

Domain 1: Prerequisites for Azure administrators

Introduction to Azure Cloud Shell

1. Introduction

<https://learn.microsoft.com/en-us/training/modules/intro-to-azure-cloud-shell/1-introduction>

Introduction

Completed

Azure Cloud Shell is a browser-accessible command-line experience for managing Azure resources. It provides the flexibility of choosing the shell experience that best suits the way you work, either Bash or PowerShell. Traditionally, to interact with Azure resources via command-line, you need to install the necessary components into your local computer (PC, Mac, Linux). With Cloud Shell, you have an authenticated, interactive shell that isn't part of a local machine.

You're an IT admin for Contoso Corporation, responsible for the cloud infrastructure on which the company hosts its applications. These applications use different cloud resources, such as Azure VMs, Azure blob storage, Azure networking, and others. On many occasions, you're asked to perform operations on these cloud resources at moments when you're not using your work laptop. In some cases, you have a friend's laptop, a smartphone, or another personal PC.

This module explains what Azure Cloud Shell does, how it works, and when you should choose to use Cloud Shell as a solution to meet your organization's needs.

Learning objectives

In this module, you'll:

- Describe Azure Cloud Shell and the functionality it provides.
- Determine whether Cloud Shell meets the needs of your organization.
- Recognize how to use Cloud Shell and persist files for multiple sessions.

2. What is Azure Cloud Shell?

What is Azure Cloud Shell?

Completed

Azure Cloud Shell is a command-line environment you can access through your web browser. You can use this environment to manage Azure resources, including VMs, storage, and networking. Just like you do when using the Azure CLI or Azure PowerShell.

Because Microsoft manages Cloud Shell, you always have access to the most recent versions of the Azure CLI and PowerShell modules right from any browser. You don't have to worry about keeping modules up to date. With Cloud Shell, you just open your browser and sign in. Just like that, you have access to a command-line environment fully connected with your account's permissions and the resources to which you have access. All that works in an infrastructure that's compliant with double encryption at rest by default. You don't need to take any further action!

Azure Cloud Shell also provides cloud storage to persist files such as SSH keys, scripts, and more. This functionality lets you access important files in between sessions and with different machines. Finally, you can use the Cloud Shell editor to make changes to files, such as scripts, that are saved into this cloud storage directly from the Cloud Shell interface.

3. How does Azure Cloud Shell work?

<https://learn.microsoft.com/en-us/training/modules/intro-to-azure-cloud-shell/3-how-azure-cloud-shell-works>

How does Azure Cloud Shell work?

Completed

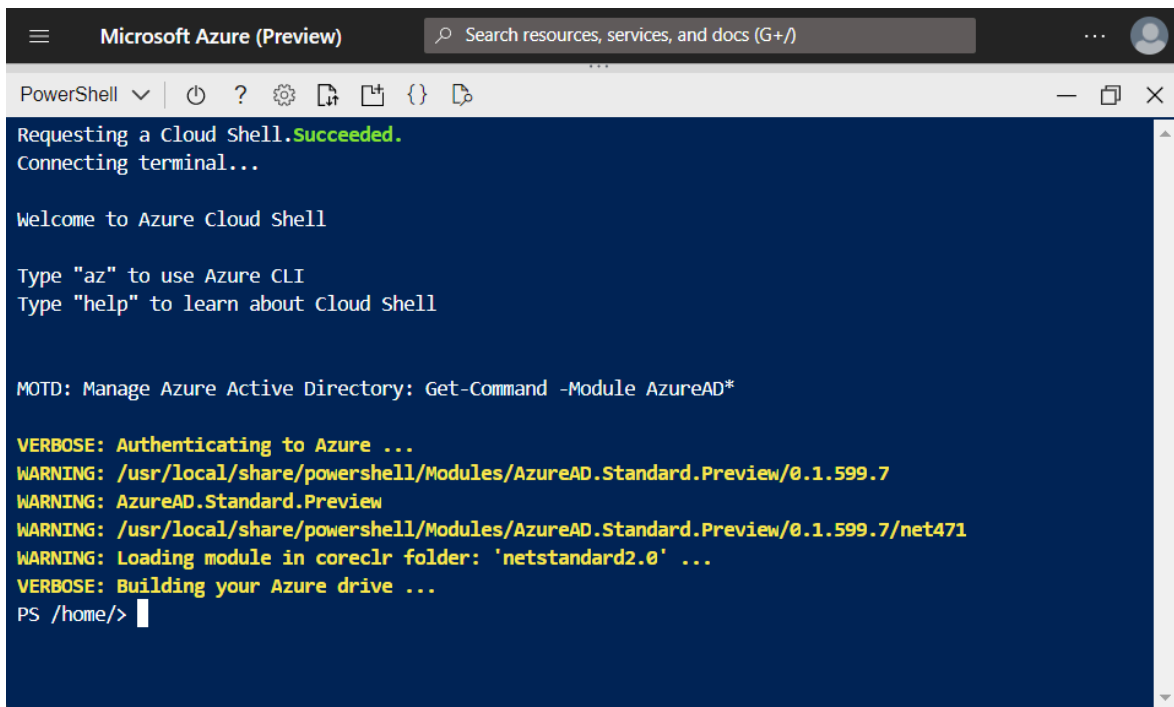
As an IT admin for Contoso Corporation, you're frequently on-call to perform administrative tasks and resolve workload disruptions to resources in your organization's Azure subscriptions. When visiting a family member during a weekend that you're on call, the development team notifies you of a problem with an Azure virtual machine (VM). The VM became nonresponsive during scheduled maintenance for the upgrade of an application that runs on it. Because the developers weren't granted access to the underlying Azure virtual machine hosting infrastructure, they're only able to remotely access the VM when it's operating normally. So, you're being called to diagnose and remediate the problem.

Since you're visiting family, you don't have access to your administrative workstation and diagnostic scripts. You do have access to a laptop with an internet browser. Using the laptop, you browse to the Azure portal, authenticate against your organization's Azure subscription, open Azure Cloud Shell, mount an Azure File Share, access your diagnostic scripts, and diagnose and remediate the problems with the VM, returning it to operation.

Access Cloud Shell

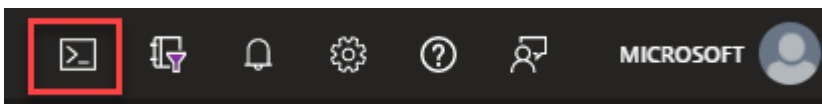
You have a few different options for accessing Azure Cloud Shell:

- From a direct link: <https://shell.azure.com>

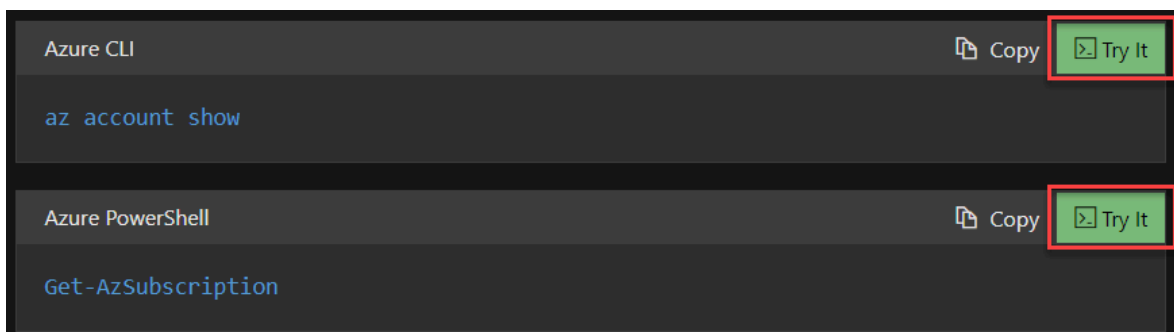


The screenshot shows a web browser window titled "Microsoft Azure (Preview)". The address bar contains a search bar with the text "Search resources, services, and docs (G+/)". Below the browser window, the Azure Cloud Shell interface is displayed. It features a dark blue background with white text. The text reads: "Requesting a Cloud Shell.Succeeded. Connecting terminal... Welcome to Azure Cloud Shell Type 'az' to use Azure CLI Type 'help' to learn about Cloud Shell MOTD: Manage Azure Active Directory: Get-Command -Module AzureAD* VERBOSE: Authenticating to Azure ... WARNING: /usr/local/share/powershell/Modules/AzureAD.Standard.Preview/0.1.599.7 WARNING: AzureAD.Standard.Preview WARNING: /usr/local/share/powershell/Modules/AzureAD.Standard.Preview/0.1.599.7/net471 WARNING: Loading module in coreclr folder: 'netstandard2.0' ... VERBOSE: Building your Azure drive ... PS /home/>".

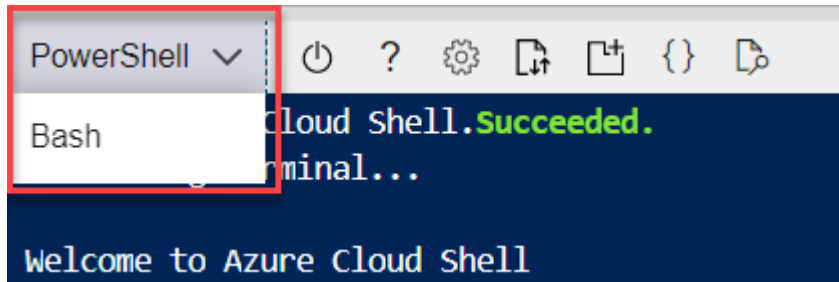
- From the Azure portal



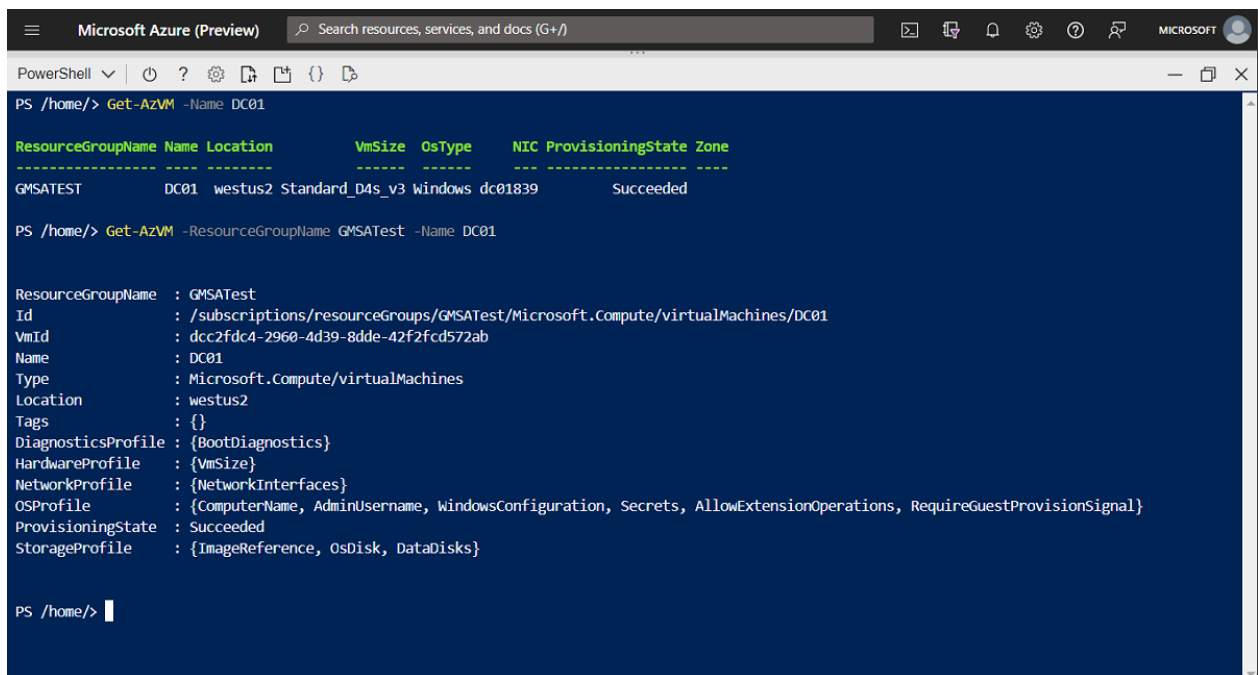
- From code snippets when accessing Microsoft Learn:



When you open a Cloud Shell session, a temporary host is allocated to your session. This VM is preconfigured with the latest versions of PowerShell and Bash. You can then select the command-line experience you want to use:



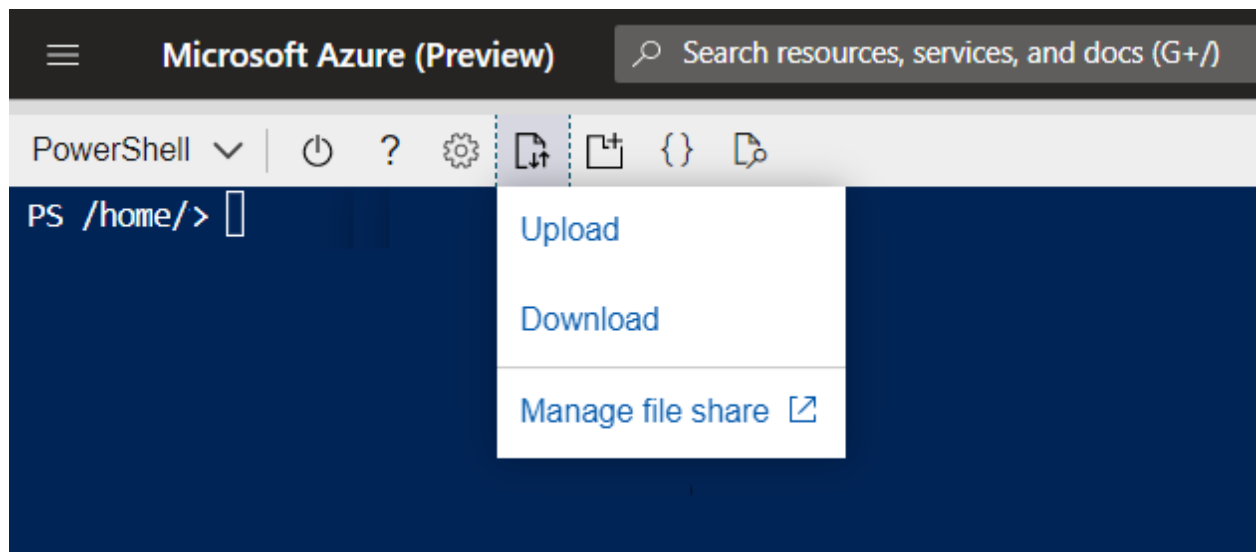
After you select the shell experience you want to use, you can start managing your Azure resources:



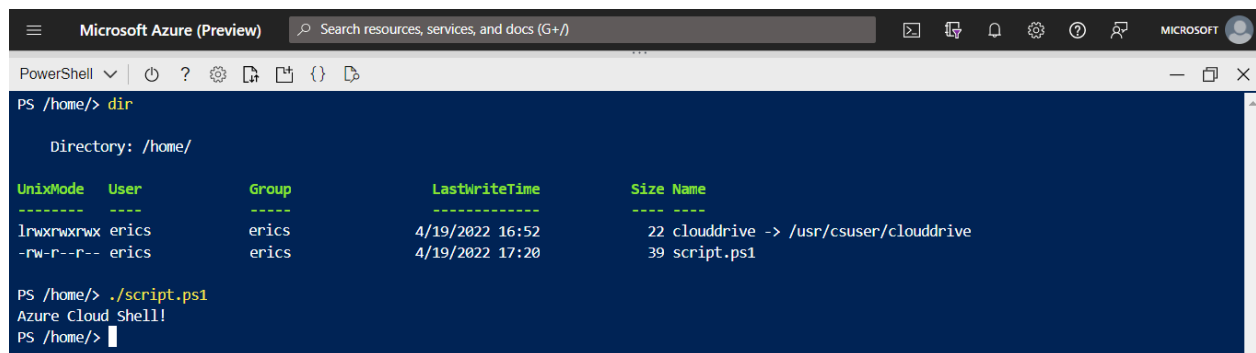
Cloud Shell sessions terminate after 20 minutes of inactivity. When a session terminates, files on your CloudDrive are persisted, but you need to start a new session to access the Cloud Shell environment.

Access your own scripts and files

When using Cloud Shell, you might also need to run scripts or use files for different actions. You can persist files on Cloud Shell by using the Azure CloudDrive:



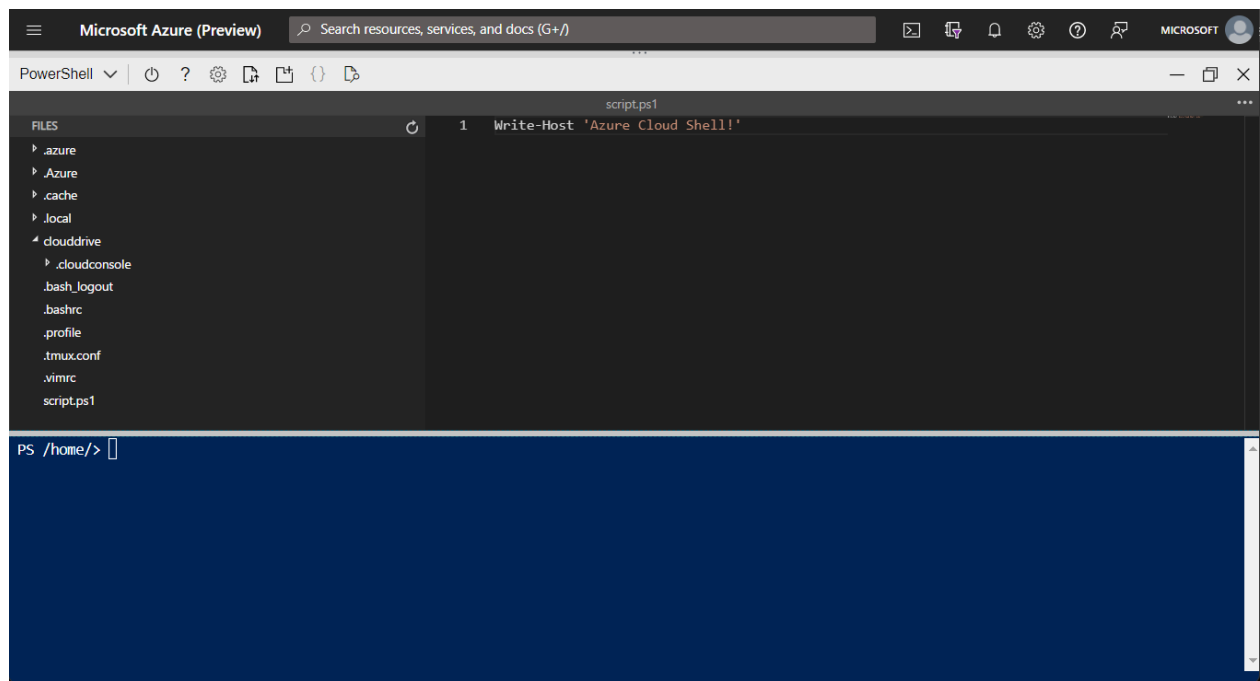
After uploading files, you can interact with them as you would in a regular PowerShell or Bash session:



Now that your file resides on CloudDrive, you can close the session and open another session on a different device and still access the same file. Cloud Shell also lets you map an Azure Storage File Share, which is tied to a specific region. Access to an Azure File Share lets you work with the contents of that share through Cloud Shell.

If you need to edit scripts hosted on the CloudDrive or File Share, you can use the Cloud Shell editor. Select the curly brackets {} icon on the browser and open the file you want to edit, or use the command `code` and specify the filename; for example:

```
code temp.txt
```



Note

The `code` command only works in the Classic mode in Cloud Shell. To enable Classic mode, select the **More** icon (...), then select **Settings > Go to Classic version**.

Cloud Shell tools

If you need to manage resources (such as Docker containers or Kubernetes Clusters) or want to use non-Microsoft tools (such as Ansible and Terraform) in Cloud Shell, the Cloud Shell session comes with these add-ons already preconfigured.

Here's a list of all add-ons available to you within a Cloud Shell session:

Category	Name
Linux tools	bash zsh sh tmux dig
Azure tools	Azure CLI AzCopy Azure Functions CLI Service Fabric CLI Batch Shipyard blobxfer

Category	Name
Text editors	code (Cloud Shell editor) vim nano emacs
Source control	git
Build tools	make maven npm pip
Containers	Docker Machine Kubectl Helm DC/OS CLI
Databases	MySQL client PostgreSQL client sqlcmd Utility mssql-scripter
Other	iPython Client Cloud Foundry CLI Terraform Ansible Chef InSpec Puppet Bolt HashiCorp Packer Office 365 CLI

4. When should you use Azure Cloud Shell?

<https://learn.microsoft.com/en-us/training/modules/intro-to-azure-cloud-shell/4-when-to-use-azure-cloud-shell>

When should you use Azure Cloud Shell?

Completed

As an IT Admin for Contoso Corporation, you need alternatives to interact with Azure resources from the command line even when not using your default administrative device.

You can use Azure Cloud Shell to:

- Open a secure command-line session from any browser-based device.
- Interact with Azure resources without the need to install plug-ins or add-ons to your device.
- Persist files between sessions for later use.
- Use either Bash or PowerShell, whichever you prefer, to manage Azure resources.
- Edit files (such as scripts) via the Cloud Shell editor.

You shouldn't use Azure Cloud Shell if:

- You intend to leave a session open for more than 20 minutes for long running scripts or activities. In these cases, your session is disconnected without warning, and the current state is lost.
- You need admin permissions, such as sudo access, from within the Azure CLI or PowerShell environment.
- You need to install tools that aren't supported in the limited Cloud Shell environment, but instead require an environment such as a custom virtual machine or container.
- You need storage from different regions. You might need to back up and synchronize this content since only one region can have the storage allocated to Azure Cloud Shell.
- You need to open multiple sessions at the same time. Azure Cloud Shell allows only one instance at time and isn't suitable for concurrent work across multiple subscriptions or tenants.

5. Module assessment

<https://learn.microsoft.com/en-us/training/modules/intro-to-azure-cloud-shell/5-knowledge-check>

Module assessment

Completed

6. Summary

<https://learn.microsoft.com/en-us/training/modules/intro-to-azure-cloud-shell/6-summary>

Summary

Completed

Azure Cloud Shell is a browser-accessible command-line experience for managing Azure resources. Rather than having to configure an Azure CLI or PowerShell session from your workstation, you can access Cloud Shell on any standard, compliant browser.

Cloud Shell provides the flexibility of choosing the shell experience that best suits the way you work, allowing you to work either in Bash or PowerShell, right from the browser. Cloud Shell also provides you with the mechanisms to persist files between sessions, and provides access to a minimalist version of the Visual Studio Code editor for more complex operations.

Learn more

Check out these articles to learn more about Azure Cloud Shell.

[Azure Cloud Shell Overview](#)

[Azure Cloud Shell – Browser-Based Command Line](#)

Deploy Azure infrastructure by using JSON ARM templates

1. Introduction

<https://learn.microsoft.com/en-us/training/modules/create-azure-resource-manager-template-vs-code/1-introduction>

Introduction

Completed

JSON Azure Resource Manager templates (ARM templates) allow you to specify your project's infrastructure in a declarative and reusable way. You can version and save the templates in the same source control as your development project.

Suppose you're managing a software team that's developing an inventory system for your partner companies. You plan to deploy this product to Azure, and let each partner company have its own solution. You plan to implement different policies for each deployment through different Azure storage accounts. You decide to use the practice of *infrastructure as code* by using ARM templates. This approach lets you track the different versions and ensure that your infrastructure deployments for each environment are consistent and flexible.

In this module, we introduce you to ARM template structure and let you practice creating and deploying an ARM template to Azure.

Note

Bicep is a language for defining your Azure resources. It has a simpler authoring experience than JSON, along with other features that help improve the quality of your infrastructure as code. We recommend that anyone new to infrastructure as code on Azure use Bicep instead of JSON. To learn about Bicep, see the [Fundamentals of Bicep](#) learning path.

Learning objectives

In this module, you will:

- Implement a JSON ARM template by using Visual Studio Code.
- Declare resources and add flexibility to your template by adding parameters and outputs.

Prerequisites

- Familiarity with Azure, including the Azure portal, subscriptions, resource groups, and resource definitions.
- An Azure account. You can get a free account [here](#).
- [Visual Studio Code](#) installed locally.
- Either:
 - The latest [Azure CLI](#) tools installed locally.
 - The latest [Azure PowerShell](#) installed locally.

2. Explore Azure Resource Manager template structure

<https://learn.microsoft.com/en-us/training/modules/create-azure-resource-manager-template-vs-code/2-explore-template-structure>

Explore Azure Resource Manager template structure

Completed

In this unit, you learn about using Azure Resource Manager templates (ARM templates) to implement infrastructure as code. You survey the sections of an ARM template, learn how to deploy your ARM template to Azure, and delve into detail on the *resources* section of the ARM template.

What is infrastructure as code?

Infrastructure as code allows you to describe, through code, the infrastructure that you need for your application.

With infrastructure as code, you can maintain both your application code and everything you need to deploy your application in a central code repository. The advantages to infrastructure as code are:

- Consistent configurations
- Improved scalability
- Faster deployments
- Better traceability

This video explains infrastructure as code:

What is an ARM template?

ARM templates are JavaScript Object Notation (JSON) files that define the infrastructure and configuration for your deployment. The template uses a *declarative syntax*. The declarative syntax is a way of building the structure and elements that outline what resources look like without describing the control flow. Declarative syntax is different than *imperative syntax*, which uses commands for the computer to perform. Imperative scripting focuses on specifying each step in deploying the resources.

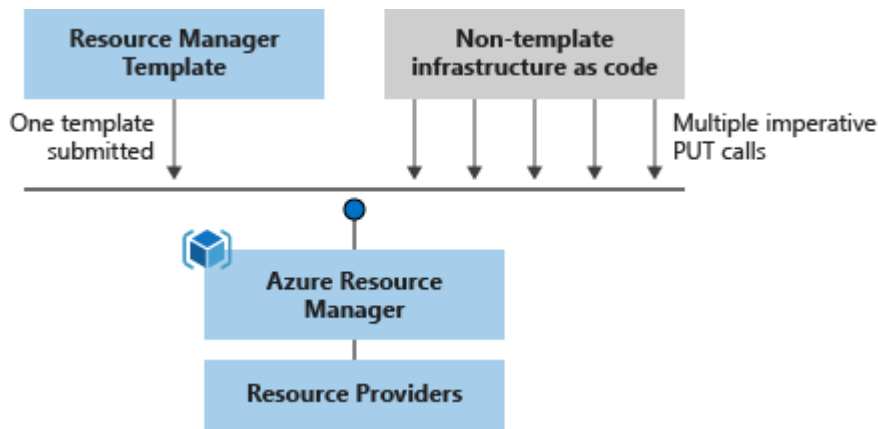
ARM templates allow you to declare what you intend to deploy without having to write the sequence of programming commands to create it. In an ARM template, you specify the resources and the properties for those resources. [Azure Resource Manager](#) then uses that information to deploy the resources in an organized and consistent manner.

Benefits of using ARM templates

ARM templates allow you to automate deployments and use the practice of infrastructure as code (IaC). The template code becomes part of your infrastructure and development projects. Just like application code, you can store the IaC files in a source repository and version it.

ARM templates are *idempotent*, which means you can deploy the same template many times and get the same resource types in the same state.

Resource Manager orchestrates deploying the resources so they're created in the correct order. When possible, resources are created in parallel, so ARM template deployments finish faster than scripted deployments.



Resource Manager also has built-in validation. It checks the template before starting the deployment to make sure the deployment succeeds.

If your deployments become more complex, you can break your ARM templates into smaller, reusable components. You can link these smaller templates together at deployment time. You can also nest templates inside other templates.

In the Azure portal, you can review your deployment history and get information about the state of the deployment. The portal displays values for all parameters and outputs.

You can also integrate your ARM templates into continuous integration and continuous deployment (CI/CD) tools like [Azure Pipelines](#), which can automate your release pipelines for fast and reliable application and infrastructure updates. By using Azure DevOps and ARM template tasks, you can continuously build and deploy your projects.

ARM template file structure

When you're writing an ARM template, you need to understand all the parts that make up the template and what they do. ARM template files are made up of the following elements:

Element	Description
schema	A required section that defines the location of the JSON schema file that describes the structure of JSON data. The version number you use depends on the scope of the deployment and your JSON editor.
contentVersion	A required section that defines the version of your template (such as 1.0.0.0). You can use this value to document significant changes in your template to ensure you're deploying the right template.
apiProfile	An optional section that defines a collection of API versions for resource types. You can use this value to avoid having to specify API versions for each resource in the template.
parameters	An optional section where you define values that are provided during deployment. You can provide these values in a parameter file, by command-line parameters, or in the Azure portal.
variables	An optional section where you define values that are used to simplify template language expressions.

Element	Description
functions	An optional section where you can define user-defined functions that are available within the template. User-defined functions can simplify your template when complicated expressions are used repeatedly in your template.
resources	A required section that defines the actual items you want to deploy or update in a resource group or a subscription.
output	An optional section where you specify the values that are returned at the end of the deployment.

Deploy an ARM template to Azure

You can deploy an ARM template to Azure in one of the following ways:

- Deploy a local template
- Deploy a linked template
- Deploy in a continuous deployment pipeline

This module focuses on deploying a local ARM template. In future Learn modules, you learn how to deploy more complicated infrastructure and how to integrate with Azure Pipelines.

To deploy a local template, you need to have either [Azure PowerShell](#) or the [Azure CLI](#) installed locally.

First, sign in to Azure by using the Azure CLI or Azure PowerShell.

- [Azure CLI](#)
- [PowerShell](#)

```
az login
```

Next, define your resource group. You can use an already-defined resource group or create a new one with the following command. You can obtain available location values from: `az account list-locations` (CLI) or `Get-AzLocation` (PowerShell). You can configure the default location using `az configure --defaults location=<location>`.

- [Azure CLI](#)
- [PowerShell](#)

```
az group create \
  --name {name of your resource group} \
  --location "{location}"
```

To start a template deployment at the resource group, use either the Azure CLI command [az deployment group create](#) or the Azure PowerShell command [New-AzResourceGroupDeployment](#).

Tip

The difference between `az deployment group create` and `az group deployment create` is that `az group deployment create` is an old command to be deprecated and will be replaced by `az deployment group create`. Therefore, we recommend using `az deployment group create` to deploy resources under the resource group scope.

Both commands require the resource group, the region, and the name for the deployment so you can easily identify it in the deployment history. For convenience, the exercises create a variable that stores the path to the template file. This variable makes it easier for you to run deployment commands, because you don't have to retype the path every time you deploy. Here's an example:

- [Azure CLI](#)
- [PowerShell](#)

To run this deployment command, you must have the [latest version](#) of Azure CLI.

```
templateFile="{provide-the-path-to-the-template-file}"
az deployment group create \
  --name blanktemplate \
  --resource-group myResourceGroup \
  --template-file $templateFile
```

Use linked templates to deploy complex solutions. You can break a template into many templates and deploy these templates through a main template. When you deploy the main template, it triggers the linked template's deployment. You can store and secure the linked template by using a SAS token.

A CI/CD pipeline automates the creation and deployment of development projects, which includes ARM template projects. The two most common pipelines used for template deployment are Azure Pipelines or [GitHub Actions](#).

More information on these two types of deployment is covered in other modules.

Add resources to the template

To add a resource to your template, you need to know the resource provider and its types of resources. The syntax for this combination is in the form of `{resource-provider}/{resource-type}`. For example, to add a storage account resource to your template, you need the `Microsoft.Storage` resource provider. One of the types for this provider is `storageAccount`. So your resource type is displayed as `Microsoft.Storage/storageAccounts`. You can use a list of [resource providers for Azure services](#) to find the providers you need.

After you define the provider and resource type, you need to understand the properties for each resource type you want to use. For details, see [Define resources in Azure Resource Manager](#)

[templates](#). To find the resource, view the list in the left column. Notice that the properties are sorted by API version.

Filter by title

- > SQL
- > SQL Virtual Machine
- > Standby Pool
- > Storage
 - > locations/
 - storageAccounts
 - > storageAccounts/
 - ▼ (Api versions)
 - > 2025-01-01
 - > 2024-01-01
 - > 2023-05-01
 - > 2023-04-01
 - > 2023-01-01
 - > 2022-09-01
 - > 2022-05-01
 - > 2021-09-01
 - > 2021-08-01
 - > 2021-06-01

Learn / Azure /

Ask Learn Focus mode

Microsoft.Storage storageAccounts

07/07/2025

Choose a deployment language

Bicep ARM template Terraform

API Versions: Latest

ARM template resource definition

The storageAccounts resource type can be deployed with operations that target:

For a list of changed properties in each API version, see [change log](#).

Resource format

To create a Microsoft.Storage/storageAccounts resource, add the following JSON to your template.

Here's an example of some of the listed properties from the Storage Accounts page:

Microsoft.Storage/storageAccounts

[Expand table](#)

Name	Description	Value
apiVersion	The api version	'2025-01-01'
extendedLocation	Optional. Set the extended location of the resource. If not set, the storage account will be created in Azure main region. Otherwise it will be created in the specified extended location	ExtendedLocation
identity	The identity of the resource.	Identity
kind	Required. Indicates the type of storage account.	'BlobStorage' 'BlockBlobStorage' 'FileStorage' 'Storage' 'StorageV2' (required)
location	Required. Gets or sets the location of the resource. This will be one of the supported and registered Azure Geo Regions (e.g. West US, East US, Southeast Asia, etc.). The geo region of a resource cannot be changed once it is created, but if an identical geo region is specified on update, the request will succeed.	string (required)
name	The resource name	string Constraints: Min length = 3 Max length = 24 Pattern = <code>^[a-z0-9]+\$</code> (required)
placement	Optional. Gets or sets the zonal placement details for the storage account.	Placement

For our storage example, your template might look like this:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json",
  "contentVersion": "1.0.0.1",
  "apiProfile": "",
  "parameters": {},
  "variables": {},
  "functions": [],
  "resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "apiVersion": "2025-01-01",
      "name": "learntemplatestorage123",
      "location": "westus",
      "sku": {
```

```

        "name": "Standard_LRS"
      },
      "kind": "StorageV2",
      "properties": {
        "supportsHttpsTrafficOnly": true
      }
    ],
    "outputs": {}
  }

```

3. Exercise - Create and deploy an Azure Resource Manager template

<https://learn.microsoft.com/en-us/training/modules/create-azure-resource-manager-template-vs-code/3-exercise-create-and-deploy-template>

Exercise - Create and deploy an Azure Resource Manager template

Completed

In this exercise, you create an Azure Resource Manager (ARM) template, deploy it to Azure, and then update that ARM template to add parameters and outputs.

Create an ARM template

1. Open Visual Studio Code and create a new file called *azuredeploy.json*.
2. Copy and paste the following code into the file.

```

{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deployme
  "contentVersion": "1.0.0.0",
  "parameters": {},
  "functions": [],
  "variables": {},
  "resources": [],
  "outputs": {}
}

```

Notice that this file has all of the sections of an ARM template that we described in the previous unit.

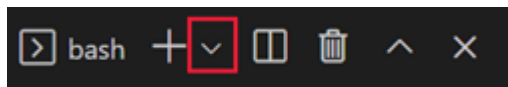
3. Save the changes to the file by pressing **Ctrl+S**.

Deploy the ARM template to Azure

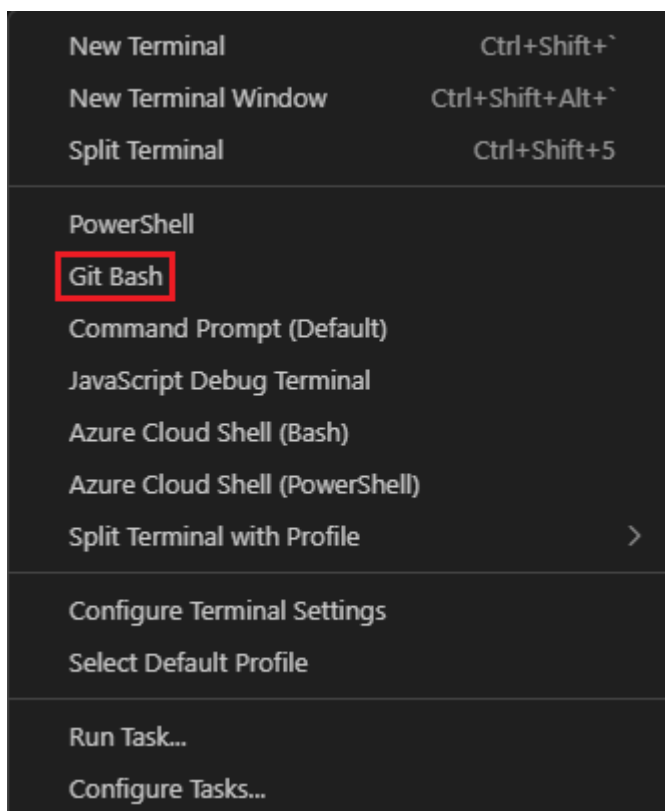
To deploy this template to Azure, you need to sign in to your Azure account from the Visual Studio Code terminal. Be sure you have the [Azure CLI](#) tools installed.

1. Select **Terminal > New Terminal** to open a terminal window.
2. If the command bar of the terminal window says **bash**, you have the right shell to work from and you can skip to the next section.

1. If not, select the drop-down and choose **Select Default Profile**.



2. Select **Git Bash**.



3. Change directory to the folder containing your ARM template file.

Sign in to Azure

In the terminal window, run this command to sign in to Azure.

```
az login
```

In the browser window that opens, sign in to your account. After you sign in, a list of the subscriptions associated with this account displays in the terminal. The default subscription is marked with an asterisk (*). If you have multiple subscriptions, select the subscription you want to use for this exercise.

Create and set the default resource group

```
az group create --name <resource-group-name> --location <location>
```

Replace *<resource-group-name>* with a unique name for your resource group. Replace *<location>* with the Azure region closest to you. For example, use *eastus* for East US.

By setting the default resource group, you can omit that parameter from the Azure CLI commands in this exercise. To set the resource group, run the following command.

```
az configure --defaults group="<resource-group-name>"
```

Replace *<resource-group-name>* with your resource group name.

Deploy the template to Azure

Run the following commands to deploy the ARM template to Azure. The ARM template doesn't have any resources yet, so there aren't any resources created. You should get a successful deployment.

```
templateFile="azuredeploy.json"
today=$(date +"%d-%b-%Y")
DeploymentName="blanktemplate-$today

az deployment group create \
  --name $DeploymentName \
  --template-file $templateFile
```

The top section of the preceding code sets the Azure CLI variables, which include the path to the template file to deploy and the name of the deployment. The bottom section, `az deployment group create`, deploys the template to Azure. Notice that the deployment name is `blanktemplate` with the date as a suffix.

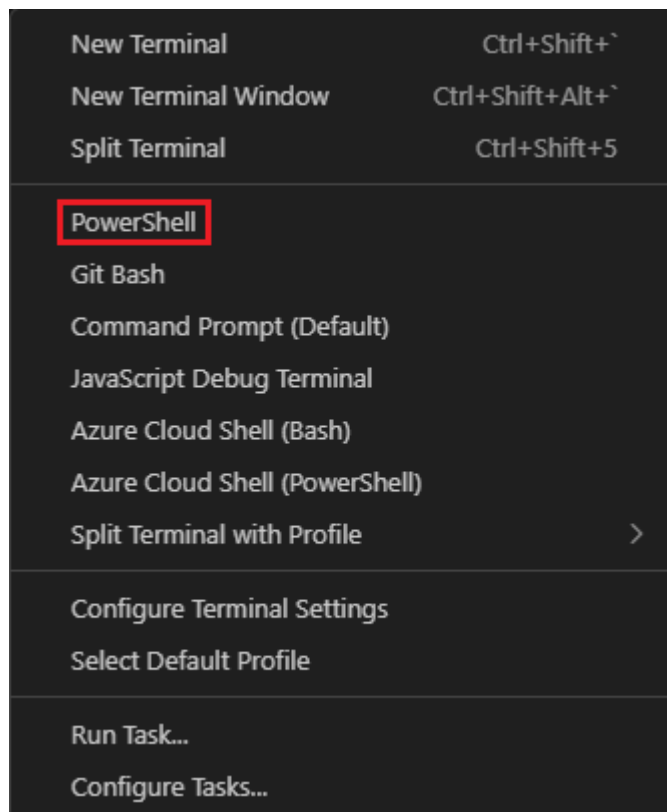
You should see `Running...` in the terminal.

To deploy this template to Azure, you need to sign in to your Azure account from the Visual Studio Code terminal. Be sure that Azure PowerShell Tools are installed from the Visual Studio Code Extensions.

1. In the command bar, select **Terminal > New Terminal** to open a PowerShell window.
2. If the command bar of the terminal window shows **PowerShell**, you have the right shell from which to work, and you can skip to the next section.



1. If not, select the down arrow and in the dropdown list select PowerShell. If that option is missing, then select **Select Default Profile**.
2. In the input field, scroll down and select **PowerShell**.



3. Change directory to the folder containing your ARM template files.

Sign in to Azure by using Azure PowerShell

From the terminal in Visual Studio Code, run the following command to sign in to Azure. A browser opens so you can sign in to your account.

```
Connect-AzAccount
```

In the browser window that opens (the browser window could be opened behind the current window, minimize the current window to see it), sign in to your account. After you sign in, a list of the subscriptions associated with this account displays in the terminal. The default subscription is marked

with an asterisk (*). If you have multiple subscriptions, select the subscription you want to use for this exercise.

Deploy the template to Azure

```
New-AzResourceGroup -Name <ResourceGroupName> -Location <Location>
```

Replace with a unique name for your resource group. Replace with the Azure region closest to you. For example, use eastus for East US.

By setting the default resource group, you can omit that parameter from the Azure CLI commands in this exercise. To set the resource group, run the following command.

```
Set-AzDefault -ResourceGroupName <ResourceGroupName>
```

replace *<ResourceGroupName>* with your resource group name.

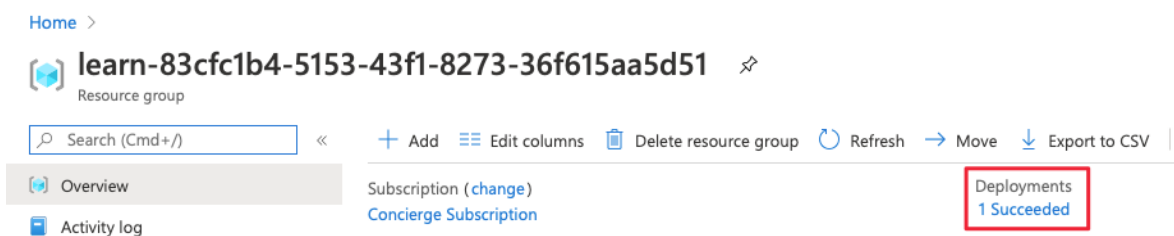
Deploy the template to Azure by running the following commands. The ARM template doesn't have any resources yet, so there aren't any resources created.

```
$templateFile="azuredeploy.json"
$today=Get-Date -Format "MM-dd-yyyy"
$deploymentName="blanktemplate-"+$today
New-AzResourceGroupDeployment `
  -Name $deploymentName `
  -TemplateFile $templateFile
```

The top section of the preceding code sets Azure PowerShell variables, which includes the path to the deployment file and the name of the deployment. Then, the `New-AzResourceGroupDeployment` command deploys the template to Azure. Notice that the deployment name is `blanktemplate` with the date as a suffix.

When you deploy your ARM template to Azure, go to the [Azure portal](#).

1. In the resource menu, select **Resource groups**.
2. Select the resource group you created in this exercise.
3. On the **Overview** pane, you see that one deployment succeeded.



4. Select **1 Succeeded** to see the details of the deployment.

learn-a0a0a0a0-bbbb-cccc-dddd-e1e1e1e1e1e1 | Deployments

Resource group

Search (Cmd+/) << Refresh Cancel Redeploy Delete View template

Overview

Activity log

Access control (IAM)

Tags

Events

Filter by deployment name or resources in the deployment...

Deployment name	Status	Last modified	Duration
blanktemplate-05-Jun-2020	Succeeded	6/5/2020, 12:00:21 PM	2 seconds

5. Select **blanktemplate** to see what resources were deployed. In this case, it's empty because you didn't specify any resources in the template yet.

Home > learn-83cfc1b4-5153-43f1-8273-36f615aa5d51 | Deployments >

blanktemplate-05-Jun-2020 | Overview

Deployment

Search (Cmd+/) << Delete Cancel Redeploy Refresh

Overview

Inputs

Outputs

Template

Your deployment is complete

Deployment name: blanktemplate-05-Jun-2020
Subscription: Concierge Subscription
Resource group: learn-83cfc1b4-5153-43f1-8273-36f615aa5d51

Start time: 6/5/2020, 12:00:19 PM
Correlation ID: aaaa0000-bb11-2222-33cc-444444444444

Deployment details (Download)

Resource	Type	Status	Operation details
No results.			

6. Leave the page open in your browser so that you can check on deployments again.

Add a resource to the ARM template

In the previous task, you learned how to create a blank template and deploy it. Now, you're ready to deploy an actual resource. In this task, you add an Azure storage account resource to the ARM template.

1. In the *azuredeploy.json* file in Visual Studio Code, update the file so it looks like:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deployme
  "contentVersion": "1.0.0.0",
  "parameters": {},
  "functions": [],
  "variables": {},
  "resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "apiVersion": "2025-01-01",
      "name": "storageaccount1",
      "tags": {
        "displayName": "storageaccount1"
      },
      "location": "[resourceGroup().location]",
```



```

        "kind": "StorageV2",
        "sku": {
            "name": "Standard_LRS"
        }
    },
    ],
    "outputs": {}
}

```

2. Change the values of the resource *name* and *displayName* to something unique (for example, **learnexercise12321**). This name must be globally unique across Azure, contain 3 to 24 characters, and include only lowercase letters, numbers, and hyphens.
3. The resource location is set to the same location as the resource group where the resource is deployed. Leave the default here.
4. Save the file.

Deploy the updated ARM template

Here, you change the name of the deployment to better reflect what this deployment does.

Run the following Azure CLI commands in the terminal. This snippet is the same code you used previously, but the name of the deployment is changed.

```

templateFile="azuredeploy.json"
today=$(date +%d-%b-%Y)
DeploymentName="addstorage-"$today

az deployment group create \
  --name $DeploymentName \
  --template-file $templateFile

```

Run the following Azure PowerShell commands in the terminal. This snippet is the same code you used previously, but the name of the deployment is changed.

```

$templateFile="azuredeploy.json"
$today=Get-Date -Format "MM-dd-yyyy"
$deploymentName="addstorage-"+"$today"
New-AzResourceGroupDeployment `
  -Name $deploymentName `
  -TemplateFile $templateFile

```

Check your deployment

1. When the deployment finishes, go back to the Azure portal in your browser. Go to your resource group, and you see that there are now **2 Succeeded** deployments. Select this link.

Notice that both deployments are in the list.

Home > **learn-a0a0a0a0-bbbb-cccc-dddd-e1e1e1e1e1** | Deployments ✕

Resource group

Search (Cmd+/) << Refresh Cancel Redeploy Delete View template

Overview

Activity log

Access control (IAM)

Tags

Events

Filter by deployment name or resources in the deployment...

Deployment name	Status	Last modified	Duration	Related events
☐ addstorage-05-Jun-2020	✓ Succeeded	6/5/2020, 12:48:02 PM	3 seconds	Related events
☐ blanktemplate-05-Jun-2020	✓ Succeeded	6/5/2020, 12:00:21 PM	2 seconds	Related events

2. Select **addstorage**.

addstorage-05-Jun-2020 | Overview ✕

Deployment

Search (Cmd+/) << Delete Cancel Redeploy Refresh

Overview

Inputs

Outputs

Template

✓ Your deployment is complete

Deployment name: addstorage-05-Jun-2020
Subscription: [Concierge Subscription](#)
Resource group: [learn-83cfc1b4-5153-43f1-8273-36f615aa5d51](#)

Start time: 6/5/2020, 12:47:59 PM
Correlation ID: aaaa0000-bb11-2222-33cc-444444ddddd

Deployment details [\(Download\)](#)

Resource	Type	Status	Operation details
✓ teststorage55555	Microsoft.Storage/storag...	OK	Operation details

Notice that the storage account is deployed.

4. Add flexibility to your Azure Resource Manager template by using parameters and outputs

<https://learn.microsoft.com/en-us/training/modules/create-azure-resource-manager-template-vs-code/4-add-flexibility-arm-template>

Add flexibility to your Azure Resource Manager template by using parameters and outputs

Completed

In the last unit, you created an Azure Resource Manager (ARM) template and added an Azure storage account to it. You might notice that there's a problem with your template. The storage account name is hardcoded. You can only use this template to deploy the same storage account every time. To deploy a storage account with a different name, you have to create a new template, which isn't a practical way to automate your deployments. The storage account SKU is also hardcoded, which

means you can't vary the type of storage account for different environments. Recall that in our scenario, each deployment might have a different type of storage account. You can make your template more reusable by adding a parameter for the storage account SKU.

In this unit, you learn about the *parameters* and *outputs* sections of the template.

ARM-template parameters

ARM-template parameters let you customize the deployment by providing values that are tailored for a particular environment. For example, you pass in different values based on whether you're deploying to an environment for development, test, production, or others. For example, the previous template uses the *Standard_LRS* storage account SKU. You can reuse this template for other deployments that create a storage account by making the name of the storage account SKU a parameter. Then, you pass in the name of the SKU you want for this particular deployment when the template is deployed. You can do this step either at the command line or by using a parameter file.

In the `parameters` section of the template, you specify which values you can input when you deploy the resources. You're limited to 256 parameters in a template. Parameter definitions can use most template functions.

The available properties for a parameter are:

```
"parameters": {
  "<parameter-name>": {
    "type": "<type-of-parameter-value>",
    "defaultValue": "<default-value-of-parameter>",
    "allowedValues": [
      "<array-of-allowed-values>"
    ],
    "minValue": <minimum-value-for-int>,
    "maxValue": <maximum-value-for-int>,
    "minLength": <minimum-length-for-string-or-array>,
    "maxLength": <maximum-length-for-string-or-array-parameters>,
    "metadata": {
      "description": "<description-of-the-parameter>"
    }
  }
}
```

The allowed types of parameters are:

- string
- secureString
- integers
- boolean
- object
- secureObject
- array

Recommendations for using parameters

Use parameters for settings that vary according to the environment; for example, SKU, size, or capacity. Also use parameters for resource names that you want to specify yourself for easy identification or to comply with internal naming conventions. Provide a description for each parameter, and use default values whenever possible.

For security reasons, never hardcode or provide default values for usernames and/or passwords in templates. Always use parameters for usernames and passwords (or secrets). Use *secureString* for all passwords and secrets. If you pass sensitive data in a JSON object, use the *secureObject* type. Template parameters with *secureString* or *secureObject* types can't be read or harvested after the deployment of the resource.

Use parameters in an ARM template

In the parameters section of the ARM template, specify the parameters that you can input when you deploy the resources. You're limited to 256 parameters in a template.

Here's an example of a template file with a parameter for the storage-account SKU defined in the template's `parameters` section. You can provide a default for the parameter to be used if no value is specified at execution.

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

Then, use the parameter in the resource definition. The syntax is `[parameters('name of the parameter')]`. Then, when you deploy you use the `parameters` function. In the next module, you learn more about functions.

```
"resources": [
  {
    "type": "Microsoft.Storage/storageAccounts",
    "apiVersion": "2025-01-01",
    "name": "learntemplatestorage123",
    "location": "[resourceGroup().location]",
    "sku": {
      "name": "[parameters('storageAccountType')]"
    }
  }
]
```

```

    },
    "kind": "StorageV2",
    "properties": {
      "supportsHttpsTrafficOnly": true
    }
  }
]

```

When you deploy the template, you can provide a value for the parameter. Notice the last line in the following command:

- [Azure CLI](#)
- [PowerShell](#)

```

templateFile="azuredeploy.json"
az deployment group create \
  --name testdeployment1 \
  --template-file $templateFile \
  --parameters storageAccountType=Standard_LRS

```

ARM template outputs

In your ARM template's outputs section, you can specify the values that are returned after a successful deployment. Here are the elements that make up the outputs section.

```

"outputs": {
  "<output-name>": {
    "condition": "<boolean-value-whether-to-output-value>",
    "type": "<type-of-output-value>",
    "value": "<output-value-expression>",
    "copy": {
      "count": <number-of-iterations>,
      "input": <values-for-the-variable>
    }
  }
}

```

Element	Description
output-name	Must be a valid JavaScript identifier.
condition	(Optional) A Boolean value that indicates whether this output value is returned. When true, the value is included in the output for the deployment. When false, the output value is skipped for this deployment. When not specified, the default value is true.
type	The type of the output value.
value	(Optional) A template language expression to be evaluated and returned as an output value.
copy	(Optional) Copy is used to return more than one value for an output.

Use outputs in an ARM template

Here's an example to output the storage account's endpoints:

```
"outputs": {  
  "storageEndpoint": {  
    "type": "object",  
    "value": "[reference('learntemplatestorage123').primaryEndpoints]"  
  }  
}
```

Notice the `reference` part of the expression. This function gets the runtime state of the storage account.

Deploy an ARM template again

Recall that ARM templates are *idempotent*, which means you can deploy the template to the same environment again, and if nothing changes in the template, nothing changes in the environment. If a change is made to the template (for example, you change a parameter value), only that change is deployed. Your template can contain all of the resources you need for your Azure solution, and you can safely execute a template again. Resources are created only if they don't already exist, and updated only if there's a change.

5. Exercise - Add parameters and outputs to your Azure Resource Manager template

<https://learn.microsoft.com/en-us/training/modules/create-azure-resource-manager-template-vs-code/5-exercise-parameters-output>

Exercise - Add parameters and outputs to your Azure Resource Manager template

Completed

In this exercise, you add a parameter to define the Azure storage account name during deployment. Then, you add a parameter to define which storage-account SKUs are allowed, and define which one to use for this deployment. You also add usefulness to the Azure Resource Manager template (ARM template) by adding an output that you can use later in the deployment process.

Create parameters for the ARM template

Here, you make your ARM template more flexible by adding parameters that can be set at runtime. Create a parameter for the `storageName` value.

1. In the `azuredeploy.json` file in Visual Studio Code, update `"parameters": {}`, so it looks like:

```
"parameters": {
  "storageName": {
    "type": "string",
    "minLength": 3,
    "maxLength": 24,
    "metadata": {
      "description": "The name of the Azure storage resource"
    }
  }
},
```

To format the JSON file correctly, press `Alt+Shift+F`.

2. Use the new parameter in the `resources` block in both the `name` and `displayName` values. The entire file looks like this code example:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deployme
  "contentVersion": "1.0.0.0",
  "parameters": {
    "storageName": {
      "type": "string",
      "minLength": 3,
      "maxLength": 24,
      "metadata": {
        "description": "The name of the Azure storage resource"
      }
    }
  },
  "functions": [],
  "variables": {},
  "resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "apiVersion": "2025-01-01",
      "name": "[parameters('storageName')]",
      "tags": {
        "displayName": "[parameters('storageName')]"
      },
      "location": "[resourceGroup().location]",
      "kind": "StorageV2",
      "sku": {
        "name": "Standard_LRS"
      }
    }
  ]
}
```

```
    ],  
    "outputs": {}  
  }  
}
```

3. Save the file.

Deploy the parameterized ARM template

Here, you change the name of the deployment to better reflect what this deployment does and fill in a value for the new parameter.

Run the following Azure CLI commands in the terminal. This script is identical to the one you used Previously, except the deployment name has been changed. Enter a unique value for the `storageName` parameter. It must be globally unique across Azure, contain 3 to 24 characters, and include only lowercase letters, numbers, and hyphens. You can reuse the unique name you created in the previous unit; if you do, Azure will update the existing resource instead of creating a new one.

```
templateFile="azuredeploy.json"  
today=$(date +%d-%b-%Y)  
DeploymentName="addnameparameter-"$today  
  
az deployment group create \  
  --name $DeploymentName \  
  --template-file $templateFile \  
  --parameters storageName={your-unique-name}
```

Run the following Azure PowerShell commands in the terminal. This script is identical to the one you used earlier, except the deployment name has been changed. Enter a unique value for the `storageName` parameter. It must be globally unique across Azure, contain 3 to 24 characters, and include only lowercase letters, numbers, and hyphens. You can reuse the unique name you created in the previous unit; if you do, Azure will update the existing resource instead of creating a new one.

```
$templateFile="azuredeploy.json"  
$today=Get-Date -Format "MM-dd-yyyy"  
$deploymentName="addnameparameter-"+"$today"  
New-AzResourceGroupDeployment `  
  -Name $deploymentName `  
  -TemplateFile $templateFile `  
  -storageName {your-unique-name}
```

Check your deployment

1. When the deployment finishes, go back to the Azure portal in your browser. Go to your resource group, and see that there are now **3 Succeeded** deployments. Select this link.

Notice that all three deployments are in the list.

2. Explore the *addnameparameter* deployment as you did previously.

Add another parameter that limits allowed values

Here, you use parameters to limit the values allowed for a parameter.

1. Add a new parameter named `storageSKU` to the `parameters` section of the `azuredeploy.json` file.

```
// This is the allowed values for an Azure storage account
"storageSKU": {
  "type": "string",
  "defaultValue": "Standard_LRS",
  "allowedValues": [
    "Standard_LRS",
    "Standard_GRS",
    "Standard_RAGRS",
    "Standard_ZRS",
    "Premium_LRS",
    "Premium_ZRS",
    "Standard_GZRS",
    "Standard_RAGZRS"
  ]
}
```

The first line is a comment. ARM templates support `//` and `/* */` comments.

2. Update **resources** to use the `storageSKU` parameter. If you take advantage of IntelliSense in Visual Studio Code, it makes this step easier.

```
"sku": {
  "name": "[parameters('storageSKU')]"
}
```

The entire file looks like this code example:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deployme
  "contentVersion": "1.0.0.0",
  "parameters": {
    "storageName": {
      "type": "string",
      "minLength": 3,
      "maxLength": 24,
      "metadata": {
        "description": "The name of the Azure storage resource"
      }
    },
    "storageSKU": {
      "type": "string",
      "defaultValue": "Standard_LRS",
      "allowedValues": [
        "Standard_LRS",
        "Standard_GRS",
```

```

        "Standard_RAGRS",
        "Standard_ZRS",
        "Premium_LRS",
        "Premium_ZRS",
        "Standard_GZRS",
        "Standard_RAGZRS"
    ]
  },
  "functions": [],
  "variables": {},
  "resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "apiVersion": "2025-01-01",
      "name": "[parameters('storageName')]",
      "tags": {
        "displayName": "[parameters('storageName')]"
      },
      "location": "[resourceGroup().location]",
      "kind": "StorageV2",
      "sku": {
        "name": "[parameters('storageSKU')]"
      }
    }
  ],
  "outputs": {}
}

```

3. Save the file.

Deploy the ARM template

Here, you deploy successfully by using a `storageSKU` parameter that's in the allowed list. Then, you try to deploy the template by using a `storageSKU` parameter that isn't in the allowed list. The second deployment fails as expected.

1. Deploy the template by running the following commands. Fill in a unique name for the `storageName` parameter. It must be globally unique across Azure, contain 3 to 24 characters, and include only lowercase letters, numbers, and hyphens. You can reuse the unique name you created in the previous unit; if you do, Azure will update the existing resource instead of creating a new one.

```

templateFile="azuredeploy.json"
today=$(date +%d-%b-%Y)
DeploymentName="addSkuParameter-$today

az deployment group create \
  --name $DeploymentName \
  --template-file $templateFile \
  --parameters storageSKU=Standard_GRS storageName={your-unique-name}

```

Allow this deployment to finish. This deployment succeeds as expected. Your list of allowed values, prevents your template's users from passing in parameter values that don't work for the resource. Let's see what happens when you provide an invalid SKU.

2. Run the following commands to deploy the template with a parameter that isn't allowed. Here, you changed the `storageSKU` parameter to **Basic**. Fill in a unique name for the `storageName` parameter. It must be globally unique across Azure, contain 3 to 24 characters, and include only lowercase letters, numbers, and hyphens. You can reuse the unique name you created in the previous unit; if you do, Azure will update the existing resource instead of creating a new one.

```
templateFile="azuredeploy.json"
today=$(date +%d-%b-%Y)
DeploymentName="addSkuParameter-"$today

az deployment group create \
  --name $DeploymentName \
  --template-file $templateFile \
  --parameters storageSKU=Basic storageName={your-unique-name}
```

This deployment fails. Notice the error.

```
New-AzResourceGroupDeployment: 2:53:10 PM - Error: Code=InvalidTemplate; Message=Deployment template validation failed: 'The provided value for the template parameter 'storageSKU' is not valid. The value 'Basic' is not part of the allowed value(s): 'Standard_LRS,Standard_GRS,Standard_RAGRS,Standard_ZRS,Premium_LRS,Premium_ZRS,Standard_GZRS,Standard_RAGZRS'.'.
New-AzResourceGroupDeployment: The deployment validation failed
```

1. Deploy the template by running the following commands. Fill in a unique name for the `storageName` parameter. It must be globally unique across Azure, contain 3 to 24 characters, and include only lowercase letters, numbers, and hyphens. You can reuse the unique name you created in the previous unit; if you do, Azure will update the existing resource instead of creating a new one.

```
$today=Get-Date -Format "MM-dd-yyyy"
$deploymentName="addSkuParameter-"$today
New-AzResourceGroupDeployment `
  -Name $deploymentName `
  -TemplateFile $templateFile `
  -storageName {your-unique-name} `
  -storageSKU Standard_GRS
```

Allow this deployment to finish. This deployment succeeds as expected. Your list of allowed values, prevents your template's users from passing in parameter values that don't work for the resource. Let's see what happens when you provide an invalid SKU.

2. Run the following commands to deploy the template with a parameter that isn't allowed. Here, you changed the `storageSKU` parameter to **Basic**. Fill in a unique name for the `storageName` parameter. It must be globally unique across Azure, contain 3 to 24 characters, and include only lowercase letters, numbers, and hyphens. You can reuse the unique name you created in

the previous unit; if you do, Azure will update the existing resource instead of creating a new one.

```
$today=Get-Date -Format "MM-dd-yyyy"
$deploymentName="addSkuParameter-"+$today"
New-AzResourceGroupDeployment `
  -Name $deploymentName `
  -TemplateFile $templateFile `
  -storageName {your-unique-name} `
  -storageSKU Basic
```

This deployment fails. Notice the error.

```
New-AzResourceGroupDeployment: 2:53:10 PM - Error: Code=InvalidTemplate; Message=Deployment template validation failed: 'The provided value for the template parameter 'storageSKU' is not valid. The value 'Basic' is not part of the allowed value(s): 'Standard_LRS,Standard_GRS,Standard_RAGRS,Standard_ZRS,Premium_LRS,Premium_ZRS,Standard_GZRS,Standard_RAGZRS'.'.
New-AzResourceGroupDeployment: The deployment validation failed
```

Add output to the ARM template

Here, you add to the `outputs` section of the ARM template to output the endpoints for the storage account resource.

1. In the `azuredeploy.json` file in Visual Studio Code, update `"outputs": {}`, so it looks like:

```
"outputs": {
  "storageEndpoint": {
    "type": "object",
    "value": "[reference(parameters('storageName')).primaryEndpoints]"
  }
}
```

2. Save the file.

Deploy the ARM template with an output

Here, you deploy the template and see the endpoints output as JSON. You need to fill in a unique name for the `storageName` parameter. It must be globally unique across Azure, contain 3 to 24 characters, and include only lowercase letters, numbers, and hyphens. You can reuse the unique name you created in the previous unit; if you do, Azure will update the existing resource instead of creating a new one.

1. Deploy the template by running the following commands. Be sure to replace `{your-unique-name}` with a string unique to you.

```
templateFile="azuredeploy.json"
today=$(date +%d-%b-%Y)
DeploymentName="addoutputs-"+$today
```

```
az deployment group create \
  --name $DeploymentName \
  --template-file $templateFile \
  --parameters storageSKU=Standard_LRS storageName={your-unique-name}
```

Notice the output.

```
],
"outputs": {
  "storageEndpoint": {
    "type": "Object",
    "value": {
      "blob": "https://storelearnvlmlemv4t3u7i.blob.core.windows.net/",
      "dfs": "https://storelearnvlmlemv4t3u7i.dfs.core.windows.net/",
      "file": "https://storelearnvlmlemv4t3u7i.file.core.windows.net/",
      "queue": "https://storelearnvlmlemv4t3u7i.queue.core.windows.net/",
      "table": "https://storelearnvlmlemv4t3u7i.table.core.windows.net/",
      "web": "https://storelearnvlmlemv4t3u7i.z22.web.core.windows.net/"
    }
  }
}
```

1. Deploy the template by running the following commands. Be sure to replace *{your-unique-name}* with a string unique to you.

```
$today=Get-Date -Format "MM-dd-yyyy"
$deploymentName="addOutputs-"+$today
New-AzResourceGroupDeployment `
  -Name $deploymentName `
  -TemplateFile $templateFile `
  -storageName {your-unique-name} `
  -storageSKU Standard_LRS
```

Notice the output.

```
],
"outputs": {
  "storageEndpoint": {
    "type": "Object",
    "value": {
      "blob": "https://storelearnvlmlemv4t3u7i.blob.core.windows.net/",
      "dfs": "https://storelearnvlmlemv4t3u7i.dfs.core.windows.net/",
      "file": "https://storelearnvlmlemv4t3u7i.file.core.windows.net/",
      "queue": "https://storelearnvlmlemv4t3u7i.queue.core.windows.net/",
      "table": "https://storelearnvlmlemv4t3u7i.table.core.windows.net/",
      "web": "https://storelearnvlmlemv4t3u7i.z22.web.core.windows.net/"
    }
  }
}
```

Check your output deployment

In the Azure portal, go to your *addOutputs* deployment. You can find your output there as well.

addoutputs-08-Jun-2020 | Outputs

Deployment

Search (Cmd+ /)

storageEndpoint

{"dfs": "https://learnstorage484848484.dfs.core.wind..."}

Overview

Inputs

Outputs

Template

6. Module assessment

<https://learn.microsoft.com/en-us/training/modules/create-azure-resource-manager-template-vs-code/6-knowledge-check>

Module assessment

Completed

7. Summary

<https://learn.microsoft.com/en-us/training/modules/create-azure-resource-manager-template-vs-code/7-summary>

Summary

Completed

In this module, you learned about Azure Resource Manager templates (ARM templates) and used them to deploy a storage account to Azure. You made the template more flexible by adding parameters and got output from the execution of the template.

In summary, you:

- Implemented an ARM template by using Visual Studio Code.
- Declared resources and added flexibility to your template by adding resources, parameters, and outputs.

Learn more

To learn more, check out the following articles:

- [Download Azure Resource Manager Tools for Visual Studio Code](#)
- [Understand the structure and syntax of ARM templates](#)
- [Learn about Azure Resource Manager](#)
- [How to install the Azure CLI](#)
- [How to install Azure PowerShell](#)
- [Learn about resource providers for Azure services](#)
- [Define resources in Azure Resource Manager templates](#)
- [Learn about Outputs in Azure Resource Manager templates](#)
- [Learn about Parameters in Azure Resource Manager templates](#)