**Introduction**

Completed100 XP

* 2 minutes

JSON Azure Resource Manager templates (ARM templates) allow you to specify your project's infrastructure in a declarative and reusable way. The templates can be versioned and saved 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 each partner company will have its own solution. Different policies for each deployment will be implemented 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, you're introduced to ARM template structure. You also practice creating and deploying an ARM template to Azure.

**Note**

Bicep is a new 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 [**Deploy and manage resources in Azure by using Bicep**](https://learn.microsoft.com/en-us/training/paths/fundamentals-bicep/).

**Learning objectives**

In this module, you will:

* Implement an 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](https://azure.microsoft.com/free).
* [Visual Studio Code](https://code.visualstudio.com/) installed locally.
* The [Azure Resource Manager Tools for Visual Studio Code](https://marketplace.visualstudio.com/items?itemName=msazurermtools.azurerm-vscode-tools) extension installed locally.
* Either:
  + The latest [Azure CLI](https://learn.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest) tools installed locally.
  + The latest [Azure PowerShell](https://learn.microsoft.com/en-us/powershell/azure/install-az-ps?view=azps-4.3.0) installed locally.

**Next unit: Explore Azure Resource Manager template structure**

# Explore Azure Resource Manager template structure

Completed100 XP

* 7 minutes

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 enables 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

## 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 will look like without describing its 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.

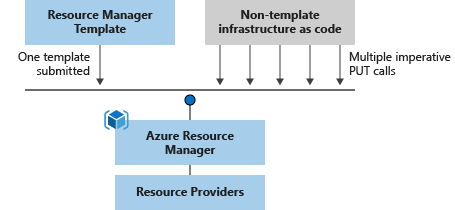
ARM templates allow you to declare what you intend to deploy without having to write the sequence of programming commands to create it. In an ARM template, you specify the resources and the properties for those resources. Then [Azure Resource Manager](https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/overview) 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 the deployment of the resources so they're created in the correct order. When possible, resources will also be 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 will succeed.

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](https://azure.microsoft.com/services/devops/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 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. These values can be provided by 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. |
| **functions** | An optional section where you can define [user-defined functions](https://learn.microsoft.com/en-us/azure/azure-resource-manager/templates/template-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 will be 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'll 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](https://learn.microsoft.com/en-us/powershell/azure/install-az-ps) or the [Azure CLI](https://learn.microsoft.com/en-us/cli/azure/install-azure-cli) installed locally.

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

* [Azure CLI](https://learn.microsoft.com/en-us/training/modules/create-azure-resource-manager-template-vs-code/2-explore-template-structure?tabs=azure-cli#tabpanel_1_azure-cli)
* [PowerShell](https://learn.microsoft.com/en-us/training/modules/create-azure-resource-manager-template-vs-code/2-explore-template-structure?tabs=azure-cli#tabpanel_1_azure-powershell)

Azure CLICopy

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](https://learn.microsoft.com/en-us/training/modules/create-azure-resource-manager-template-vs-code/2-explore-template-structure?tabs=azure-cli#tabpanel_2_azure-cli)
* [PowerShell](https://learn.microsoft.com/en-us/training/modules/create-azure-resource-manager-template-vs-code/2-explore-template-structure?tabs=azure-cli#tabpanel_2_azure-powershell)

Azure CLICopy

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](https://learn.microsoft.com/en-us/cli/azure/deployment/group" \l "az-deployment-group-create) or the Azure PowerShell command [New-AzResourceGroupDeployment](https://learn.microsoft.com/en-us/powershell/module/az.resources/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, it is recommended to use 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](https://learn.microsoft.com/en-us/training/modules/create-azure-resource-manager-template-vs-code/2-explore-template-structure?tabs=azure-cli#tabpanel_3_azure-cli)
* [PowerShell](https://learn.microsoft.com/en-us/training/modules/create-azure-resource-manager-template-vs-code/2-explore-template-structure?tabs=azure-cli#tabpanel_3_azure-powershell)

To run this deployment command, you must have the [latest version](https://learn.microsoft.com/en-us/cli/azure/install-azure-cli) of Azure CLI.

Azure CLICopy

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 deployment of the linked template. You can store and secure the linked template by using a SAS token.

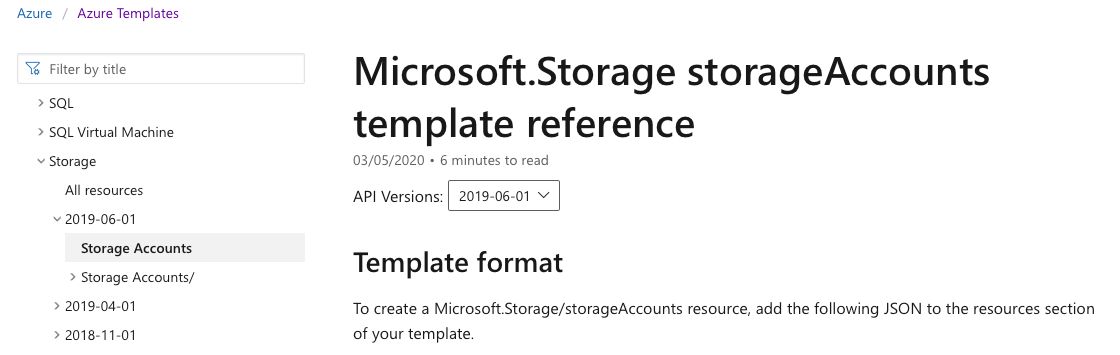
A CI/CD pipeline automates the creation and deployment of development projects, which includes ARM template projects. The two most common pipelines used for template deployment are [Azure Pipelines](https://learn.microsoft.com/en-us/training/paths/deploy-applications-with-azure-devops/) or [GitHub Actions](https://learn.microsoft.com/en-us/training/paths/automate-workflow-github-actions/).

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

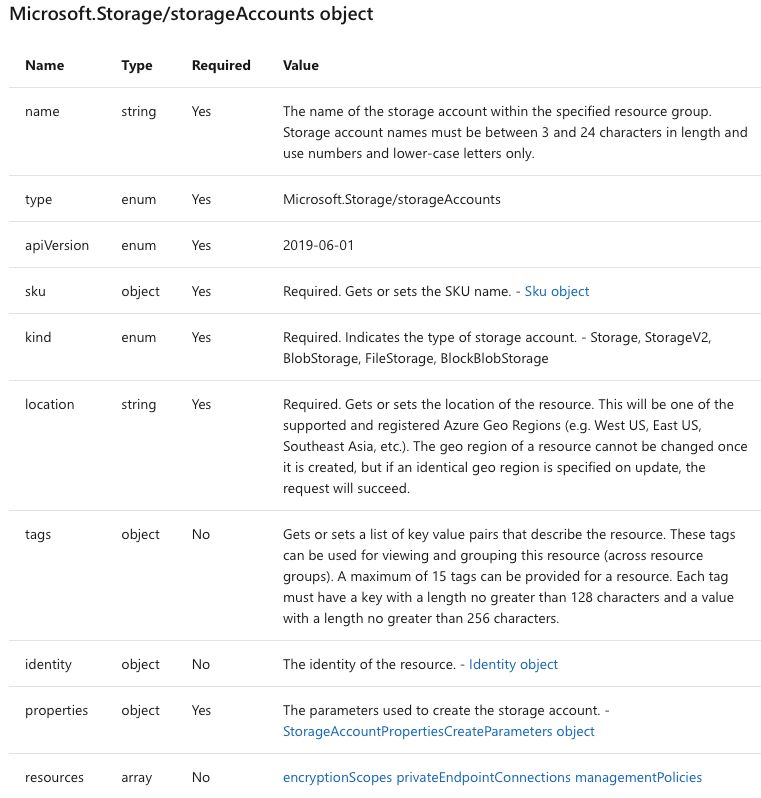
## Add resources to the template

To add a resource to your template, you'll 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'll need the Microsoft.Storage resource provider. One of the types for this provider is storageAccount. So your resource type will be displayed as Microsoft.Storage/storageAccounts. You can use a list of [resource providers for Azure services](https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/azure-services-resource-providers) to find the providers you need.

After you've defined 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](https://learn.microsoft.com/en-us/azure/templates). View the list in the left column to find the resource. Notice that the properties are sorted by API version.



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



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

JSONCopy

{

"$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": "2019-06-01",

"name": "learntemplatestorage123",

"location": "westus",

"sku": {

"name": "Standard\_LRS"

},

"kind": "StorageV2",

"properties": {

"supportsHttpsTrafficOnly": true

}

}

],

"outputs": {}

}

## Next unit: Exercise - Create and deploy an Azure Resource Manager template

# Exercise - Create and deploy an Azure Resource Manager template

Completed100 XP

* 10 minutes

Sandbox activated! Time remaining:

3 hr 20 min

You have used 3 of 10 sandboxes for today. More sandboxes will be available tomorrow.

Top of Form

Choose your Azure shell

PowerShellAzure CLI

Bottom of Form

**Note**

The first time you activate a sandbox and accept the terms, your Microsoft account is associated with a new Azure directory named Microsoft Learn Sandbox. You're also added to a special subscription named Concierge Subscription.

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

This exercise uses [Azure Resource Manager Tools for Visual Studio Code](https://marketplace.visualstudio.com/items?itemName=msazurermtools.azurerm-vscode-tools). Be sure to install this extension in Visual Studio Code before starting the exercise.

## Create an ARM template

1. Open Visual Studio Code, and create a new file called azuredeploy.json.
2. The Visual Studio Code ARM template extension comes configured with snippets to help you develop templates. Let's start by adding a blank template. On the first line of the file, enter arm.
3. The VS Code automatically displays several potential choices that start with **arm!**. Select the **Azure Resource Manager (ARM) template**. VS Code automatically processes the schemas and languages for your template.

Your file now looks like this:

JSONCopy

{

"$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",

"contentVersion": "1.0.0.0",

"parameters": {},

"functions": [],

"variables": {},

"resources": [],

"outputs": {}

}

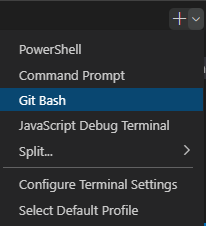
Notice that this file has all of the sections of an ARM template that we described in the previous unit.

1. 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 installed Azure PowerShell Tools from the VS Code Extensions, and sign in to the same account that activated the sandbox.

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 to work from, 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**.



* 1. Select **Terminal > New Terminal** to open a PowerShell terminal window.

### Sign in to Azure by using Azure PowerShell

1. 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.

Azure PowerShellCopy

Connect-AzAccount

**Tip**

The [**Az PowerShell module**](https://learn.microsoft.com/en-us/powershell/azure/new-azureps-module-az) is the replacement of AzureRM and is the recommended version to use for interacting with Azure.

1. Sign in using the account you used to activate the sandbox. After you've signed in, VS Code lists the subscriptions associated with your account in the terminal window. If you activated the sandbox, you see a code block that contains "name": "Concierge Subscription". This is the subscription to use for the rest of the exercise.

### Set the default subscription for all PowerShell commands in this session

1. Run the following command to obtain your subscription(s) and their ID(s). The subscription ID is the second column. Look for Concierge Subscription, and copy the value in the second column. It will look something like cf49fbbc-217c-4eb6-9eb5-a6a6c68295a0:

Azure PowerShellCopy

Get-AzSubscription

1. Run the following command, replacing {Your subscription ID} with the one you copied in the previous step to change your active subscription to the Concierge Subscription.

Azure PowerShellCopy

$context = Get-AzSubscription -SubscriptionId {Your subscription ID}

Set-AzContext $context

1. Run the following command to let the default resource group be the resource group created for you in the sandbox environment. This action lets you omit that parameter from the rest of the Azure PowerShell commands in this exercise.

Azure PowerShellCopy

Set-AzDefault -ResourceGroupName learn-cbe03d57-3f87-4cab-b6a9-e4d1b1f1329c

### Deploy the template to Azure

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

Azure PowerShellCopy

$templateFile="azuredeploy.json"

$today=Get-Date -Format "MM-dd-yyyy"

$deploymentName="blanktemplate-"+"$today"

New-AzResourceGroupDeployment `

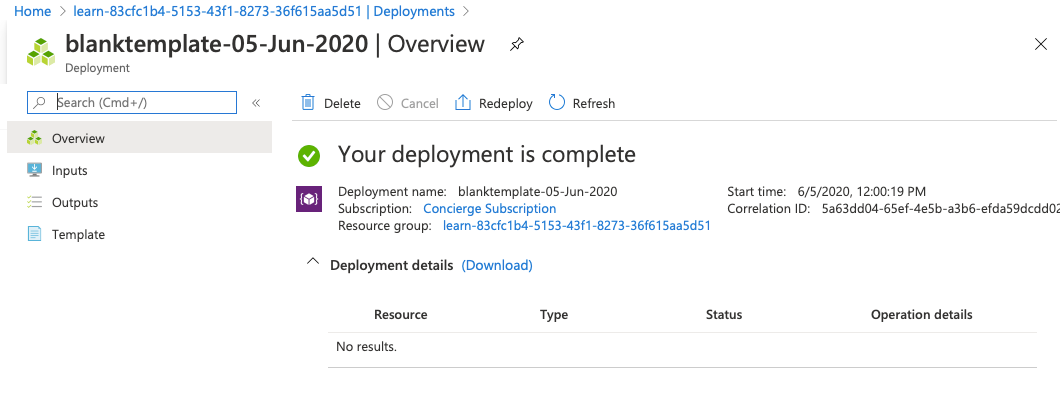
-Name $deploymentName `

-TemplateFile $templateFile

The top section of the preceding code sets Azure PowerShell variables, which includes the path to the deployment path 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've deployed your ARM template to Azure, go to the [Azure portal](https://portal.azure.com/) and make sure you're in the sandbox subscription. To do that, select your avatar in the upper-right corner of the page. Select **Switch directory**. In the list, choose the **Microsoft Learn Sandbox** directory.

1. In the resource menu, select **Resource groups**.
2. Select the learn-cbe03d57-3f87-4cab-b6a9-e4d1b1f1329c resource group.
3. On the **Overview** pane, you see that one deployment succeeded.
4. Select **1 Succeeded** to see the details of the deployment.
5. Select blanktemplate to see what resources were deployed. In this case, it will be empty because you didn't specify any resources in the template yet.



1. Leave the page open in your browser. You'll 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 by using a snippet from the Azure Resource Manager Tools extension for Visual Studio Code.

1. In the azuredeploy.json file in Visual Studio Code, place your cursor inside the brackets in the resources block "resources":[],.
2. Enter storage inside the brackets. A list of related snippets appears. Select **arm-storage**.

Your file will look like this:

JSONCopy

{

"$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",

"contentVersion": "1.0.0.0",

"parameters": {},

"functions": [],

"variables": {},

"resources": [

{

"name": "storageaccount1",

"type": "Microsoft.Storage/storageAccounts",

"apiVersion": "2019-06-01",

"tags": {

"displayName": "storageaccount1"

},

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

"kind": "StorageV2",

"sku": {

"name": "Premium\_LRS",

"tier": "Premium"

}

}

],

"outputs": {}

}

Values that you should edit are highlighted in the new section of your file and can be navigated by pressing the Tab key.

Notice the tags and location attributes are filled in. The location attribute uses a function to set the location of the resource to the location of the resource group. You'll learn about tags and functions in the next module.

1. Change the values of the resource name and displayName to something unique, (for example, **learnexercise12321**). This name must be unique across all of Azure, so choose something unique to you.
2. Change the value of the sku name from **Premium\_LRS** to **Standard\_LRS**. Change the value of tier to **Standard**. Notice that Visual Studio Code gives you the proper choices for your attribute values in IntelliSense. Delete the default value including the quotation marks, and enter quotation marks to see this work.
3. The location of the resource is set to the location of the resource group where it will be 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 PowerShell commands in the terminal. This snippet is the same code you used previously, but the name of the deployment is changed.

Azure PowerShellCopy

$templateFile="azuredeploy.json"

$today=Get-Date -Format "MM-dd-yyyy"

$deploymentName="addstorage-"+"$today"

New-AzResourceGroupDeployment `

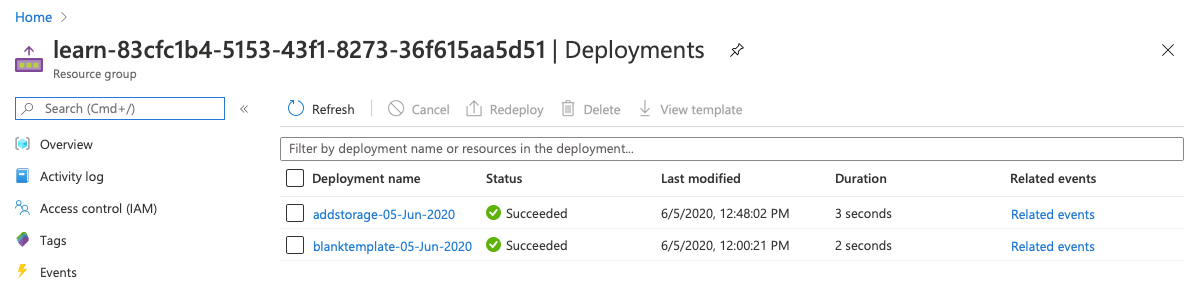
-Name $deploymentName `

-TemplateFile $templateFile

### Check your deployment

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

Notice that both deployments are in the list.



1. Select \*\*addstorage\*\*.

Notice that the storage account has been deployed.

## Next unit: Add flexibility to your Azure Resource Manager template by using parameters and outputs