-hardcoded-env-urls": {
        "level": "warning",
        "disallowedhosts": [
          "gallery.azure.com",
          "management.core.windows.net",
          "management.azure.com",
          "database.windows.net",
          "core.windows.net",
          "login.microsoftonline.com",
          "graph.windows.net",
          "trafficmanager.net",
          "datalake.azure.net",
          "azuredatalakestore.net",
          "azuredatalakeanalytics.net",
          "vault.azure.net",
          "api.loganalytics.io",
          "asazure.windows.net",
          "region.asazure.windows.net",
          "batch.core.windows.net"
        ],
        "excludedhosts": [
          "schema.management.azure.com"
        ]
      }
    }
  }
}
```

You can customize it by adding a bicepconfig.json file and applying new settings.

Next steps

For more information about the linter, see Use Bicep linter.

Linter rule - no hard-coded locations

9/21/2022 • 2 minutes to read • Edit Online

This rule finds uses of Azure location values that aren't parameterized.

Linter rule code

Use the following value in the Bicep configuration file to customize rule settings:

`no-hardcoded-location`

Solution

Template users may have limited access to regions where they can create resources. A hard-coded resource location might block users from creating a resource, thus preventing them from using the template. By providing a location parameter that defaults to the resource group location, users can use the default value when convenient but also specify a different location.

Rather than using a hard-coded string or variable value, use a parameter, the string 'global', or an expression (but not `resourceGroup().location` or `deployment().location`, see no-loc-expr-outside-params). Best practice suggests that to set your resources' locations, your template should have a string parameter named `location`. This parameter may default to the resource group or deployment location (`resourceGroup().location` or `deployment().location`).

The following example fails this test because the resource's `location` property uses a string literal:

```
resource stg 'Microsoft.Storage/storageAccounts@2021-02-01' = {
    location: 'westus'
}
```

You can fix it by creating a new `location` string parameter (which may optionally have a default value - resourceGroup().location is frequently used as a default):

```
param location string = resourceGroup().location
resource stg 'Microsoft.Storage/storageAccounts@2021-02-01' = {
    location: location
}
```

The following example fails this test because the resource's `location` property uses a variable with a string literal.

```
var location = 'westus'
resource stg 'Microsoft.Storage/storageAccounts@2021-02-01' = {
    location: location
}
```

You can fix it by turning the variable into a parameter:

```
  param location string = 'westus'
  resource stg 'Microsoft.Storage/storageAccounts@2021-02-01' = {
      location: location
  }
```

The following example fails this test because a string literal is being passed in to a module parameter that is in turn used for a resource's `location` property:

```
module m1 'module1.bicep' = {
  name: 'module1'
  params: {
    location: 'westus'
  }
}
```

where module1.bicep is:

```
param location string

resource storageaccount 'Microsoft.Storage/storageAccounts@2021-02-01' = {
  name: 'storageaccount'
  location: location
  kind: 'StorageV2'
  sku: {
    name: 'Premium_LRS'
  }
}
```

You can fix the failure by creating a new parameter for the value:

```
param location string // optionally with a default value
module m1 'module1.bicep' = {
  name: 'module1'
  params: {
    location: location
  }
}
```

Next steps

For more information about the linter, see Use Bicep linter.

Linter rule - no location expressions outside of parameter default values

9/21/2022 • 2 minutes to read • Edit Online

This rule finds `resourceGroup().location` or `deployment().location` used outside of a parameter default value.

Linter rule code

Use the following value in the Bicep configuration file to customize rule settings:

`no-loc-expr-outside-params`

Solution

`resourceGroup().location` and `deployment().location` should only be used as the default value of a parameter.

Template users may have limited access to regions where they can create resources. The expressions `resourceGroup().location` or `deployment().location` could block users if the resource group or deployment was created in a region the user can't access, thus preventing them from using the template.

Best practice suggests that to set your resources' locations, your template should have a string parameter named `location`. If you default the `location` parameter to `resourceGroup().location` or `deployment().location` instead of using these functions elsewhere in the template, users of the template can use the default value when convenient but also specify a different location when needed.

```
resource storageaccount 'Microsoft.Storage/storageAccounts@2021-02-01' = {
  location: resourceGroup().location
}
```

You can fix the failure by creating a `location` property that defaults to `resourceGroup().location` and use this new parameter instead:

```
param location string = resourceGroup().location

resource storageaccount 'Microsoft.Storage/storageAccounts@2021-02-01' = {
  location: location
}
```

The following example fails this test because `location` is using `resourceGroup().location` but isn't a parameter:

```
  var location = resourceGroup().location
```

You can fix the failure by turning the variable into a parameter:

```
  param location string  = resourceGroup().location
```

If you're using Azure PowerShell to deploy to a subscription, management group, or tenant, you should use a parameter name other than `location`. The New-AzDeployment, New-AzManagementGroupDeployment, and New-AzTenantDeployment commands have a parameter named `location`. This command parameter conflicts

with the parameter in your Bicep file. You can avoid this conflict by using a name such as `rgLocation`.

You can use `location` for a parameter name when deploying to a resource group, because New-AzResourceGroupDeployment doesn't have a parameter named `location`.

Next steps

For more information about the linter, see Use Bicep linter.

Linter rule - no unnecessary dependsOn entries

9/21/2022 • 2 minutes to read • Edit Online

This rule finds when an unnecessary dependsOn entry has been added to a resource or module declaration.

Linter rule code

Use the following value in the Bicep configuration file to customize rule settings:

`no-unnecessary-dependson`

Solution

To reduce confusion in your template, delete any dependsOn entries which are not necessary. Bicep automatically infers most resource dependencies as long as template expressions reference other resources via symbolic names rather than strings with hard-coded IDs or names.

The following example fails this test because the dependsOn entry `appServicePlan` is automatically inferred by Bicep implied by the expression `appServicePlan.id` (which references resource symbolic name `appServicePlan`) in the `serverFarmId` property's value.

```
resource appServicePlan 'Microsoft.Web/serverfarms@2020-12-01' = {
  name: 'name'
  location: resourceGroup().location
  sku: {
    name: 'F1'
    capacity: 1
  }
}

resource webApplication 'Microsoft.Web/sites@2018-11-01' = {
  name: 'name'
  location: resourceGroup().location
  properties: {
    serverFarmId: appServicePlan.id
  }
  dependsOn: [
    appServicePlan
  ]
}
```

You can fix it by removing the unnecessary dependsOn entry.

```
resource appServicePlan 'Microsoft.Web/serverfarms@2020-12-01' = {
  name: 'name'
  location: resourceGroup().location
  sku: {
    name: 'F1'
    capacity: 1
  }
}

resource webApplication 'Microsoft.Web/sites@2018-11-01' = {
  name: 'name'
  location: resourceGroup().location
  properties: {
    serverFarmId: appServicePlan.id
  }
}
```

Next steps

For more information about the linter, see Use Bicep linter.

Linter rule - no unused existing resources

9/21/2022 • 2 minutes to read • Edit Online

This rule finds existing resources that aren't referenced anywhere in the Bicep file.

Linter rule code

Use the following value in the Bicep configuration file to customize rule settings:

```
no-unused-existing-resources
```

Solution

To reduce confusion in your template, delete any existing resources that are defined but not used. This test finds any existing resource that isn't used anywhere in the template.

Next steps

For more information about the linter, see Use Bicep linter.

Linter rule - no unused parameters

9/21/2022 • 2 minutes to read • Edit Online

This rule finds parameters that aren't referenced anywhere in the Bicep file.

Linter rule code

Use the following value in the Bicep configuration file to customize rule settings:

`no-unused-params`

Solution

To reduce confusion in your template, delete any parameters that are defined but not used. This test finds any parameters that aren't used anywhere in the template. Eliminating unused parameters also makes it easier to deploy your template because you don't have to provide unnecessary values.

Next steps

For more information about the linter, see Use Bicep linter.

Linter rule - no unused variables

9/21/2022 • 2 minutes to read • Edit Online

This rule finds variables that aren't referenced anywhere in the Bicep file.

Linter rule code

Use the following value in the Bicep configuration file to customize rule settings:

`no-unused-vars`

Solution

To reduce confusion in your template, delete any variables that are defined but not used. This test finds any variables that aren't used anywhere in the template.

Next steps

For more information about the linter, see Use Bicep linter.

Linter rule - outputs should not contain secrets

9/21/2022 • 2 minutes to read • Edit Online

This rule finds possible exposure of secrets in a template's outputs.

Linter rule code

Use the following value in the Bicep configuration file to customize rule settings:
`outputs-should-not-contain-secrets`

Solution

Don't include any values in an output that could potentially expose secrets. For example, secure parameters of type secureString or secureObject, or `list*` functions such as listKeys. The output from a template is stored in the deployment history, so a user with read-only permissions could gain access to information otherwise not available with read-only permission. The following example fails because it includes a secure parameter in an output value.

```
@secure()
param secureParam string

output badResult string = 'this is the value ${secureParam}'
```

The following example fails because it uses a `list*` function in an output.

```
param storageName string
resource stg 'Microsoft.Storage/storageAccounts@2021-04-01' existing = {
  name: storageName
}

output badResult object = {
  value: stg.listKeys().keys[0].value
}
```

The following example fails because the output name contains 'password', indicating that it may contain a secret

```
output accountPassword string = '...'
```

To fix it, you will need to remove the secret data from the output. The recommended practice is to output the resourceId of the resource containing the secret and retrieve the secret when the resource needing the information is created or updated. Secrets may also be stored in KeyVault for more complex deployment scenarios.

The following example shows a secure pattern for retrieving a storageAccount key from a module.

```
output storageId string = stg.id
```

Which can be used in a subsequent deployment as shown in the following example

```
someProperty: listKeys(myStorageModule.outputs.storageId.value, '2021-09-01').keys[0].value
```

Silencing false positives

Sometimes this rule will alert on template outputs that do not actually contain secrets. For instance, not all `list*` functions actually return sensitive data. In these cases, you can disable the warning for this line by adding `#disable-next-line outputs-should-not-contain-secrets` before the line with the warning.

```
#disable-next-line outputs-should-not-contain-secrets // Does not contain a password
output notAPassword string = '...'
```

It is good practice to add a comment explaining why the rule does not apply to this line.

Next steps

For more information about the linter, see Use Bicep linter.

Linter rule - prefer interpolation

9/21/2022 • 2 minutes to read • Edit Online

This rule finds uses of the concat function that can be replaced by string interpolation.

Linter rule code

Use the following value in the Bicep configuration file to customize rule settings:

`prefer-interpolation`

Solution

Use string interpolation instead of the concat function.

The following example fails this test because the concat function is used.

```
param suffix string = '001'
var vnetName = concat('vnet-', suffix)
```

You can fix it by replacing concat with string interpolation. The following example passes this test.

```
param suffix string = '001'
var vnetName = 'vnet-${suffix}'
```

Next steps

For more information about the linter, see Use Bicep linter.

Linter rule - prefer unquoted property names

9/21/2022 • 2 minutes to read • Edit Online

This rule finds unnecessary single quotes where an object property name is declared and where an object property is dereferenced with array access.

In Bicep, quotes are optionally allowed when the object property keys contain numbers or special characters. For example, space, '-', or '.'. For more information, see Objects.

Linter rule code

Use the following value in the Bicep configuration file to customize rule settings:

`prefer-unquoted-property-names`

Solution

Quotes are not required in following code:

```
var obj = {
  newProp: {} // Property name is fine.
  'my-prop' : {} // Quotes are required.
  '1' : {} //  Quotes are required.
  'myProp': {} // Quotes are NOT required.
 }

var x0 = obj.newProp // Code is fine.
var x1 = obj['my-prop'] // Quotes and square brackets are required.
var x2 = obj['1'] // Quotes and square brackets are required.
var x3 = obj['myProp'] // Use obj.myProp instead.
```

You can fix it by removing the unnecessary quotes:

```
var obj = {
  newProp: {}
  'my-prop' : {}
  '1' : {}
  myProp: {}
 }


var x0 = obj.newProp
var x1 = obj['my-prop']
var x2 = obj['1']
var x3 = obj.myProp
```

Next steps

For more information about the linter, see Use Bicep linter.

Linter rule - use protectedSettings for commandToExecute secrets

9/21/2022 • 2 minutes to read • Edit Online

This rule finds possible exposure of secrets in the settings property of a custom script resource.

Linter rule code

Use the following value in the Bicep configuration file to customize rule settings:

`protect-commandtoexecute-secrets`

Solution

For custom script resources, the `commandToExecute` value should be placed under the `protectedSettings` property object instead of the `settings` property object if it includes secret data such as a password. For example, secret data could be found in secure parameters, `list*` functions such as listKeys, or in custom scripts arguments.

Don't use secret data in the `settings` object because it uses clear text. For more information, see Microsoft.Compute virtualMachines/extensions, Custom Script Extension for Windows, and Use the Azure Custom Script Extension Version 2 with Linux virtual machines.

The following example fails because `commandToExecute` is specified under `settings` and uses a secure parameter.

```
param vmName string
param location string
param fileUris string
param storageAccountName string

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-06-01' existing = {
  name: storageAccountName
}

resource customScriptExtension 'Microsoft.HybridCompute/machines/extensions@2019-08-02-preview' = {
  name: '${vmName}/CustomScriptExtension'
  location: location
  properties: {
    publisher: 'Microsoft.Compute'
    type: 'CustomScriptExtension'
    autoUpgradeMinorVersion: true
    settings: {
      fileUris: split(fileUris, ' ')
      commandToExecute: 'mycommand ${storageAccount.listKeys().keys[0].value}'
    }
  }
}
```

You can fix it by moving the commandToExecute property to the `protectedSettings` object.

```
param vmName string
param location string
param fileUris string
param storageAccountName string

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-06-01' existing = {
  name: storageAccountName
}

resource customScriptExtension 'Microsoft.HybridCompute/machines/extensions@2019-08-02-preview' = {
  name: '${vmName}/CustomScriptExtension'
  location: location
  properties: {
    publisher: 'Microsoft.Compute'
    type: 'CustomScriptExtension'
    autoUpgradeMinorVersion: true
    settings: {
      fileUris: split(fileUris, ' ')
    }
    protectedSettings: {
      commandToExecute: 'mycommand ${storageAccount.listKeys().keys[0].value}'
    }
  }
}
```

Next steps

For more information about the linter, see Use Bicep linter.

Linter rule - secure parameter default

9/21/2022 • 2 minutes to read • Edit Online

This rule finds hard-coded default values for secure parameters.

Linter rule code

Use the following value in the Bicep configuration file to customize rule settings:

`secure-parameter-default`

Solution

Don't provide a hard-coded default value for a secure parameter in your Bicep file, unless it's an empty string or an expression calling the newGuid() function.

You use the `@secure()` decorator on parameters that contain sensitive values, like passwords. When a parameter uses a secure decorator, the value of the parameter isn't logged or stored in the deployment history. This action prevents a malicious user from discovering the sensitive value.

However, when you provide a default value for a secured parameter, that value is discoverable by anyone who can access the template or the deployment history.

The following example fails this test because the parameter has a default value that is hard-coded.

```
@secure()
param adminPassword string = 'HardcodedPassword'
```

You can fix it by removing the default value.

```
@secure()
param adminPassword string
```

Or, by providing an empty string for the default value.

```
@secure()
param adminPassword string = ''
```

Or, by using `newGuid()` to generate the default value.

```
@secure()
param adminPassword string = newGuid()
```

Next steps

For more information about the linter, see Use Bicep linter.

Linter rule - secure secrets in parameters

9/21/2022 • 2 minutes to read • Edit Online

This rule finds parameters whose names look like secrets but without the secure decorator, for example: a parameter name contains the following keywords:

- password
- pwd
- secret
- accountkey
- acctkey

Linter rule code

Use the following value in the Bicep configuration file to customize rule settings:

`secure-secrets-in-params`

Solution

Use the secure decorator for the parameters that contain secrets. The secure decorator marks the parameter as secure. The value for a secure parameter isn't saved to the deployment history and isn't logged.

The following example fails this test because the parameter name may contain secrets.

```
param mypassword string
```

You can fix it by adding the secure decorator:

```
@secure()
param mypassword string
```

Silencing false positives

Sometimes this rule alerts on parameters that don't actually contain secrets. In these cases, you can disable the warning for this line by adding `#disable-next-line secure-secrets-in-params` before the line with the warning. For example:

```
#disable-next-line secure-secrets-in-params   // Doesn't contain a secret
param mypassword string
```

It's good practice to add a comment explaining why the rule doesn't apply to this line.

Next steps

For more information about the linter, see Use Bicep linter.

Linter rule - simplify interpolation

9/21/2022 • 2 minutes to read • Edit Online

This rule finds syntax that uses string interpolation when it isn't needed.

Linter rule code

Use the following value in the Bicep configuration file to customize rule settings:

`simplify-interpolation`

Solution

Remove any uses of string interpolation that isn't part of an expression to combine values.

The following example fails this test because it just references a parameter.

```
param AutomationAccountName string

resource AutomationAccount 'Microsoft.Automation/automationAccounts@2020-01-13-preview' = {
  name: '${AutomationAccountName}'
  ...
}
```

You can fix it by removing the string interpolation syntax.

```
param AutomationAccountName string

resource AutomationAccount 'Microsoft.Automation/automationAccounts@2020-01-13-preview' = {
  name: AutomationAccountName
  ...
}
```

Next steps

For more information about the linter, see Use Bicep linter.

Linter rule - use explicit values for module location parameters

9/21/2022 • 2 minutes to read • Edit Online

This rule finds module parameters that are used for resource locations and may inadvertently default to an unexpected value.

Linter rule code

Use the following value in the Bicep configuration file to customize rule settings:

`explicit-values-for-loc-params`

Solution

When you consume a module, any location-related parameters that have a default value should be assigned an explicit value. Location-related parameters include parameters that have a default value referencing `resourceGroup().location` or `deployment().location` and also any parameter that is referenced from a resource's location property.

A parameter that defaults to a resource group's or deployment's location is convenient when a bicep file is used as a main deployment template. However, when such a default value is used in a module, it may cause unexpected behavior if the main template's resources aren't located in the same region as the resource group.

Examples

The following example fails this test. Module `m1`'s parameter `location` isn't assigned an explicit value, so it will default to `resourceGroup().location`, as specified in *module1.bicep*. But using the resource group location may not be the intended behavior, since other resources in *main.bicep* might be created in a different location than the resource group's location.

main.bicep:

```
param location string = 'eastus'

module m1 'module1.bicep' = {
 name: 'm1'
}

resource storageaccount 'Microsoft.Storage/storageAccounts@2021-02-01' = {
  name: 'storageaccount'
  location: location
  kind: 'StorageV2'
  sku: {
    name: 'Standard_LRS'
  }
}
```

module1.bicep:

```
param location string = resourceGroup().location

resource stg 'Microsoft.Storage/storageAccounts@2021-02-01' = {
  name: 'stg'
  location: location
  kind: 'StorageV2'
  sku: {
    name: 'Premium_LRS'
  }
}
```

You can fix the failure by explicitly passing in a value for the module's `location` property:

main.bicep:

```
param location string = 'eastus'

module m1 'module1.bicep' = {
  name: 'm1'
  params: {
   location: location // An explicit value will override the default value specified in module1.bicep
  }
}

resource storageaccount 'Microsoft.Storage/storageAccounts@2021-02-01' = {
  name: 'storageaccount'
  location: location
  kind: 'StorageV2'
  sku: {
    name: 'Standard_LRS'
  }
}
```

Next steps

For more information about the linter, see Use Bicep linter.

Linter rule - use stable resource identifier

9/21/2022 • 2 minutes to read • Edit Online

Resource name shouldn't use a non-deterministic value. For example, `newGuid()` or `utcNow()` can't be used in resource name; resource name can't contain a parameter/variable whose default value uses `newGuid()` or `utcNow()`.

Linter rule code

Use the following value in the Bicep configuration file to customize rule settings:

`use-stable-resource-identifiers`

Solution

The following example fails this test because `utcNow()` is used in the resource name.

```
param location string = resourceGroup().location
param time string = utcNow()

resource sa 'Microsoft.Storage/storageAccounts@2021-09-01' = {
  name: 'store${toLower(time)}'
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
  properties: {
    accessTier: 'Hot'
  }
}
```

You can fix it by removing the `utcNow()` function from the example.

```
param location string = resourceGroup().location

resource sa 'Microsoft.Storage/storageAccounts@2021-09-01' = {
  name: 'store${uniqueString(resourceGroup().id)}'
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
  properties: {
    accessTier: 'Hot'
  }
}
```

Next steps

For more information about the linter, see Use Bicep linter.

Linter rule - use stable VM image

9/21/2022 • 2 minutes to read • Edit Online

Virtual machines shouldn't use preview images. This rule checks the following properties under "imageReference" and fails if any of them contain the string "preview":

- offer
- sku
- version

Linter rule code

Use the following value in the Bicep configuration file to customize rule settings:

`use-stable-vm-image`

Solution

The following example fails this test.

```
resource vm 'Microsoft.Compute/virtualMachines@2020-06-01' = {
  name: 'virtualMachineName'
  location: resourceGroup().location
  properties: {
    storageProfile: {
      imageReference: {
        offer: 'WindowsServer-preview'
        sku: '2019-Datacenter-preview'
        version: 'preview'
      }
    }
  }
}
```

You can fix it by using an image that does not contain the string `preview` in the imageReference.

```
resource vm 'Microsoft.Compute/virtualMachines@2020-06-01' = {
  name: 'virtualMachineName'
  location: resourceGroup().location
  properties: {
    storageProfile: {
      imageReference: {
        offer: 'WindowsServer'
        sku: '2019-Datacenter'
        version: 'latest'
      }
    }
  }
}
```

Next steps

For more information about the linter, see Use Bicep linter.

Bicep deployment what-if operation

9/21/2022 • 9 minutes to read • Edit Online

Before deploying a Bicep file, you can preview the changes that will happen. Azure Resource Manager provides the what-if operation to let you see how resources will change if you deploy the Bicep file. The what-if operation doesn't make any changes to existing resources. Instead, it predicts the changes if the specified Bicep file is deployed.

You can use the what-if operation with Azure PowerShell, Azure CLI, or REST API operations. What-if is supported for resource group, subscription, management group, and tenant level deployments.

Training resources

If you would rather learn about the what-if operation through step-by-step guidance, see Preview Azure deployment changes by using what-if.

Required permissions

To deploy a Bicep file or ARM template, you need write access on the resources you're deploying and access to all operations on the Microsoft.Resources/deployments resource type. For example, to deploy a virtual machine, you need `Microsoft.Compute/virtualMachines/write` and `Microsoft.Resources/deployments/*` permissions. The what-if operation has the same permission requirements.

For a list of roles and permissions, see Azure built-in roles.

Install Azure PowerShell module

To use what-if in PowerShell, you must have version **4.2 or later of the Az module**.

To install the module, use:

```
Install-Module -Name Az -Force
```

For more information about installing modules, see Install Azure PowerShell.

Install Azure CLI module

To use what-if in Azure CLI, you must have Azure CLI 2.14.0 or later. If needed, install the latest version of Azure CLI.

See results

When you use what-if in PowerShell or Azure CLI, the output includes color-coded results that help you see the different types of changes.

```
Resource and property changes are indicated with these symbols:
  - Delete
  + Create
  ~ Modify

The deployment will update the following scope:

Scope: /subscriptions/resourceGroups/ExampleGroup

  ~ Microsoft.Network/virtualNetworks/vnet-001 [2018-10-01]
    - tags.Owner:                      "Team A"
    + properties.enableVmProtection: false
    ~ properties.addressSpace.addressPrefixes: [
      - 0: "10.0.0.0/16"
      + 0: "10.0.0.0/15"
      ]
    ~ properties.subnets: [
      - 0:

          name:                      "subnet001"
          properties.addressPrefix: "10.0.0.0/24"

      ]

Resource changes: 1 to modify.
```

The text output is:

```
Resource and property changes are indicated with these symbols:
  - Delete
  + Create
  ~ Modify

The deployment will update the following scope:

Scope: /subscriptions/./resourceGroups/ExampleGroup

  ~ Microsoft.Network/virtualNetworks/vnet-001 [2018-10-01]
    - tags.Owner: "Team A"
    ~ properties.addressSpace.addressPrefixes: [
      - 0: "10.0.0.0/16"
      + 0: "10.0.0.0/15"
      ]
    ~ properties.subnets: [
      - 0:

          name:                      "subnet001"
          properties.addressPrefix: "10.0.0.0/24"

      ]

Resource changes: 1 to modify.
```

> **NOTE**
>
> The what-if operation can't resolve the reference function. Every time you set a property to a template expression that includes the reference function, what-if reports the property will change. This behavior happens because what-if compares the current value of the property (such as `true` or `false` for a boolean value) with the unresolved template expression. Obviously, these values will not match. When you deploy the Bicep file, the property will only change when the template expression resolves to a different value.

What-if commands

Azure PowerShell

To preview changes before deploying a Bicep file, use New-AzResourceGroupDeployment or New-AzSubscriptionDeployment. Add the `-Whatif` switch parameter to the deployment command.

- `New-AzResourceGroupDeployment -Whatif` for resource group deployments

- `New-AzSubscriptionDeployment -Whatif` and `New-AzDeployment -Whatif` for subscription level deployments

You can use the `-Confirm` switch parameter to preview the changes and get prompted to continue with the deployment.

- `New-AzResourceGroupDeployment -Confirm` for resource group deployments
- `New-AzSubscriptionDeployment -Confirm` and `New-AzDeployment -Confirm` for subscription level deployments

The preceding commands return a text summary that you can manually inspect. To get an object that you can programmatically inspect for changes, use Get-AzResourceGroupDeploymentWhatIfResult or Get-AzSubscriptionDeploymentWhatIfResult.

- `$results = Get-AzResourceGroupDeploymentWhatIfResult` for resource group deployments
- `$results = Get-AzSubscriptionDeploymentWhatIfResult` or `$results = Get-AzDeploymentWhatIfResult` for subscription level deployments

Azure CLI

To preview changes before deploying a Bicep file, use:

- az deployment group what-if for resource group deployments
- az deployment sub what-if for subscription level deployments
- az deployment mg what-if for management group deployments
- az deployment tenant what-if for tenant deployments

You can use the `--confirm-with-what-if` switch (or its short form `-c`) to preview the changes and get prompted to continue with the deployment. Add this switch to:

- az deployment group create
- az deployment sub create.
- az deployment mg create
- az deployment tenant create

For example, use `az deployment group create --confirm-with-what-if` or `-c` for resource group deployments.

The preceding commands return a text summary that you can manually inspect. To get a JSON object that you can programmatically inspect for changes, use the `--no-pretty-print` switch. For example, use `az deployment group what-if --no-pretty-print` for resource group deployments.

If you want to return the results without colors, open your Azure CLI configuration file. Set **no_color** to **yes**.

Azure REST API

For REST API, use:

- Deployments - What If for resource group deployments
- Deployments - What If At Subscription Scope for subscription deployments
- Deployments - What If At Management Group Scope for management group deployments
- Deployments - What If At Tenant Scope for tenant deployments.

Change types

The what-if operation lists six different types of changes:

- **Create**: The resource doesn't currently exist but is defined in the Bicep file. The resource will be created.
- **Delete**: This change type only applies when using complete mode for JSON template deployment. The resource exists, but isn't defined in the Bicep file. With complete mode, the resource will be deleted. Only resources that support complete mode deletion are included in this change type.

- **Ignore**: The resource exists, but isn't defined in the Bicep file. The resource won't be deployed or modified.
- **NoChange**: The resource exists, and is defined in the Bicep file. The resource will be redeployed, but the properties of the resource won't change. This change type is returned when ResultFormat is set to `FullResourcePayloads`, which is the default value.
- **Modify**: The resource exists, and is defined in the Bicep file. The resource will be redeployed, and the properties of the resource will change. This change type is returned when ResultFormat is set to `FullResourcePayloads`, which is the default value.
- **Deploy**: The resource exists, and is defined in the Bicep file. The resource will be redeployed. The properties of the resource may or may not change. The operation returns this change type when it doesn't have enough information to determine if any properties will change. You only see this condition when ResultFormat is set to `ResourceIdOnly`.

Result format

You control the level of detail that is returned about the predicted changes. You have two options:

- **FullResourcePayloads** - returns a list of resources that will change and details about the properties that will change
- **ResourceIdOnly** - returns a list of resources that will change

The default value is **FullResourcePayloads**.

For PowerShell deployment commands, use the `-WhatIfResultFormat` parameter. In the programmatic object commands, use the `ResultFormat` parameter.

For Azure CLI, use the `--result-format` parameter.

The following results show the two different output formats:

- Full resource payloads

```
Resource and property changes are indicated with these symbols:
  - Delete
  + Create
  ~ Modify

The deployment will update the following scope:

Scope: /subscriptions/./resourceGroups/ExampleGroup

  ~ Microsoft.Network/virtualNetworks/vnet-001 [2018-10-01]
    - tags.Owner: "Team A"
    ~ properties.addressSpace.addressPrefixes: [
      - 0: "10.0.0.0/16"
      + 0: "10.0.0.0/15"
      ]
    ~ properties.subnets: [
      - 0:

        name:                      "subnet001"
        properties.addressPrefix: "10.0.0.0/24"

      ]

Resource changes: 1 to modify.
```

- Resource ID only

```
  Resource and property changes are indicated with this symbol:
    ! Deploy

  The deployment will update the following scope:

  Scope: /subscriptions/./resourceGroups/ExampleGroup

    ! Microsoft.Network/virtualNetworks/vnet-001

  Resource changes: 1 to deploy.
```

Run what-if operation

Set up environment

To see how what-if works, let's runs some tests. First, deploy a Bicep file that creates a virtual network. You'll use this virtual network to test how changes are reported by what-if. Download a copy of the Bicep file.

```
resource vnet 'Microsoft.Network/virtualNetworks@2021-02-01' = {
  name: 'vnet-001'
  location: resourceGroup().location
  tags: {
    CostCenter: '12345'
    Owner: 'Team A'
  }
  properties: {
    addressSpace: {
      addressPrefixes: [
        '10.0.0.0/16'
      ]
    }
    enableVmProtection: false
    enableDdosProtection: false
    subnets: [
      {
        name: 'subnet001'
        properties: {
          addressPrefix: '10.0.0.0/24'
        }
      }
      {
        name: 'subnet002'
        properties: {
          addressPrefix: '10.0.1.0/24'
        }
      }
    ]
  }
}
```

To deploy the Bicep file, use:

- PowerShell
- Azure CLI

```
New-AzResourceGroup `
  -Name ExampleGroup `
  -Location centralus
New-AzResourceGroupDeployment `
  -ResourceGroupName ExampleGroup `
  -TemplateFile "what-if-before.bicep"
```

Test modification

After the deployment completes, you're ready to test the what-if operation. This time you deploy a Bicep file that changes the virtual network. It's missing one of the original tags, a subnet has been removed, and the address prefix has changed. Download a copy of the Bicep file.

```
resource vnet 'Microsoft.Network/virtualNetworks@2021-02-01' = {
  name: 'vnet-001'
  location: resourceGroup().location
  tags: {
    CostCenter: '12345'
  }
  properties: {
    addressSpace: {
      addressPrefixes: [
        '10.0.0.0/15'
      ]
    }
    enableVmProtection: false
    enableDdosProtection: false
    subnets: [
      {
        name: 'subnet002'
        properties: {
          addressPrefix: '10.0.1.0/24'
        }
      }
    ]
  }
}
```

To view the changes, use:

- PowerShell
- Azure CLI

```
New-AzResourceGroupDeployment `
  -Whatif `
  -ResourceGroupName ExampleGroup `
  -TemplateFile "what-if-after.bicep"
```

The what-if output appears similar to:

```
Resource and property changes are indicated with these symbols:
  - Delete
  + Create
  ~ Modify

The deployment will update the following scope:

Scope: /subscriptions/resourceGroups/ExampleGroup

  ~ Microsoft.Network/virtualNetworks/vnet-001 [2018-10-01]
    - tags.Owner:                      "Team A"
    + properties.enableVmProtection: false
    ~ properties.addressSpace.addressPrefixes: [
      - 0: "10.0.0.0/16"
      + 0: "10.0.0.0/15"
      ]
    ~ properties.subnets: [
      - 0:

          name:                      "subnet001"
          properties.addressPrefix: "10.0.0.0/24"

      ]

Resource changes: 1 to modify.
```

The text output is:

```
Resource and property changes are indicated with these symbols:
  - Delete
  + Create
  ~ Modify

The deployment will update the following scope:

Scope: /subscriptions/./resourceGroups/ExampleGroup

  ~ Microsoft.Network/virtualNetworks/vnet-001 [2018-10-01]
    - tags.Owner:                      "Team A"
    + properties.enableVmProtection: false
    ~ properties.addressSpace.addressPrefixes: [
      - 0: "10.0.0.0/16"
      + 0: "10.0.0.0/15"
      ]
    ~ properties.subnets: [
      - 0:

          name:                      "subnet001"
          properties.addressPrefix: "10.0.0.0/24"

      ]

Resource changes: 1 to modify.
```

Notice at the top of the output that colors are defined to indicate the type of changes.

At the bottom of the output, it shows the tag Owner was deleted. The address prefix changed from 10.0.0.0/16 to 10.0.0.0/15. The subnet named subnet001 was deleted. Remember these changes weren't deployed. You see a preview of the changes that will happen if you deploy the Bicep file.

Some of the properties that are listed as deleted won't actually change. Properties can be incorrectly reported as deleted when they aren't in the Bicep file, but are automatically set during deployment as default values. This result is considered "noise" in the what-if response. The final deployed resource will have the values set for the properties. As the what-if operation matures, these properties will be filtered out of the result.

Programmatically evaluate what-if results

Now, let's programmatically evaluate the what-if results by setting the command to a variable.

- PowerShell

- Azure CLI

```
$results = Get-AzResourceGroupDeploymentWhatIfResult `
  -ResourceGroupName ExampleGroup `
  --template-file "what-if-after.bicep"
```

You can see a summary of each change.

```
foreach ($change in $results.Changes)
{
  $change.Delta
}
```

Confirm deletion

To preview changes before deploying a Bicep file, use the confirm switch parameter with the deployment command. If the changes are as you expected, respond that you want the deployment to complete.

- PowerShell
- Azure CLI

```
New-AzResourceGroupDeployment `
  -ResourceGroupName ExampleGroup `
  -Confirm `
  -TemplateFile "what-if-after.bicep"
```

```
Resource and property changes are indicated with these symbols:
  - Delete
  + Create
  ~ Modify

The deployment will update the following scope:

Scope: /subscriptions/resourceGroups/ExampleGroup

  ~ Microsoft.Network/virtualNetworks/vnet-001 [2018-10-01]
    - tags.Owner:                      "Team A"
    + properties.enableVmProtection: false
    ~ properties.addressSpace.addressPrefixes: [
      - 0: "10.0.0.0/16"
      + 0: "10.0.0.0/15"
      ]
    ~ properties.subnets: [
      - 0:

          name:                      "subnet001"
          properties.addressPrefix: "10.0.0.0/24"

      ]

Resource changes: 1 to modify.
```

The text output is:

```
Resource and property changes are indicated with these symbols:
  - Delete
  + Create
  ~ Modify

The deployment will update the following scope:

Scope: /subscriptions/./resourceGroups/ExampleGroup

  ~ Microsoft.Network/virtualNetworks/vnet-001 [2018-10-01]
    - tags.Owner:                    "Team A"
    + properties.enableVmProtection: false
    ~ properties.addressSpace.addressPrefixes: [
      - 0: "10.0.0.0/16"
      + 0: "10.0.0.0/15"
      ]
    ~ properties.subnets: [
      - 0:

          name:                      "subnet001"
          properties.addressPrefix: "10.0.0.0/24"

      ]

Resource changes: 1 to modify.

Are you sure you want to execute the deployment?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):
```

You see the expected changes and can confirm that you want the deployment to run.

Clean up resources

When you no longer need the example resources, use Azure CLI or Azure PowerShell to delete the resource group.

- CLI
- PowerShell

```
az group delete --name ExampleGroup
```

SDKs

You can use the what-if operation through the Azure SDKs.

- For Python, use what-if.
- For Java, use DeploymentWhatIf Class.
- For .NET, use DeploymentWhatIf Class.

Next steps

- To use the what-if operation in a pipeline, see Test ARM templates with What-If in a pipeline.
- If you notice incorrect results from the what-if operation, please report the issues at https://aka.ms/whatifissues.
- For a Learn module that demonstrates using what-if, see Preview changes and validate Azure resources by using what-if and the ARM template test toolkit.

Deploy Bicep files from Visual Studio Code

9/21/2022 • 2 minutes to read • Edit Online

You can use Visual Studio Code with the Bicep extension to deploy a Bicep file. You can deploy to any scope. This article shows deploying to a resource group.

From an opened Bicep file in VS Code, there are there ways you can find the command:

- Right-click the Bicep file name from the Explorer pane, not the one under **OPEN EDITORS**:

- Right-click anywhere inside a Bicep file, and then select **Deploy Bicep File**.
- Select **Command Palette** from the **View** menu, and then select **Bicep: Deploy Bicep File**.

After you select the command, you follow the wizard to enter the values:

1. Sign in to Azure and select subscription.

2. Select or create a resource group.

3. Select a parameter file or select **None** to enter the parameter values.

4. If you choose **None**, enter the parameter values.

After you enter the values, you have the option to create a parameters file from values used in this deployment:

If you select **Yes**, a parameter file with the file name **<Bicep-file-name>.parameters.json** is created in the same folder.

For more information about VS Code commands, see Visual Studio Code.

Next steps

- For more information about deployment commands, see Deploy resources with Bicep and Azure CLI and Deploy resources with Bicep and Azure PowerShell.
- To preview changes before deploying a Bicep file, see Bicep deployment what-if operation.

How to deploy resources with Bicep and Azure CLI

9/21/2022 • 7 minutes to read • Edit Online

This article explains how to use Azure CLI with Bicep files to deploy your resources to Azure. If you aren't familiar with the concepts of deploying and managing your Azure solutions, see Bicep overview.

Prerequisites

You need a Bicep file to deploy. The file must be local.

You need Azure CLI and to be connected to Azure:

- **Install Azure CLI commands on your local computer.** To deploy Bicep files, you need Azure CLI version **2.20.0 or later**.
- **Connect to Azure by using az login**. If you have multiple Azure subscriptions, you might also need to run az account set.

Samples for the Azure CLI are written for the `bash` shell. To run this sample in Windows PowerShell or Command Prompt, you may need to change elements of the script.

If you don't have Azure CLI installed, you can use Azure Cloud Shell. For more information, see Deploy Bicep files from Azure Cloud Shell.

Required permissions

To deploy a Bicep file or ARM template, you need write access on the resources you're deploying and access to all operations on the Microsoft.Resources/deployments resource type. For example, to deploy a virtual machine, you need `Microsoft.Compute/virtualMachines/write` and `Microsoft.Resources/deployments/*` permissions. The what-if operation has the same permission requirements.

For a list of roles and permissions, see Azure built-in roles.

Deployment scope

You can target your deployment to a resource group, subscription, management group, or tenant. Depending on the scope of the deployment, you use different commands.

- To deploy to a **resource group**, use az deployment group create:

  ```
  az deployment group create --resource-group <resource-group-name> --template-file <path-to-bicep>
  ```

- To deploy to a **subscription**, use az deployment sub create:

  ```
  az deployment sub create --location <location> --template-file <path-to-bicep>
  ```

  For more information about subscription level deployments, see Create resource groups and resources at the subscription level.

- To deploy to a **management group**, use az deployment mg create:

```
az deployment mg create --location <location> --template-file <path-to-bicep>
```

For more information about management group level deployments, see Create resources at the management group level.

- To deploy to a **tenant**, use az deployment tenant create:

```
az deployment tenant create --location <location> --template-file <path-to-bicep>
```

For more information about tenant level deployments, see Create resources at the tenant level.

For every scope, the user deploying the Bicep file must have the required permissions to create resources.

Deploy local Bicep file

You can deploy a Bicep file from your local machine or one that is stored externally. This section describes deploying a local Bicep file.

If you're deploying to a resource group that doesn't exist, create the resource group. The name of the resource group can only include alphanumeric characters, periods, underscores, hyphens, and parenthesis. It can be up to 90 characters. The name can't end in a period.

```
az group create --name ExampleGroup --location "Central US"
```

To deploy a local Bicep file, use the `--template-file` parameter in the deployment command. The following example also shows how to set a parameter value.

```
az deployment group create \
  --name ExampleDeployment \
  --resource-group ExampleGroup \
  --template-file <path-to-bicep> \
  --parameters storageAccountType=Standard_GRS
```

The deployment can take a few minutes to complete. When it finishes, you see a message that includes the result:

```
"provisioningState": "Succeeded",
```

Deploy remote Bicep file

Currently, Azure CLI doesn't support deploying remote Bicep files. You can use Bicep CLI to build the Bicep file to a JSON template, and then load the JSON file to the remote location.

Parameters

To pass parameter values, you can use either inline parameters or a parameter file.

Inline parameters

To pass inline parameters, provide the values in `parameters`. For example, to pass a string and array to a Bicep file in a Bash shell, use:

```
az deployment group create \
  --resource-group testgroup \
  --template-file <path-to-bicep> \
  --parameters exampleString='inline string' exampleArray='("value1", "value2")'
```

If you're using Azure CLI with Windows Command Prompt (CMD) or PowerShell, pass the array in the format: `exampleArray="['value1','value2']"`.

You can also get the contents of file and provide that content as an inline parameter. Preface the file name with `@`.

```
az deployment group create \
  --resource-group testgroup \
  --template-file <path-to-bicep> \
  --parameters exampleString=@stringContent.txt exampleArray=@arrayContent.json
```

Getting a parameter value from a file is helpful when you need to provide configuration values. For example, you can provide cloud-init values for a Linux virtual machine.

The *arrayContent.json* format is:

```
[
    "value1",
    "value2"
]
```

To pass in an object, for example, to set tags, use JSON. For example, your Bicep file might include a parameter like this one:

```
    "resourceTags": {
      "type": "object",
      "defaultValue": {
        "Cost Center": "IT Department"
      }
    }
```

In this case, you can pass in a JSON string to set the parameter as shown in the following Bash script:

```
tags='{"Owner":"Contoso","Cost Center":"2345-324"}'
az deployment group create --name addstorage  --resource-group myResourceGroup \
--template-file $bicepFile \
--parameters resourceName=abcdef4556 resourceTags="$tags"
```

Use double quotes around the JSON that you want to pass into the object.

If you're using Azure CLI with Windows Command Prompt (CMD) or PowerShell, pass the object in the following format:

```
$tags="{'Owner':'Contoso','Cost Center':'2345-324'}"
az deployment group create --name addstorage  --resource-group myResourceGroup \
--template-file $bicepFile \
--parameters resourceName=abcdef4556 resourceTags=$tags
```

You can use a variable to contain the parameter values. In Bash, set the variable to all of the parameter values and add it to the deployment command.

```
params="prefix=start suffix=end"

az deployment group create \
  --resource-group testgroup \
  --template-file <path-to-bicep> \
  --parameters $params
```

However, if you're using Azure CLI with Windows Command Prompt (CMD) or PowerShell, set the variable to a JSON string. Escape the quotation marks:
`$params = '{ \"prefix\": {\"value\":\"start\"}, \"suffix\": {\"value\":\"end\"} }'`.

Parameter files

Rather than passing parameters as inline values in your script, you may find it easier to use a JSON file that contains the parameter values. The parameter file must be a local file. External parameter files aren't supported with Azure CLI. Bicep file uses JSON parameter files.

For more information about the parameter file, see Create Resource Manager parameter file.

To pass a local parameter file, specify the path and file name. The following example shows a parameter file named *storage.parameters.json*. The file is in the same directory where the command is run.

```
az deployment group create \
  --name ExampleDeployment \
  --resource-group ExampleGroup \
  --template-file storage.bicep \
  --parameters storage.parameters.json
```

Preview changes

Before deploying your Bicep file, you can preview the changes the Bicep file will make to your environment. Use the what-if operation to verify that the Bicep file makes the changes that you expect. What-if also validates the Bicep file for errors.

Deploy template specs

Currently, Azure CLI doesn't support creating template specs by providing Bicep files. However you can create a Bicep file with the Microsoft.Resources/templateSpecs resource to deploy a template spec. The Create template spec sample shows how to create a template spec in a Bicep file. You can also build your Bicep file to JSON by using the Bicep CLI, and then create a template spec with the JSON template.

Deployment name

When deploying a Bicep file, you can give the deployment a name. This name can help you retrieve the deployment from the deployment history. If you don't provide a name for the deployment, the name of the Bicep file is used. For example, if you deploy a Bicep file named `main.bicep` and don't specify a deployment name, the deployment is named `main`.

Every time you run a deployment, an entry is added to the resource group's deployment history with the deployment name. If you run another deployment and give it the same name, the earlier entry is replaced with the current deployment. If you want to maintain unique entries in the deployment history, give each deployment a unique name.

To create a unique name, you can assign a random number.

```
deploymentName='ExampleDeployment'$RANDOM
```

Or, add a date value.

```
deploymentName='ExampleDeployment'$(date +"%d-%b-%Y")
```

If you run concurrent deployments to the same resource group with the same deployment name, only the last deployment is completed. Any deployments with the same name that haven't finished are replaced by the last deployment. For example, if you run a deployment named `newStorage` that deploys a storage account named `storage1`, and at the same time run another deployment named `newStorage` that deploys a storage account named `storage2`, you deploy only one storage account. The resulting storage account is named `storage2`.

However, if you run a deployment named `newStorage` that deploys a storage account named `storage1`, and immediately after it completes you run another deployment named `newStorage` that deploys a storage account named `storage2`, then you have two storage accounts. One is named `storage1`, and the other is named `storage2`. But, you only have one entry in the deployment history.

When you specify a unique name for each deployment, you can run them concurrently without conflict. If you run a deployment named `newStorage1` that deploys a storage account named `storage1`, and at the same time run another deployment named `newStorage2` that deploys a storage account named `storage2`, then you have two storage accounts and two entries in the deployment history.

To avoid conflicts with concurrent deployments and to ensure unique entries in the deployment history, give each deployment a unique name.

Next steps

- To understand how to define parameters in your file, see Understand the structure and syntax of Bicep files.

Deploy resources with Bicep and Azure PowerShell

9/21/2022 • 6 minutes to read • Edit Online

This article explains how to use Azure PowerShell with Bicep files to deploy your resources to Azure. If you aren't familiar with the concepts of deploying and managing your Azure solutions, see Bicep overview.

Prerequisites

You need a Bicep file to deploy. The file must be local.

You need Azure PowerShell and to be connected to Azure:

- **Install Azure PowerShell cmdlets on your local computer.** To deploy Bicep files, you need Azure PowerShell version **5.6.0 or later**. For more information, see Get started with Azure PowerShell.
- **Install Bicep CLI.** Azure PowerShell doesn't automatically install the Bicep CLI. Instead, you must manually install the Bicep CLI.
- **Connect to Azure by using Connect-AzAccount**. If you have multiple Azure subscriptions, you might also need to run Set-AzContext. For more information, see Use multiple Azure subscriptions.

If you don't have PowerShell installed, you can use Azure Cloud Shell. For more information, see Deploy Bicep files from Azure Cloud Shell.

Required permissions

To deploy a Bicep file or ARM template, you need write access on the resources you're deploying and access to all operations on the Microsoft.Resources/deployments resource type. For example, to deploy a virtual machine, you need `Microsoft.Compute/virtualMachines/write` and `Microsoft.Resources/deployments/*` permissions. The what-if operation has the same permission requirements.

For a list of roles and permissions, see Azure built-in roles.

Deployment scope

You can target your deployment to a resource group, subscription, management group, or tenant. Depending on the scope of the deployment, you use different commands.

- To deploy to a **resource group**, use New-AzResourceGroupDeployment:

  ```
  New-AzResourceGroupDeployment -ResourceGroupName <resource-group-name> -TemplateFile <path-to-bicep>
  ```

- To deploy to a **subscription**, use New-AzSubscriptionDeployment, which is an alias of the `New-AzDeployment` cmdlet:

  ```
  New-AzSubscriptionDeployment -Location <location> -TemplateFile <path-to-bicep>
  ```

  For more information about subscription level deployments, see Create resource groups and resources at the subscription level.

- To deploy to a **management group**, use New-AzManagementGroupDeployment.

```
New-AzManagementGroupDeployment -ManagementGroupId <management-group-id> -Location <location> -
TemplateFile <path-to-bicep>
```

For more information about management group level deployments, see Create resources at the management group level.

- To deploy to a **tenant**, use New-AzTenantDeployment.

```
New-AzTenantDeployment -Location <location> -TemplateFile <path-to-bicep>
```

For more information about tenant level deployments, see Create resources at the tenant level.

For every scope, the user deploying the template must have the required permissions to create resources.

Deploy local Bicep file

You can deploy a Bicep file from your local machine or one that is stored externally. This section describes deploying a local Bicep file.

If you're deploying to a resource group that doesn't exist, create the resource group. The name of the resource group can only include alphanumeric characters, periods, underscores, hyphens, and parenthesis. It can be up to 90 characters. The name can't end in a period.

```
New-AzResourceGroup -Name ExampleGroup -Location "Central US"
```

To deploy a local Bicep file, use the `-TemplateFile` parameter in the deployment command.

```
New-AzResourceGroupDeployment `
  -Name ExampleDeployment `
  -ResourceGroupName ExampleGroup `
  -TemplateFile <path-to-bicep>
```

The deployment can take several minutes to complete.

Deploy remote Bicep file

Currently, Azure PowerShell doesn't support deploying remote Bicep files. Use Bicep CLI to build the Bicep file to a JSON template, and then load the JSON file to the remote location.

Parameters

To pass parameter values, you can use either inline parameters or a parameter file.

Inline parameters

To pass inline parameters, provide the names of the parameter with the `New-AzResourceGroupDeployment` command. For example, to pass a string and array to a Bicep file, use:

```
$arrayParam = "value1", "value2"
New-AzResourceGroupDeployment -ResourceGroupName testgroup `
  -TemplateFile <path-to-bicep> `
  -exampleString "inline string" `
  -exampleArray $arrayParam
```

You can also get the contents of file and provide that content as an inline parameter.

```
$arrayParam = "value1", "value2"
New-AzResourceGroupDeployment -ResourceGroupName testgroup `
  -TemplateFile <path-to-bicep> `
  -exampleString $(Get-Content -Path c:\MyTemplates\stringcontent.txt -Raw) `
  -exampleArray $arrayParam
```

Getting a parameter value from a file is helpful when you need to provide configuration values. For example, you can provide cloud-init values for a Linux virtual machine.

If you need to pass in an array of objects, create hash tables in PowerShell and add them to an array. Pass that array as a parameter during deployment.

```
$hash1 = @{ Name = "firstSubnet"; AddressPrefix = "10.0.0.0/24"}
$hash2 = @{ Name = "secondSubnet"; AddressPrefix = "10.0.1.0/24"}
$subnetArray = $hash1, $hash2
New-AzResourceGroupDeployment -ResourceGroupName testgroup `
  -TemplateFile <path-to-bicep> `
  -exampleArray $subnetArray
```

Parameter files

Rather than passing parameters as inline values in your script, you may find it easier to use a JSON file that contains the parameter values. The parameter file can be a local file or an external file with an accessible URI. Bicep file uses JSON parameter files.

For more information about the parameter file, see Create Resource Manager parameter file.

To pass a local parameter file, use the `TemplateParameterFile` parameter:

```
New-AzResourceGroupDeployment -Name ExampleDeployment -ResourceGroupName ExampleResourceGroup `
  -TemplateFile c:\BicepFiles\storage.bicep `
  -TemplateParameterFile c:\BicepFiles\storage.parameters.json
```

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

```
New-AzResourceGroupDeployment -Name ExampleDeployment -ResourceGroupName ExampleResourceGroup `
  -TemplateFile c:\BicepFiles\storage.bicep `
  -TemplateParameterUri https://raw.githubusercontent.com/Azure/azure-quickstart-
templates/master/quickstarts/microsoft.storage/storage-account-create/azuredeploy.parameters.json
```

Preview changes

Before deploying your Bicep file, you can preview the changes the Bicep file will make to your environment. Use the what-if operation to verify that the Bicep file makes the changes that you expect. What-if also validates the Bicep file for errors.

Deploy template specs

Currently, Azure PowerShell doesn't support creating template specs by providing Bicep files. However you can create a Bicep file with the Microsoft.Resources/templateSpecs resource to deploy a template spec. The Create template spec sample shows how to create a template spec in a Bicep file. You can also build your Bicep file to JSON by using the Bicep CLI, and then create a template spec with the JSON template.

Deployment name

When deploying a Bicep file, you can give the deployment a name. This name can help you retrieve the deployment from the deployment history. If you don't provide a name for the deployment, the name of the Bicep file is used. For example, if you deploy a Bicep named `main.bicep` and don't specify a deployment name, the deployment is named `main`.

Every time you run a deployment, an entry is added to the resource group's deployment history with the deployment name. If you run another deployment and give it the same name, the earlier entry is replaced with the current deployment. If you want to maintain unique entries in the deployment history, give each deployment a unique name.

To create a unique name, you can assign a random number.

```
$suffix = Get-Random -Maximum 1000
$deploymentName = "ExampleDeployment" + $suffix
```

Or, add a date value.

```
$today=Get-Date -Format "MM-dd-yyyy"
$deploymentName="ExampleDeployment"+"$today"
```

If you run concurrent deployments to the same resource group with the same deployment name, only the last deployment is completed. Any deployments with the same name that haven't finished are replaced by the last deployment. For example, if you run a deployment named `newStorage` that deploys a storage account named `storage1`, and at the same time run another deployment named `newStorage` that deploys a storage account named `storage2`, you deploy only one storage account. The resulting storage account is named `storage2`.

However, if you run a deployment named `newStorage` that deploys a storage account named `storage1`, and immediately after it completes you run another deployment named `newStorage` that deploys a storage account named `storage2`, then you have two storage accounts. One is named `storage1`, and the other is named `storage2`. But, you only have one entry in the deployment history.

When you specify a unique name for each deployment, you can run them concurrently without conflict. If you run a deployment named `newStorage1` that deploys a storage account named `storage1`, and at the same time run another deployment named `newStorage2` that deploys a storage account named `storage2`, then you have two storage accounts and two entries in the deployment history.

To avoid conflicts with concurrent deployments and to ensure unique entries in the deployment history, give each deployment a unique name.

Next steps

- To understand how to define parameters in your file, see Understand the structure and syntax of Bicep files.

Deploy Bicep files from Azure Cloud Shell

9/21/2022 • 2 minutes to read • Edit Online

You can use Azure Cloud Shell to deploy a Bicep file. Currently you can only deploy a local Bicep file from the Cloud Shell.

You can deploy to any scope. This article shows deploying to a resource group.

Deploy local Bicep file

To deploy a local Bicep file, you must first upload your Bicep file to your Cloud Shell session.

1. Sign in to the Cloud Shell.
2. Select either **PowerShell** or **Bash**.

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

4. Select the Bicep file you want to upload, and then select **Open**.
5. To deploy the Bicep file, use the following commands:

   - Azure CLI
   - PowerShell

```
az group create --name ExampleGroup --location "South Central US"
az deployment group create \
  --resource-group ExampleGroup \
  --template-file azuredeploy.bicep \
  --parameters storageAccountType=Standard_GRS
```

Next steps

- For more information about deployment commands, see Deploy resources with Bicep and Azure CLI and Deploy resources with Bicep and Azure PowerShell.
- To preview changes before deploying a Bicep file, see Bicep deployment what-if operation.

Configure your Bicep environment

9/21/2022 • 2 minutes to read • Edit Online

Bicep supports a configuration file named `bicepconfig.json`. Within this file, you can add values that customize your Bicep development experience. If you don't add this file, Bicep uses default values.

To customize values, create this file in the directory where you store Bicep files. You can add `bicepconfig.json` files in multiple directories. The configuration file closest to the Bicep file in the directory hierarchy is used.

To create a `bicepconfig.json` file in Visual Studio Code, see Visual Studio Code.

Available settings

When working with modules, you can add aliases for module paths. These aliases simplify your Bicep file because you don't have to repeat complicated paths. For more information, see Add module settings to Bicep config.

The Bicep linter checks Bicep files for syntax errors and best practice violations. You can override the default settings for the Bicep file validation by modifying `bicepconfig.json`. For more information, see Add linter settings to Bicep config.

You can also configure the credential precedence for authenticating to Azure from Bicep CLI and Visual Studio Code. The credentials are used to publish modules to registries and to restore external modules to the local cache when using the insert resource function.

Credential precedence

You can configure the credential precedence for authenticating to the registry. By default, Bicep uses the credentials from the user authenticated in Azure CLI or Azure PowerShell. To customize the credential precedence, add `cloud` and `credentialPrecedence` elements to the config file.

```
{
    "cloud": {
      "credentialPrecedence": [
        "AzureCLI",
        "AzurePowerShell"
      ]
    }
}
```

The available credential types are:

- AzureCLI
- AzurePowerShell
- Environment
- ManagedIdentity
- VisualStudio
- VisualStudioCode

Intellisense

The Bicep extension for Visual Studio Code supports intellisense for your `bicepconfig.json` file. Use the

intellisense to discover available properties and values.

Next steps

- Add module settings in Bicep config
- Add linter settings to Bicep config
- Learn about the Bicep linter

Add linter settings in the Bicep config file

9/21/2022 • 2 minutes to read • Edit Online

In a **bicepconfig.json** file, you can customize validation settings for the Bicep linter. The linter uses these settings when evaluating your Bicep files for best practices.

This article describes the settings that are available for working with the Bicep linter.

Customize linter

The linter settings are available under the `analyzers` element. You can enable or disable the linter, supply rule-specific values, and set the level of rules.

The following example shows the rules that are available for configuration.

```
{
  "analyzers": {
    "core": {
      "enabled": true,
      "rules": {
        "adminusername-should-not-be-literal": {
          "level": "warning"
        },
        "artifacts-parameters": {
          "level": "warning"
        },
        "no-hardcoded-env-urls": {
          "level": "warning"
        },
        "no-unnecessary-dependson": {
          "level": "warning"
        },
        "no-unused-existing-resources": {
          "level": "warning"
        },
        "no-unused-params": {
          "level": "warning"
        },
        "no-unused-vars": {
          "level": "warning"
        },
        "outputs-should-not-contain-secrets": {
          "level": "warning"
        },
        "prefer-interpolation": {
          "level": "warning"
        },
        "prefer-unquoted-property-names": {
          "level": "warning"
        },
        "protect-commandtoexecute-secrets": {
          "level": "warning"
        },
        "secure-parameter-default": {
          "level": "warning"
        },
        "simplify-interpolation": {
          "level": "warning"
        },
        "secure-secrets-in-params": {
          "level": "warning"
        },
        "use-stable-resource-identifiers": {
          "level": "warning"
        },
        "use-stable-vm-image": {
          "level": "warning"
        }
      }
    }
  }
}
```

The properties are:

- **enabled**: specify **true** for enabling linter, **false** for disabling linter.
- **verbose**: specify **true** to show the bicepconfig.json file used by Visual Studio Code.
- **rules**: specify rule-specific values. Each rule has a level that determines how the linter responds when a violation is found.

The available values for **level** are:

LEVEL	BUILD-TIME BEHAVIOR	EDITOR BEHAVIOR
`Error`	Violations appear as Errors in command-line build output, and causes the build to fail.	Offending code is underlined with a red squiggle and appears in Problems tab.
`Warning`	Violations appear as Warnings in command-line build output, but they don't cause the build to fail.	Offending code is underlined with a yellow squiggle and appears in Problems tab.
`Info`	Violations don't appear in the command-line build output.	Offending code is underlined with a blue squiggle and appears in Problems tab.
`Off`	Suppressed completely.	Suppressed completely.

Environment URLs

For the rule about hardcoded environment URLs, you can customize which URLs are checked. By default, the following settings are applied:

```
{
  "analyzers": {
    "core": {
      "enabled": true,
      "rules": {
        "no-hardcoded-env-urls": {
          "level": "warning",
          "disallowedhosts": [
            "management.core.windows.net",
            "gallery.azure.com",
            "management.core.windows.net",
            "management.azure.com",
            "database.windows.net",
            "core.windows.net",
            "login.microsoftonline.com",
            "graph.windows.net",
            "trafficmanager.net",
            "vault.azure.net",
            "datalake.azure.net",
            "azuredatalakestore.net",
            "azuredatalakeanalytics.net",
            "vault.azure.net",
            "api.loganalytics.io",
            "api.loganalytics.iov1",
            "asazure.windows.net",
            "region.asazure.windows.net",
            "api.loganalytics.iov1",
            "api.loganalytics.io",
            "asazure.windows.net",
            "region.asazure.windows.net",
            "batch.core.windows.net"
          ],
          "excludedhosts": [
            "schema.management.azure.com"
          ]
        }
      }
    }
  }
}
```

Next steps

- Configure your Bicep environment
- Add module settings in Bicep config
- Learn about the Bicep linter

Add module settings in the Bicep config file

9/21/2022 • 2 minutes to read • Edit Online

In a **bicepconfig.json** file, you can create aliases for module paths and configure credential precedence for restoring a module.

This article describes the settings that are available for working with modules.

Aliases for modules

To simplify the path for linking to modules, create aliases in the config file. An alias refers to either a module registry or a resource group that contains template specs.

The config file has a property for `moduleAliases`. This property contains all of the aliases you define. Under this property, the aliases are divided based on whether they refer to a registry or a template spec.

To create an alias for a **Bicep registry**, add a `br` property. To add an alias for a **template spec**, use the `ts` property.

```
{
  "moduleAliases": {
    "br": {
      <add-registry-aliases>
    },
    "ts": {
      <add-template-specs-aliases>
    }
  }
}
```

Within the `br` property, add as many aliases as you need. For each alias, give it a name and the following properties:

- **registry** (required): registry login server name
- **modulePath** (optional): registry repository where the modules are stored

Within the `ts` property, add as many aliases as you need. For each alias, give it a name and the following properties:

- **subscription** (required): the subscription ID that hosts the template specs
- **resourceGroup** (required): the name of the resource group that contains the template specs

The following example shows a config file that defines two aliases for a module registry, and one alias for a resource group that contains template specs.

```
{
  "moduleAliases": {
    "br": {
      "ContosoRegistry": {
        "registry": "contosoregistry.azurecr.io"
      },
      "CoreModules": {
        "registry": "contosoregistry.azurecr.io",
        "modulePath": "bicep/modules/core"
      }
    },
    "ts": {
      "CoreSpecs": {
        "subscription": "00000000-0000-0000-0000-000000000000",
        "resourceGroup": "CoreSpecsRG"
      }
    }
  }
}
```

When using an alias in the module reference, you must use the formats:

```
br/<alias>:<file>:<tag>
ts/<alias>:<file>:<tag>
```

Define your aliases to the folder or resource group that contains modules, not the file itself. The file name must be included in the reference to the module.

Without the aliases, you would link to a module in a registry with the full path.

```
module stgModule 'br:contosoregistry.azurecr.io/bicep/modules/core/storage:v1' = {
```

With the aliases, you can simplify the link by using the alias for the registry.

```
module stgModule 'br/ContosoRegistry:bicep/modules/core/storage:v1' = {
```

Or, you can simplify the link by using the alias that specifies the registry and module path.

```
module stgModule  'br/CoreModules:storage:v1' = {
```

For a template spec, use:

```
module stgModule  'ts/CoreSpecs:storage:v1' = {
```

An alias has been predefined for the public module registry. To reference a public module, you can use the format:

```
br/public:<file>:<tag>
```

You can override the public module registry alias definition in the bicepconfig.json file:

```
{
  "moduleAliases": {
    "br": {
      "public": {
        "registry": "<your_module_registry>",
        "modulePath": "<optional_module_path>"
      }
    }
  }
}
```

Credentials for publishing/restoring modules

To publish modules to a private module registry or to restore external modules to the local cache, the account must have the correct permissions to access the registry. You can configure the credential precedence for authenticating to the registry. By default, Bicep uses the credentials from the user authenticated in Azure CLI or Azure PowerShell. To customize the credential precedence, see Add credential precedence to Bicep config.

Next steps

- Configure your Bicep environment
- Add linter settings to Bicep config
- Learn about modules

Migrate to Bicep

9/21/2022 • 8 minutes to read • Edit Online

There are a number of benefits to defining your Azure resources in Bicep including: simpler syntax, modularization, automatic dependency management, type validation and IntelliSense, and an improved authoring experience.

When you have existing JSON Azure Resource Manager templates (ARM templates) and/or deployed resources, and you want to safely migrate those to Bicep, we suggest following a recommended workflow, consisting of five phases:

The first step in the process is to capture an initial representation of your Azure resources. If required, you then decompile the JSON file to an initial Bicep file, which you improve upon by refactoring. When you have a working file, you test and deploy using a process that minimizes the risk of breaking changes to your Azure environment.

In this article we summarize this recommended workflow. For detailed guidance, see Migrate Azure resources and JSON ARM templates to use Bicep.

Phase 1: Convert

In the *convert* phase of migrating your resources to Bicep, the goal is to capture an initial representation of your Azure resources. The Bicep file you create in this phase isn't complete, and it's not ready to be used. However, the file gives you a starting point for your migration.

The convert phase consists of two steps, which you complete in sequence:

1. **Capture a representation of your Azure resources.** If you have an existing JSON template that you're converting to Bicep, the first step is easy - you already have your source template. If you're converting Azure resources that were deployed by using the portal or another tool, you need to capture the resource definitions. You can capture a JSON representation of your resources using the Azure portal, Azure CLI, or Azure PowerShell cmdlets to *export* single resources, multiple resources, and entire resource groups. You can use the **Insert Resource** command within Visual Studio Code to import a

Bicep representation of your Azure resource.

2. **If required, convert the JSON representation to Bicep using the *decompile* command.** The Bicep tooling includes the `decompile` command to convert templates. You can invoke the `decompile` command from either the Azure CLI, or from the Bicep CLI. The decompilation process is a best-effort process and doesn't guarantee a full mapping from JSON to Bicep. You may need to revise the generated Bicep file to meet your template best practices before using the file to deploy resources.

> **NOTE**
>
> You can import a resource by opening the Visual Studio Code command palette. Use `Ctrl+Shift+P` on Windows and Linux and `⌘+Shift+P` on macOS.

Phase 2: Migrate

In the *migrate* phase of migrating your resources to Bicep, the goal is to create the first draft of your deployable Bicep file, and to ensure it defines all of the Azure resources that are in scope for the migration.

The migrate phase consists of three steps, which you complete in sequence:

1. **Create a new empty Bicep file.** It's good practice to create a brand new Bicep file. The file you created in the *convert* phase is a reference point for you to look at, but you shouldn't treat it as final or deploy it as-is.

2. **Copy each resource from your decompiled template.** Copy each resource individually from the converted Bicep file to the new Bicep file. This process helps you resolve any issues on a per-resource basis and to avoid any confusion as your template grows in size.

3. **Identify and recreate any missing resources.** Not all Azure resource types can be exported through the Azure portal, Azure CLI, or Azure PowerShell. For example, virtual machine extensions such as the DependencyAgentWindows and MMAExtension (Microsoft Monitoring Agent) aren't supported resource types for export. For any resource that wasn't exported, such as virtual machine extensions, you'll need to recreate those resources in your new Bicep file. There are several tools and approaches you can use to recreate resources, including Azure Resource Explorer, the Bicep and ARM template reference documentation, and the Azure Quickstart Templates site.

Phase 3: Refactor

In the *refactor* phase of migrating your resourced to Bicep, the goal is to improve the quality of your Bicep code. These improvements can include changes, such as adding code comments, that bring the template in line with your template standards.

The deploy phase consists of eight steps, which you complete in any order:

1. **Review resource API versions.** When exporting Azure resources, the exported template may not have the latest API version for a resource type. If there are specific properties that you need for future deployments, update the API to the appropriate version. It's good practice to review the API versions for each exported resource.

2. **Review the linter suggestions in your new Bicep file.** When creating Bicep files using the Bicep extension for Visual Studio Code, the Bicep linter runs automatically and highlights suggestions and errors in your code. Many of the suggestions and errors include an option to apply a quick fix of the issue. Review these recommendations and adjust your Bicep file.

3. **Revise parameters, variables, and symbolic names.** It's possible the names of parameters, variables, and symbolic names generated by the decompiler won't match your standard naming

convention. Review the generated names and make adjustments as necessary.

4. **Simplify expressions.** The decompile process may not always take advantage of some of Bicep's features. Review any expressions generated in the conversion and simplify them. For example, the decompiled template may include a `concat()` or `format()` function that could be simplified by using string interpolation. Review any suggestions from the linter and make adjustments as necessary.

5. **Review child and extension resources.** With Bicep, there are multiple ways to declare child resources and extension resources, including concatenating the names of your resources, using the `parent` keyword, and using nested resources. Consider reviewing these resources after decompilation and make sure the structure meets your standards. For example, ensure that you don't use string concatenation to create child resource names - you should use the `parent` property or a nested resource. Similarly, subnets can either be referenced as properties of a virtual network, or as a separate resource.

6. **Modularize.** If you're converting a template that has many resources, consider breaking the individual resource types into modules for simplicity. Bicep modules help to reduce the complexity of your deployments and increase the reusability of your Bicep code.

   > **NOTE**
   >
   > It's possible to use your JSON templates as modules in a Bicep deployment. Bicep has the ability to recognize JSON modules and reference them similarly to how you use Bicep modules.

7. **Add comments and descriptions.** Good Bicep code is *self-documenting*. Bicep allows you to add comments and `@description()` attributes to your code that help you document your infrastructure. Bicep supports both single-line comments using a `//` character sequence and multi-line comments that start with a `/*` and end with a `*/`. You can add comments to specific lines in your code and for sections of code.

8. **Follow Bicep best practices.** Make sure your Bicep file is following the standard recommendations. Review the Bicep best practices reference document for anything you might have missed.

Phase 4: Test

In the *test* phase of migrating your resources to Bicep, the goal is to verify the integrity of your migrated templates and to perform a test deployment.

The test phase consists of two steps, which you complete in sequence:

1. **Run the ARM template deployment what-if operation.** To help you verify your converted templates before deployment, you can use the Azure Resource Manager template deployment what-if operation. It compares the current state of your environment with the desired state that is defined in the template. The tool outputs the list of changes that will occur *without* applying the changes to your environment. You can use what-if with both incremental and complete mode deployments. Even if you plan to deploy your template using incremental mode, it's a good idea to run your what-if operation in complete mode.

2. **Perform a test deployment.** Before introducing your converted Bicep template to production, consider running multiple test deployments. If you have multiple environments (for example, development, test, and production), you may want to try deploying your template to one of your non-production environments first. After the deployment, compare the original resources with the new resource deployments for consistency.

Phase 5: Deploy

In the *deploy* phase of migrating your resources to Bicep, the goal is to deploy your final Bicep file to production.

The deploy phase consists of four steps, which you complete in sequence:

1. **Prepare a rollback plan.** The ability to recover from a failed deployment is crucial. Develop a rollback plan in the event of any breaking changes introduced into your environments. Take inventory of the types of resources that are deployed, such as virtual machines, web apps, and databases. Each resource's data plane should be considered as well. Do you have a way to recover a virtual machine and its data? Do you have a way to recover a database after deletion? A well-developed rollback plan will help to keep your downtime to a minimum if any issues arise from a deployment.

2. **Run the what-if operation against production.** Before deploying your final Bicep file to production, run the what-if operation against your production environment, making sure to use production parameter values, and consider documenting the results.

3. **Deploy manually.** If you're going to use the converted template in a pipeline, such as Azure DevOps or GitHub Actions, consider running the deployment from your local machine first. It is better to verify the functionality of the template before adding it to your production pipeline. That way, you can respond quickly if there's a problem.

4. **Run smoke tests.** After your deployment completes, it is a good idea to run a series of *smoke tests* - simple checks that validate that your application or workload is functioning properly. For example, test to see if your web app is accessible through normal access channels, such as the public Internet or across a corporate VPN. For databases, attempt to make a database connection and execute a series of queries. With virtual machines, log in to the virtual machine and make sure that all services are up and running.

Next steps

To learn more about the Bicep decompiler, see Decompiling ARM template JSON to Bicep.

Decompiling ARM template JSON to Bicep

9/21/2022 • 3 minutes to read • Edit Online

This article describes how to decompile Azure Resource Manager templates (ARM templates) to Bicep files. You must have the Bicep CLI installed to run the conversion commands.

> **NOTE**
> From Visual Studio Code, you can directly create resource declarations by importing from existing resources. For more information, see Bicep commands.

Decompiling an ARM template helps you get started with Bicep development. If you have a library of ARM templates and want to use Bicep for future development, you can decompile them to Bicep. However, the Bicep file might need revisions to implement best practices for Bicep.

This article shows how to run the `decompile` command in Azure CLI. If you're not using Azure CLI, run the command without `az` at the start of the command. For example, `az bicep decompile` becomes `bicep decompile`.

Decompile from JSON to Bicep

To decompile ARM template JSON to Bicep, use:

```
az bicep decompile --file main.json
```

The command creates a file named *main.bicep* in the same directory as *main.json*. If *main.bicep* exists in the same directory, use the **--force** switch to overwrite the existing Bicep file.

Caution

Decompilation attempts to convert the file, but there is no guaranteed mapping from ARM template JSON to Bicep. You may need to fix warnings and errors in the generated Bicep file. Or, decompilation can fail if an accurate conversion isn't possible. To report any issues or inaccurate conversions, create an issue.

The decompile and build commands produce templates that are functionally equivalent. However, they might not be exactly the same in implementation. Converting a template from JSON to Bicep and then back to JSON probably results in a template with different syntax than the original template. When deployed, the converted templates produce the same results.

Fix conversion issues

Suppose you have the following ARM template:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "storageAccountType": {
      "type": "string",
      "defaultValue": "Standard_LRS",
      "allowedValues": [
        "Standard_LRS",
        "Standard_GRS",
        "Standard_ZRS",
        "Premium_LRS"
      ],
      "metadata": {
        "description": "Storage Account type"
      }
    },
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]",
      "metadata": {
        "description": "Location for all resources."
      }
    }
  },
  "variables": {
    "storageAccountName": "[concat('store', uniquestring(resourceGroup().id))]"
  },
  "resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "apiVersion": "2019-06-01",
      "name": "[variables('storageAccountName')]",
      "location": "[parameters('location')]",
      "sku": {
        "name": "[parameters('storageAccountType')]"
      },
      "kind": "StorageV2",
      "properties": {}
    }
  ],
  "outputs": {
    "storageAccountName": {
      "type": "string",
      "value": "[variables('storageAccountName')]"
    }
  }
}
```

When you decompile it, you get:

```
@allowed([
  'Standard_LRS'
  'Standard_GRS'
  'Standard_ZRS'
  'Premium_LRS'
])
@description('Storage Account type')
param storageAccountType string = 'Standard_LRS'

@description('Location for all resources.')
param location string = resourceGroup().location

var storageAccountName_var = 'store${uniqueString(resourceGroup().id)}'

resource storageAccountName 'Microsoft.Storage/storageAccounts@2019-06-01' = {
  name: storageAccountName_var
  location: location
  sku: {
    name: storageAccountType
  }
  kind: 'StorageV2'
  properties: {}
}

output storageAccountName string = storageAccountName_var
```

The decompiled file works, but it has some names that you might want to change. The variable `var storageAccountName_var` has an unusual naming convention. Let's change it to:

```
var uniqueStorageName = 'store${uniqueString(resourceGroup().id)}'
```

The resource has a symbolic name that you might want to change. Instead of `storageAccountName` for the symbolic name, use `exampleStorage`.

```
resource exampleStorage 'Microsoft.Storage/storageAccounts@2019-06-01' = {
```

Since you changed the name of the variable for the storage account name, you need to change where it's used.

```
resource exampleStorage 'Microsoft.Storage/storageAccounts@2019-06-01' = {
  name: uniqueStorageName
```

And in the output, use:

```
output storageAccountName string = uniqueStorageName
```

The complete file is:

```
@allowed([
  'Standard_LRS'
  'Standard_GRS'
  'Standard_ZRS'
  'Premium_LRS'
])
@description('Storage Account type')
param storageAccountType string = 'Standard_LRS'

@description('Location for all resources.')
param location string = resourceGroup().location

var uniqueStorageName = 'store${uniqueString(resourceGroup().id)}'

resource exampleStorage 'Microsoft.Storage/storageAccounts@2019-06-01' = {
  name: uniqueStorageName
  location: location
  sku: {
    name: storageAccountType
  }
  kind: 'StorageV2'
  properties: {}
}

output storageAccountName string = uniqueStorageName
```

Export template and convert

You can export the template for a resource group, and then pass it directly to the `decompile` command. The following example shows how to decompile an exported template.

- Azure CLI
- PowerShell
- Portal

```
az group export --name "your_resource_group_name" > main.json
az bicep decompile --file main.json
```

Side-by-side view

The Bicep Playground enables you to view equivalent ARM template and Bicep files side by side. You can select **Sample Template** to see both versions. Or, select **Decompile** to upload your own ARM template and view the equivalent Bicep file.

Next steps

To learn about all of the Bicep CLI commands, see Bicep CLI commands.

Contribute to Bicep

9/21/2022 • 2 minutes to read • Edit Online

Bicep is an open-source project. That means you can contribute to Bicep's development, and participate in the broader Bicep community.

Contribution types

- **Azure Quickstart Templates.** You can contribute example Bicep files and ARM templates to the Azure Quickstart Templates repository. For more information, see the Azure Quickstart Templates contribution guide.
- **Documentation.** Bicep's documentation is open to contributions, too. For more information, see our contributor guide overview.
- **Snippets.** Do you have a favorite snippet you think the community would benefit from? You can add it to the Visual Studio Code extension's collection of snippets. For more information, see Contributing to Bicep.
- **Code changes.** If you're a developer and you have ideas you'd like to see in the Bicep language or tooling, you can contribute a pull request. For more information, see Contributing to Bicep.

Next steps

To learn about the structure and syntax of Bicep, see Bicep file structure.

Bicep CLI commands

9/21/2022 • 5 minutes to read • Edit Online

This article describes the commands you can use in the Bicep CLI. You must have the Bicep CLI installed to run the commands.

You can either run the Bicep CLI commands through Azure CLI or by calling Bicep directly. This article shows how to run the commands in Azure CLI. When running through Azure CLI, you start the commands with `az`. If you're not using Azure CLI, run the commands without `az` at the start of the command. For example, `az bicep build` becomes `bicep build`.

build

The `build` command converts a Bicep file to an Azure Resource Manager template (ARM template). Typically, you don't need to run this command because it runs automatically when you deploy a Bicep file. Run it manually when you want to see the ARM template JSON that is created from your Bicep file.

The following example converts a Bicep file named *main.bicep* to an ARM template named *main.json*. The new file is created in the same directory as the Bicep file.

```
az bicep build --file main.bicep
```

The next example saves *main.json* to a different directory.

```
az bicep build --file main.bicep --outdir c:\jsontemplates
```

The next example specifies the name and location of the file to create.

```
az bicep build --file main.bicep --outfile c:\jsontemplates\azuredeploy.json
```

To print the file to `stdout`, use:

```
az bicep build --file main.bicep --stdout
```

If your Bicep file includes a module that references an external registry, the build command automatically calls restore. The restore command gets the file from the registry and stores it in the local cache.

> **NOTE**
> The restore command doesn't refresh the cache. For more information, see restore.

To not call restore automatically, use the `--no-restore` switch:

```
az bicep build --no-restore <bicep-file>
```

The build process with the `--no-restore` switch fails if one of the external modules isn't already cached:

```
The module with reference "br:exampleregistry.azurecr.io/bicep/modules/storage:v1" has not been restored.
```

When you get this error, either run the `build` command without the `--no-restore` switch or run `bicep restore` first.

To use the `--no-restore` switch, you must have Bicep CLI version **0.4.1008 or later**.

decompile

The `decompile` command converts ARM template JSON to a Bicep file.

```
az bicep decompile --file main.json
```

The command creates a file named *main.bicep* in the same directory as *main.json*. If *main.bicep* exists in the same directory, use the **--force** switch to overwrite the existing Bicep file.

For more information about using this command, see Decompiling ARM template JSON to Bicep.

generate-params

The `generate-params` command builds *.parameters.json* file from the given bicep file, updates if there is an existing parameters.json file.

```
az bicep generate-params --file main.bicep
```

The command creates a parameter file named *main.parameters.json*. The parameter file only contains the parameters without default values configured in the Bicep file.

install

The `install` command adds the Bicep CLI to your local environment. For more information, see Install Bicep tools. This command is only available through Azure CLI.

To install the latest version, use:

```
az bicep install
```

To install a specific version:

```
az bicep install --version v0.3.255
```

list-versions

The `list-versions` command returns all available versions of the Bicep CLI. Use this command to see if you want to upgrade or install a new version. This command is only available through Azure CLI.

```
az bicep list-versions
```

The command returns an array of available versions.

```
[
  "v0.4.1",
  "v0.3.539",
  "v0.3.255",
  "v0.3.126",
  "v0.3.1",
  "v0.2.328",
  "v0.2.317",
  "v0.2.212",
  "v0.2.59",
  "v0.2.14",
  "v0.2.3",
  "v0.1.226-alpha",
  "v0.1.223-alpha",
  "v0.1.37-alpha",
  "v0.1.1-alpha"
]
```

publish

The `publish` command adds a module to a registry. The Azure container registry must exist and the account publishing to the registry must have the correct permissions. For more information about setting up a module registry, see Use private registry for Bicep modules.

After publishing the file to the registry, you can reference it in a module.

To use the publish command, you must have Bicep CLI version **0.4.1008 or later**.

To publish a module to a registry, use:

```
az bicep publish --file <bicep-file> --target br:<registry-name>.azurecr.io/<module-path>:<tag>
```

For example:

```
az bicep publish --file storage.bicep --target br:exampleregistry.azurecr.io/bicep/modules/storage:v1
```

The `publish` command doesn't recognize aliases that you've defined in a bicepconfig.json file. Provide the full module path.

> **WARNING**
> Publishing to the same target overwrites the old module. We recommend that you increment the version when updating.

restore

When your Bicep file uses modules that are published to a registry, the `restore` command gets copies of all the required modules from the registry. It stores those copies in a local cache. A Bicep file can only be built when the external files are available in the local cache. Typically, you don't need to run `restore` because it's called automatically by `build`.

To restore external modules to the local cache, the account must have the correct permissions to access the registry. You can configure the credential precedence for authenticating to the registry in the Bicep config file.

To use the restore command, you must have Bicep CLI version **0.4.1008 or later**. This command is currently only available when calling the Bicep CLI directly. It's not currently available through the Azure CLI command.

To manually restore the external modules for a file, use:

```
bicep restore <bicep-file> [--force]
```

The Bicep file you provide is the file you wish to deploy. It must contain a module that links to a registry. For example, you can restore the following file:

```
module stgModule 'br:exampleregistry.azurecr.io/bicep/modules/storage:v1' = {
  name: 'storageDeploy'
  params: {
    storagePrefix: 'examplestg1'
  }
}
```

The local cache is found in:

- On Windows

  ```
  %USERPROFILE%\.bicep\br\<registry-name>.azurecr.io\<module-path\<tag>
  ```

- On Linux

  ```
  /home/<username>/.bicep
  ```

The `restore` command doesn't refresh the cache if a module is already cached. To fresh the cache, you can either delete the module path from the cache or use the `--force` switch with the `restore` command.

upgrade

The `upgrade` command updates your installed version with the latest version. This command is only available through Azure CLI.

```
az bicep upgrade
```

version

The `version` command returns your installed version.

```
az bicep version
```

The command shows the version number.

```
Bicep CLI version 0.4.1008 (223b8d227a)
```

To call this command directly through the Bicep CLI, use:

```
bicep --version
```

If you haven't installed Bicep CLI, you see an error indicating Bicep CLI wasn't found.

Next steps

To learn about deploying a Bicep file, see:

- Azure CLI
- Cloud Shell
- PowerShell