✓ 100 XP

Introduction

1 minute

Let's say you're a systems administrator at a growing financial services company. Analysts create financial models to help recommend the best investment options to investors.

These financial models are run on virtual machines running on Azure. A model can take between a few minutes and several hours to run. You typically create a new virtual machine to run each new model, then delete it after the analyst collects the results.

The financial services business moves quickly. You find yourself constantly deploying and deleting cloud resources. You initially created resources through the Azure portal, and now you use the Azure CLI and scripts to further automate things. Although you see certain patterns among your deployments, it still takes time to connect virtual machines to networking and storage components. At the end of each run, you need to delete only the components related to that run.

How can you keep up with the pace of your analysts? Azure Resource Manager templates are one way to further automate your deployments.

Learning objectives

In this module, you will:

- Identify what Azure Resource Manager is, and what's in a Resource Manager template
- Deploy a VM using the Azure CLI and a prebuilt Resource Manager template
- Customize your Resource Manager template to configure a basic web server on your VM

Prerequisites

- Experience using Bash and the Azure CLI
- Experience working with Azure resources and resource groups

---

Next unit: Define Azure Resource Manager templates

Continue >

✓ 200 XP

Define Azure Resource Manager templates

7 minutes

If you've been using Azure for a while, you've likely heard about Azure Resource Manager. Let's review Resource Manager's role and define what makes up a Resource Manager template.

What's Azure Resource Manager?

Azure Resource Manager is the interface for managing and organizing cloud resources. Think of Resource Manager as a way to deploy cloud resources.

If you're familiar with Azure resource groups, you know that they enable you to treat sets of related resources as a single unit. Resource Manager is what organizes the resource groups that let you deploy, manage, and delete all of the resources together in a single action.

Think about the financial models you run for your analysts. To run a model, you might need one or more VMs, a database to store data, and a virtual network to enable connectivity between everything. With Resource Manager, you deploy these assets into the same resource group and manage and monitor them together. When you're done, you can delete all of the resources in a resource group in one operation.

What are Resource Manager templates?

A Resource Manager *template* precisely defines all the Resource Manager resources in a deployment. You can deploy a Resource Manager template into a resource group as a single operation.

A Resource Manager template is a JSON file, making it a form of *declarative automation*. Declarative automation means that you define *what* resources you need but not *how* to create them. Put another way, you define what you need and it is Resource Manager's responsibility to ensure that resources are deployed correctly.

You can think of declarative automation similar to how web browsers display HTML files. The HTML file describes *what* elements appear on the page, but doesn't describe *how* to display them. The "how" is the web browser's responsibility.

> **Note**
>
> You may hear others refer to Resource Manager templates as "ARM templates". We prefer the full names "Azure Resource Manager templates" or "Resource Manager templates".

Why use Resource Manager templates?

Using Resource Manager templates will make your deployments faster and more repeatable. For example, you no longer have to create a VM in the portal, wait for it to finish, then create the next VM, and so on. Resource Manager takes care of the entire deployment for you.

Here are some other benefits to consider.

- **Templates improve consistency**

  Resource Manager templates provide a common language for you and others to describe your deployments. Regardless of the tool or SDK used to deploy the template, the structure, format, and expressions inside the template remain the same.

- **Templates help express complex deployments**

  Templates enable you to deploy multiple resources in the correct order. For example, you wouldn't want to deploy a virtual machine before creating OS disk or network interface. Resource Manager maps out each resource and its dependent resources and creates dependent resources first. Dependency mapping helps ensure that the deployment is carried out in the correct order.

- **Templates reduce manual, error-prone tasks**

  Manually creating and connecting resources can be time consuming, and it's easy to make mistakes along the way. Resource Manager ensures that the deployment happens the same way every time.

- **Templates are code**

  Templates express your requirements through code. Think of a template as a type of *infrastructure as code* that can be shared, tested, and versioned like any other piece of software. Also, because templates are code, you can create a "paper trail" that you can follow. The template code documents the deployment. Most users maintain their templates under some kind of revision control, such as Git. When you change the template, its revision history also documents how the template (and your deployment) has evolved over time.

- **Templates promote reuse**

  Your template can contain parameters that are filled in when the template runs. A parameter can define a username or password, a domain name, and so on. Template parameters enable you to create multiple versions of your infrastructure, such as staging and production, but still utilize the exact same template.

- **Templates are linkable**

  Resource Manager templates can be linked together to make the templates themselves modular. You can write small templates that each define a piece of a solution and combine them to create a complete system.

The models your financial analysts run are unique, but you see patterns in the underlying infrastructure. For example, most models require a database to store data. Many models use the same programming languages, frameworks, and operating systems to carry out the details. You can define templates that describe each individual component (compute, storage, networking, and so on), and combine them to meet each analyst's specific needs.

What's in a Resource Manager template?

Note

Here you'll see a few code examples to give you a sense of how each section is structured. Don't worry if what you see is unfamiliar to you — you'll be able to read others' templates and write your own as you gain hands-on experience with them.

You may have used JSON, or JavaScript Object Notation, to send data between servers and web applications. JSON is also a popular way to describe how applications and infrastructure are configured.

JSON allows us to express data stored as an object (such as a virtual machine) in text. A JSON document is essentially a collection of key-value pairs. Each key is a string; its value can be a string, a number, a Boolean expression, a list of values, or an object (which is a collection of other key-value pairs).

A Resource Manager template can contain the following sections. These sections are expressed using JSON notation, but are not related to the JSON language itself.

JSON

Copy

```json
{
    "$schema": "http://schema.management.azure.com/schemas/2015-01-01/de-
ploymentTemplate.json#",
    "contentVersion": "",
    "parameters": {  },
    "variables": {  },
    "functions": [  ],
    "resources": [  ],
    "outputs": {  }
}
```

Let's look at each of these sections in a little more detail.

Parameters

This is where you specify which values are configurable when the template runs. For example, you might allow users of your template to specify a username, password, or domain name.

Here's an example that illustrates two parameters – one for a VM's username and one for its password.

JSON

Copy

```json
"parameters": {
  "adminUsername": {
    "type": "string",
    "metadata": {
      "description": "Username for the Virtual Machine."
    }
  },
  "adminPassword": {
    "type": "securestring",
    "metadata": {
      "description": "Password for the Virtual Machine."
    }
  }
}
```

Variables

This is where you define values that are used throughout the template. Variables can help make your templates easier to maintain. For example, you might define a storage account name one time as a variable and use that variable throughout the template. If the storage account name changes, you need to only update the variable.

Here's an example that illustrates a few variables that describe networking features for a VM.

JSON

Copy

```json
"variables": {
  "nicName": "myVMNic",
  "addressPrefix": "10.0.0.0/16",
  "subnetName": "Subnet",
  "subnetPrefix": "10.0.0.0/24",
  "publicIPAddressName": "myPublicIP",
  "virtualNetworkName": "MyVNET"
}
```

Functions

This is where you define procedures that you don't want to repeat throughout the template. Like variables, functions can help make your templates easier to maintain. Here's an example that creates a function to create a unique name that could be used when creating resources that have globally unique naming requirements.

JSON

```
"functions": [
  {
    "namespace": "contoso",
    "members": {
      "uniqueName": {
        "parameters": [
          {
            "name": "namePrefix",
            "type": "string"
          }
        ],
        "output": {
          "type": "string",
          "value": "[concat(toLower(parameters('namePrefix')),
uniqueString(resourceGroup().id))]"
        }
      }
    }
  }
],
```

Resources

This section is where you define the Azure resources that make up your deployment.

Here's an example that creates a public IP address resource.

JSON

Copy

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

Here, the type of resource is `Microsoft.Network/publicIPAddresses`. Its name is read from the variables section and its location, or Azure region, is read from the parameters section.

Because resource types can change over time, `apiVersion` refers to the version of the resource type you want to use. As resource types evolve and change, you can modify your templates to work with the latest features when you're ready.

Outputs

This is where you define any information you'd like to receive when the template runs. For example, you might want to receive your VM's IP address or FQDN – information you do not know until the deployment runs.

Here's an example that illustrates an output named "hostname". The FQDN value is read from the VM's public IP address settings.

JSON

Copy

```json
"outputs": {
  "hostname": {
    "type": "string",
    "value": "[reference(variables('publicIPAddressName')).dnsSet-
tings.fqdn]"
  }
}
```

How do I write a Resource Manager template?

There are many approaches to writing Resource Manager templates. Although you can write a template from scratch, it's common to start with an existing template and modify it to suit your needs.

Here are a few ways you can get a starter template:

- Use the Azure portal to create a template based on the resources in an existing resource group.
- Start with a template you or your team built that serves a similar purpose.
- Start with an Azure Quickstart template. You'll see how in the next part.

No matter your approach, writing a template involves working with a text editor. You can bring your favorite editor, but Visual Studio Code's Azure Resource Manager Tools extension is specially designed for the task of creating templates. This extension makes it easier to navigate your template code and provides autocompletion for many common tasks.

As you explore and write your templates, you'll want to refer to the documentation to understand what resource types are available and how to use them.

Check your knowledge

1. *Declarative automation* means that:

○ You define both what resources you need as well as the steps needed to create them.

[✓ You define *what* resources you need but not *how* to create them.] ✓

In the case of Resource Manager templates, Resource Manager takes care of these details for you.

○ You define what resources you need, and the system ensures that those resources are always available.

Next unit: Discover Azure Quickstart templates

Continue >

✓ 100 XP

Discover Azure Quickstart templates

5 minutes

Choose which platform you want to run in the cloud

Windows | Linux

Recall that your analysts' financial models are run on Azure virtual machines. To further automate your deployments, you want to move from Azure CLI commands and scripts to Resource Manager templates.

Before you begin, you may wonder what existing templates are available to learn from and build upon.

Here you'll explore what an Azure Quickstart template is and what prebuilt templates are available for you to use right now.

Tip

Prefer Linux or want to try something new? Select **Linux** from the top of this page to deploy a Linux virtual machine.

What are Azure Quickstart templates?

Azure Quickstart templates are Resource Manager templates that are provided by the Azure community. Quickstart templates are available on GitHub.

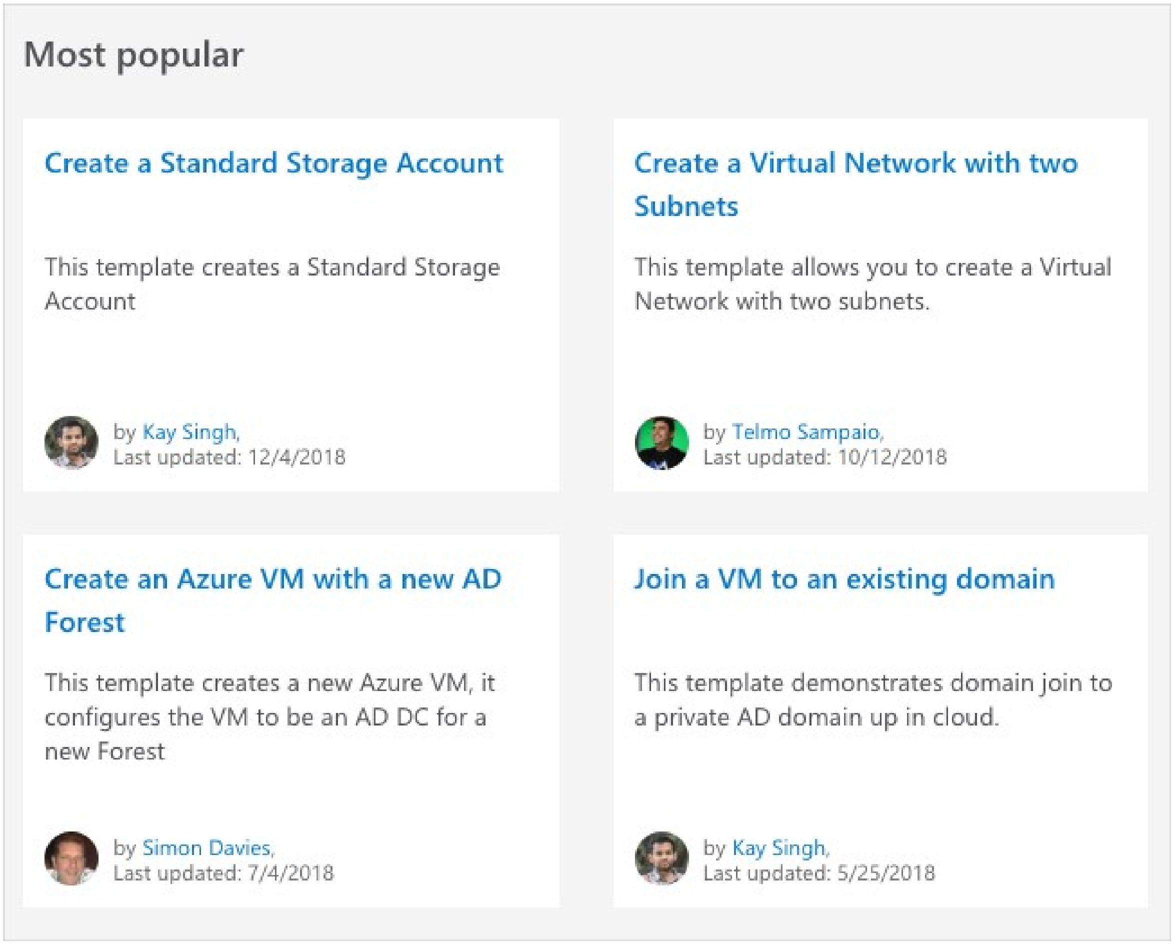
Many templates provide everything you need to deploy your solution. Others might serve as a starting point for your template. Either way, you can study these templates to learn how to best author and structure your own templates.

Discover what's on the Quickstart template gallery

Let's say you want to find a Resource Manager template that brings up a basic VM configuration – one that includes a VM, basic network settings, and storage.

1. Start by browsing to the Quickstart template gallery to see what's available.

   You see a number of popular and recently updated templates. These templates work with both Azure resources and popular software packages.

   Most popular

   Create a Standard Storage Account

   This template creates a Standard Storage Account

   by Kay Singh,
   Last updated: 12/4/2018

   Create a Virtual Network with two Subnets

   This template allows you to create a Virtual Network with two subnets.

   by Telmo Sampaio,
   Last updated: 10/12/2018

   Create an Azure VM with a new AD Forest

   This template creates a new Azure VM, it configures the VM to be an AD DC for a new Forest

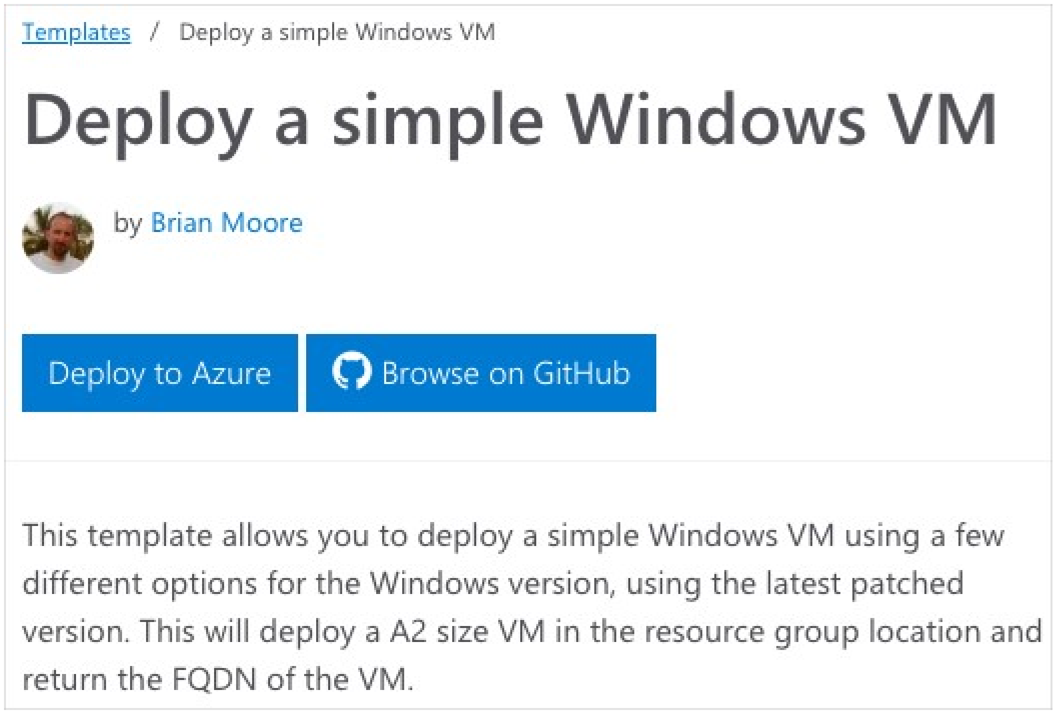
   by Simon Davies,
   Last updated: 7/4/2018

   Join a VM to an existing domain

   This template demonstrates domain join to a private AD domain up in cloud.

   by Kay Singh,
   Last updated: 5/25/2018

2. Let's say you come across the Deploy a simple Windows VM template.


Templates / Deploy a simple Windows VM

Deploy a simple Windows VM

by Brian Moore

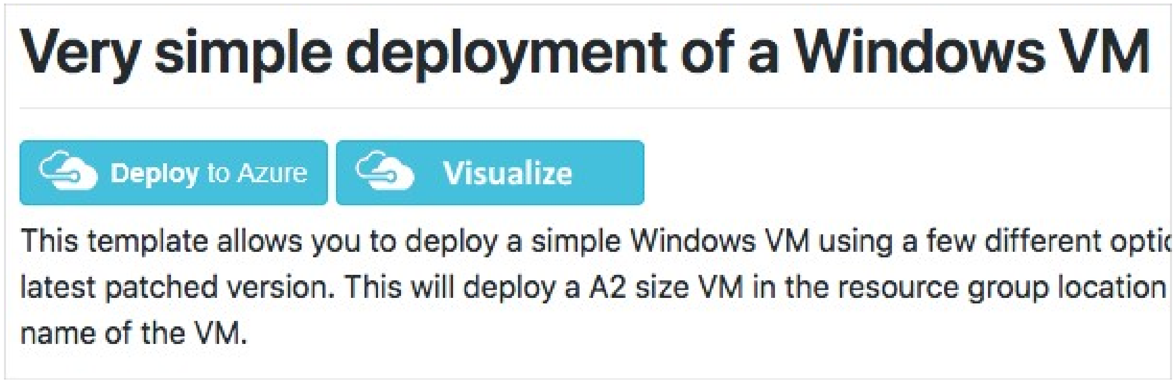
This template allows you to deploy a simple Windows VM using a few different options for the Windows version, using the latest patched version. This will deploy a A2 size VM in the resource group location and return the FQDN of the VM.

The name sounds like exactly what you need. But let's take a closer look to see what this template accomplishes.

The **Deploy to Azure** button enables you to deploy the template directly through the Azure portal. But you won't do that here. Rather, you'll use the Azure CLI to deploy the template from Cloud Shell.

3. Click **Browse on GitHub** to navigate to the template's source code on GitHub.

   You see this.


Very simple deployment of a Windows VM

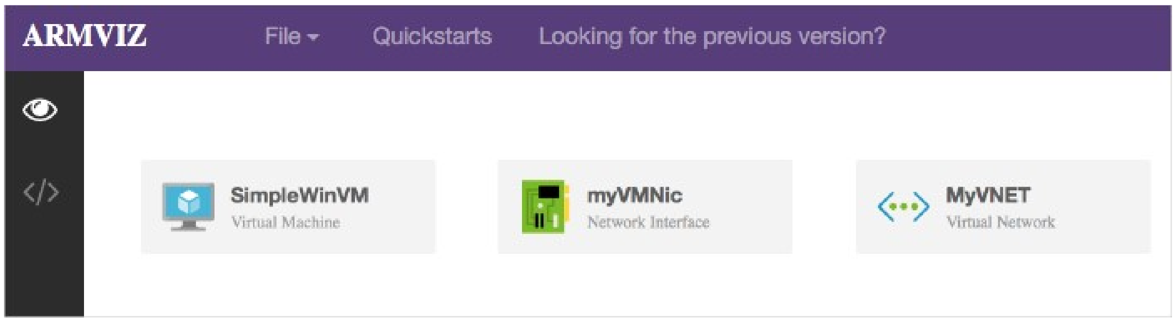
This template allows you to deploy a simple Windows VM using a few different optic
latest patched version. This will deploy a A2 size VM in the resource group location
name of the VM.

The **Deploy to Azure** button enables you to deploy the template directly through the Azure portal, just like you saw on the gallery page.

4. Click **Visualize** to navigate to the Azure Resource Manager Visualizer.

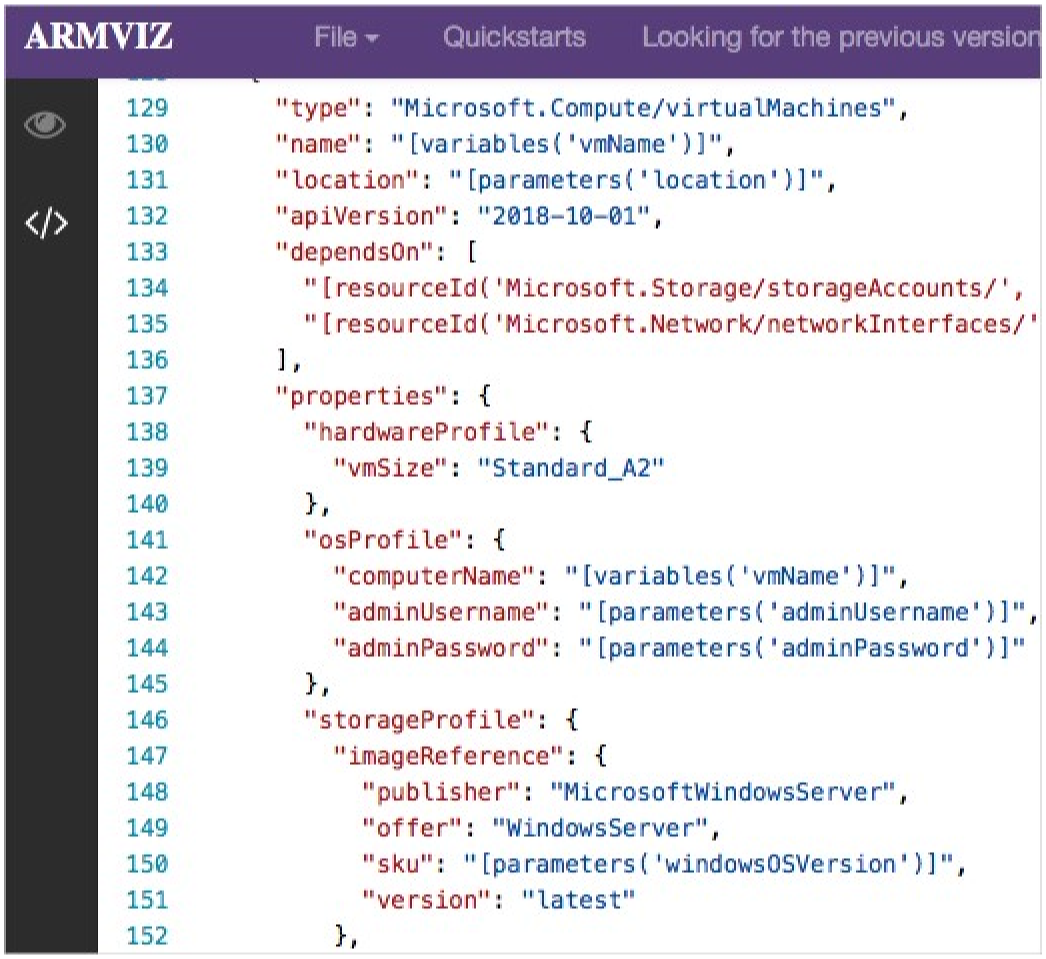
   You see the resources that make up the deployment, including a VM, a storage account, and network resources.

   You can use your mouse to arrange the resources. You can also use your mouse's scroll wheel to zoom in an out.

5. Click on the **Virtual Machine** resource labeled **SimpleWinVM**.

   You see the source code that defines the VM resource.



```
      "type": "Microsoft.Compute/virtualMachines",
      "name": "[variables('vmName')]",
      "location": "[parameters('location')]",
      "apiVersion": "2018-10-01",
      "dependsOn": [
        "[resourceId('Microsoft.Storage/storageAccounts/',
        "[resourceId('Microsoft.Network/networkInterfaces/'
      ],
      "properties": {
        "hardwareProfile": {
          "vmSize": "Standard_A2"
        },
        "osProfile": {
          "computerName": "[variables('vmName')]",
          "adminUsername": "[parameters('adminUsername')]",
          "adminPassword": "[parameters('adminPassword')]"
        },
        "storageProfile": {
          "imageReference": {
            "publisher": "MicrosoftWindowsServer",
            "offer": "WindowsServer",
            "sku": "[parameters('windowsOSVersion')]",
            "version": "latest"
          },
```

You'll have more time to inspect the source code in just a bit. But for now, take a moment to review it briefly.

You see that:

- The resource's type is `Microsoft.Compute/virtualMachines`.
- It's location, or Azure region, comes from the template parameter named `location`.
- The VM's size is **Standard_A2**.
- The computer name is read from a template variable and the username and password for the VM are read from template parameters.

In practice, you might review the **README.md** file on GitHub and further inspect the source code to see whether this template suits your needs.

But for now, this template looks promising. In the next part, you'll go ahead and deploy this template.

Next unit: Exercise - Deploy a VM using an Azure Quickstart template

Continue >

✓ 100 XP

Discover Azure Quickstart templates

5 minutes

Choose which platform you want to run in the cloud

Windows Linux

Recall that your analysts' financial models are run on Azure virtual machines. To further automate your deployments, you want to move from Azure CLI commands and scripts to Resource Manager templates.

Before you begin, you may wonder what existing templates are available to learn from and build upon.

Here you'll explore what an Azure Quickstart template is and what prebuilt templates are available for you to use right now.

Tip

Prefer Windows or want to try something new? Select **Windows** from the top of this page to deploy a Windows Server virtual machine.

What are Azure Quickstart templates?

Azure Quickstart templates are Resource Manager templates that are provided by the Azure community. Quickstart templates are available on GitHub.

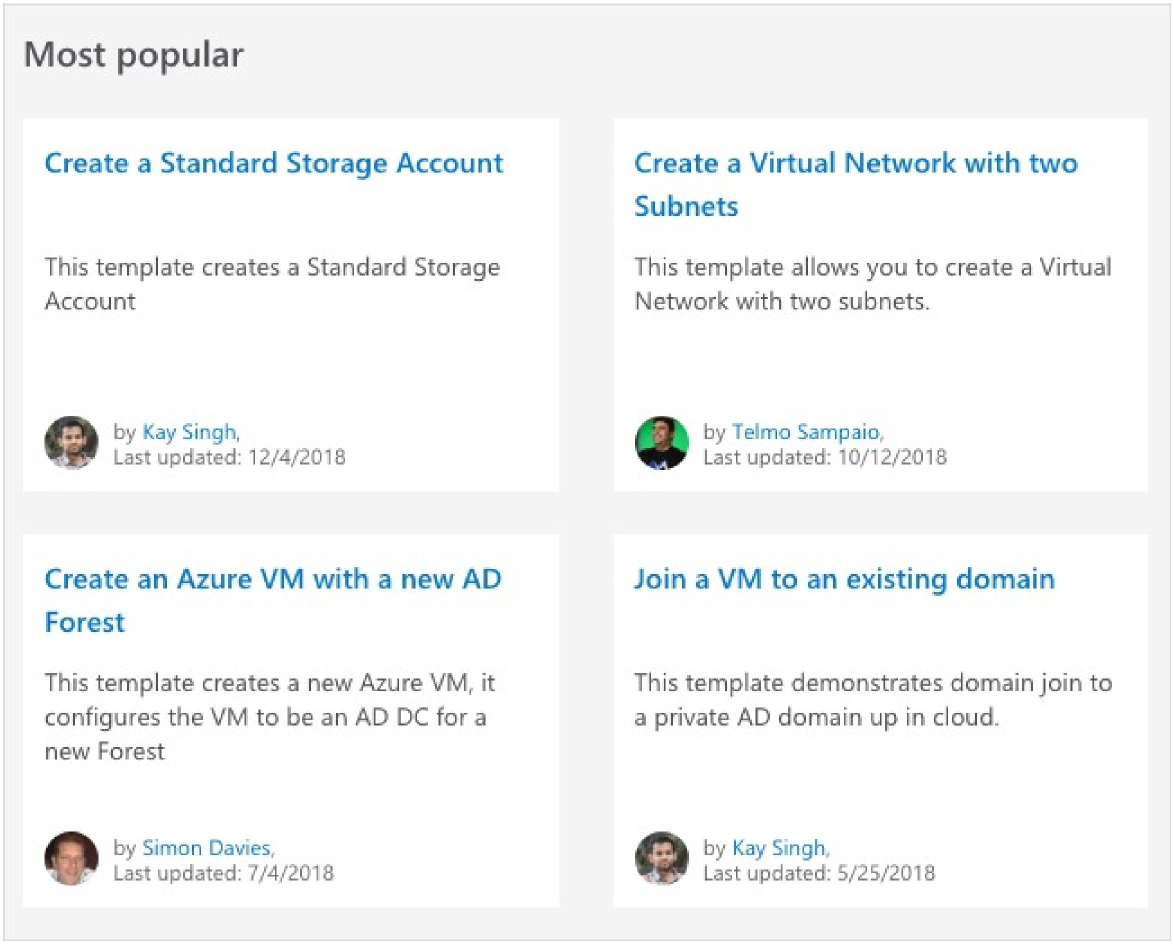
Many templates provide everything you need to deploy your solution. Others might serve as a starting point for your template. Either way, you can study these templates to learn how to best author and structure your own templates.

Discover what's on the Quickstart template gallery

Let's say you want to find a Resource Manager template that brings up a basic VM configuration – one that includes a VM, basic network settings, and storage.

1. Start by browsing to the [Quickstart template gallery](#) to see what's available.

   You see a number of popular and recently updated templates. These templates work with both Azure resources and popular software packages.

   Most popular

   Create a Standard Storage Account

   This template creates a Standard Storage Account

   by Kay Singh,
   Last updated: 12/4/2018

   Create a Virtual Network with two Subnets

   This template allows you to create a Virtual Network with two subnets.

   by Telmo Sampaio,
   Last updated: 10/12/2018

   Create an Azure VM with a new AD Forest

   This template creates a new Azure VM, it configures the VM to be an AD DC for a new Forest

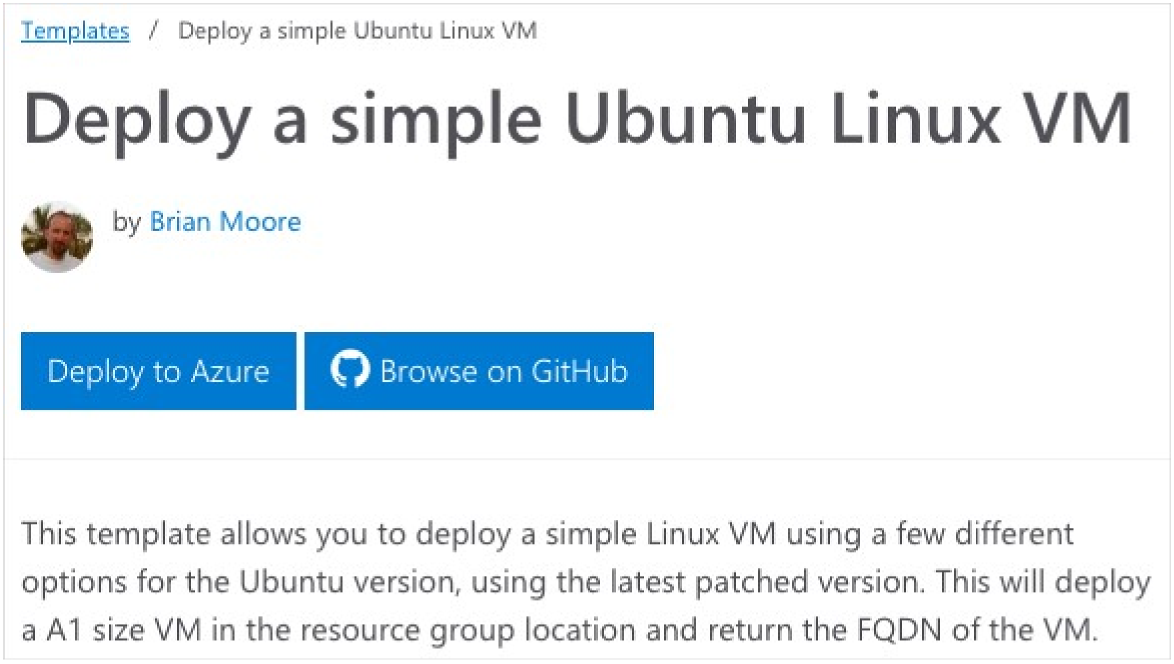
   by Simon Davies,
   Last updated: 7/4/2018

   Join a VM to an existing domain

   This template demonstrates domain join to a private AD domain up in cloud.

   by Kay Singh,
   Last updated: 5/25/2018

2. Let's say you come across the [Deploy a simple Ubuntu Linux VM](#) template.


Templates / Deploy a simple Ubuntu Linux VM

Deploy a simple Ubuntu Linux VM

by Brian Moore

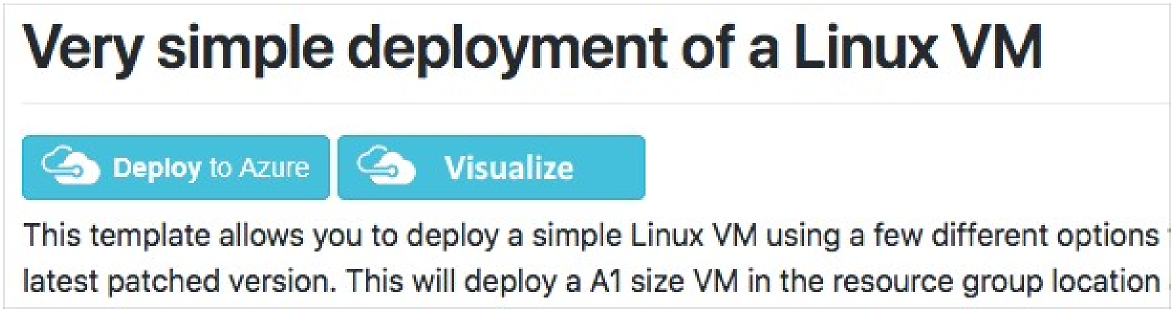
This template allows you to deploy a simple Linux VM using a few different options for the Ubuntu version, using the latest patched version. This will deploy a A1 size VM in the resource group location and return the FQDN of the VM.

The name sounds like exactly what you need. But let's take a closer look to see what this template accomplishes.

The **Deploy to Azure** button enables you to deploy the template directly through the Azure portal. But you won't do that here. Rather, you'll use the Azure CLI to deploy the template from Cloud Shell.

3. Click **Browse on GitHub** to navigate to the template's source code on GitHub.

   You see this.


Very simple deployment of a Linux VM

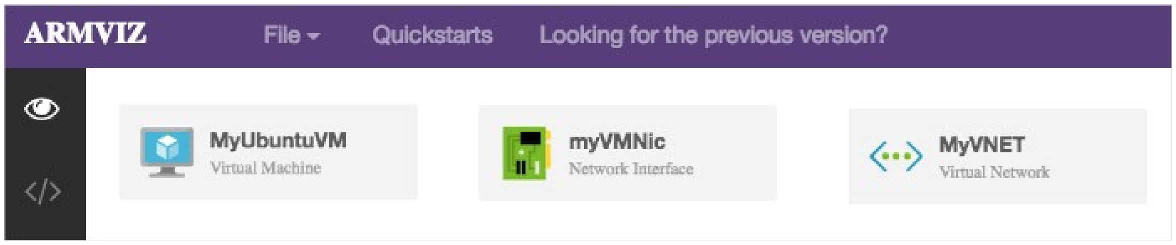
This template allows you to deploy a simple Linux VM using a few different options
latest patched version. This will deploy a A1 size VM in the resource group location

   The **Deploy to Azure** button enables you to deploy the template directly through the Azure portal, just like you saw on the gallery page.

4. Click **Visualize** to navigate to the Azure Resource Manager Visualizer.

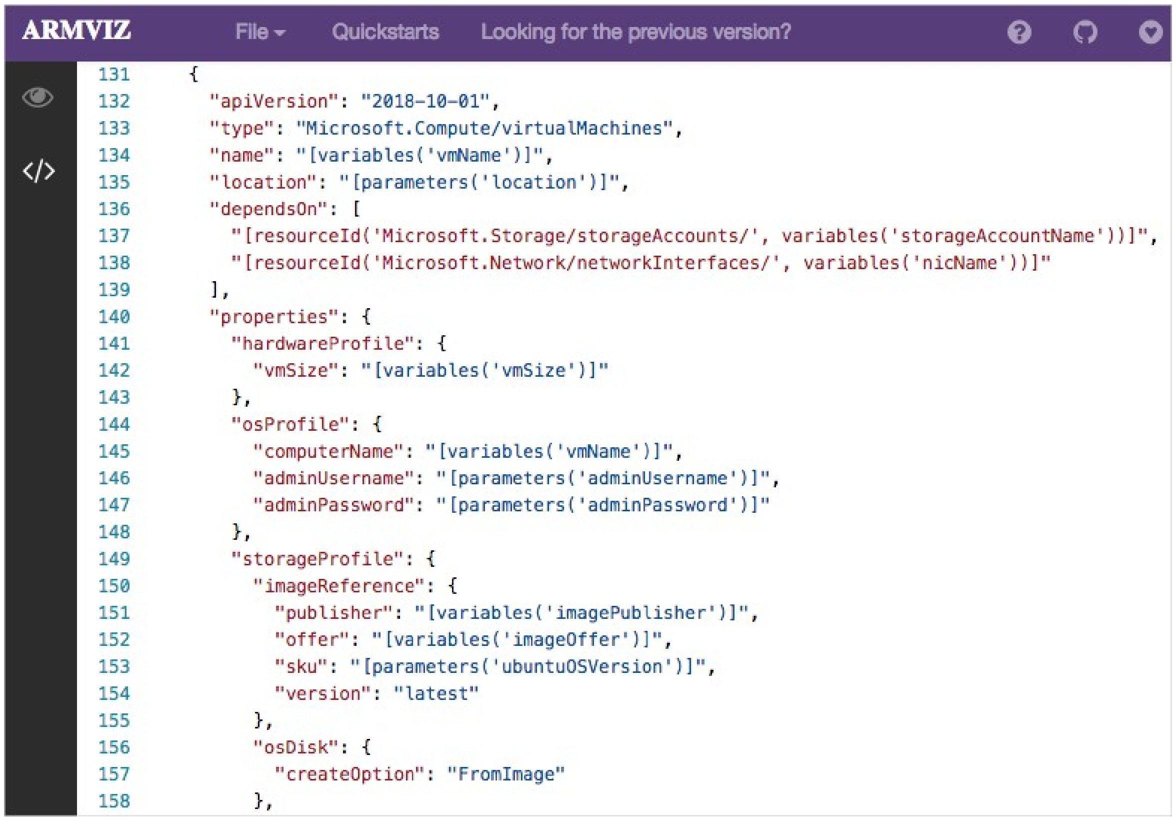
You see the resources that make up the deployment, including a VM, a storage account, and network resources.

You can use your mouse to arrange the resources. You can also use your mouse's scroll wheel to zoom in an out.



5. Click on the **Virtual Machine** resource labeled **MyUbuntuVM**.

   You see the source code that defines the VM resource.



```
    {
      "apiVersion": "2018-10-01",
      "type": "Microsoft.Compute/virtualMachines",
      "name": "[variables('vmName')]",
      "location": "[parameters('location')]",
      "dependsOn": [
        "[resourceId('Microsoft.Storage/storageAccounts/', variables('storageAccountName'))]",
        "[resourceId('Microsoft.Network/networkInterfaces/', variables('nicName'))]"
      ],
      "properties": {
        "hardwareProfile": {
          "vmSize": "[variables('vmSize')]"
        },
        "osProfile": {
          "computerName": "[variables('vmName')]",
          "adminUsername": "[parameters('adminUsername')]",
          "adminPassword": "[parameters('adminPassword')]"
        },
        "storageProfile": {
          "imageReference": {
            "publisher": "[variables('imagePublisher')]",
            "offer": "[variables('imageOffer')]",
            "sku": "[parameters('ubuntuOSVersion')]",
            "version": "latest"
          },
          "osDisk": {
            "createOption": "FromImage"
          },
```

You'll have more time to inspect the source code in just a bit. But for now, take a moment to review it briefly.

You see that:

- The resource's type is `Microsoft.Compute/virtualMachines`.
- It's location, or Azure region, comes from the template parameter named `location`.
- The VM's size comes from the template variable `vmSize`.
- The computer name is read from a template variable and the username and password for the VM are read from template parameters.

In practice, you might review the **README.md** file on GitHub and further inspect the source code to see whether this template suits your needs.

But for now, this template looks promising. In the next part, you'll go ahead and deploy this template.

Next unit: Exercise - Deploy a VM using an Azure Quickstart template

Continue >

✓ 100 XP

Exercise - Deploy a VM using an Azure Quickstart template

10 minutes

Choose which platform you want to run in the cloud

Windows Linux

Here you'll deploy the Quickstart template you explored in the previous part. The Quickstart template brings up a basic virtual machine configuration.

Important

You need your own Azure subscription to run this exercise and you may incur charges. If you don't already have an Azure subscription, create a free account before you begin.

Before you do that, let's briefly review the deployment process.

How do I deploy a Resource Manager template?

You can use automation scripting tools such as the Azure CLI, Azure PowerShell, or even the Azure REST APIs with your favorite programming language to deploy resources from templates. You can also deploy your templates through Visual Studio, Visual Studio Code, and the Azure portal. Shortly, you'll deploy a Resource Manager template using the Azure CLI from Cloud Shell.

Verifying a template

Before you run your template, you might want to verify it first.

A good first step is to process your template with a *linter*, a tool that verifies that the JSON syntax of your template is correct. You can find JSON linting tools that run on the command line, in a browser, or in your favorite code editor.

Tip

Visual Studio Code provides many **built-in features** that make it easier to work with JSON files. Your favorite editor may provide similar built-in features or plugins.

The next step might be to visualize your template. Azure Resource Manager Visualizer enables you to upload your Resource Manager template and graphically see how your resources relate to one another. You can inspect this visualization to verify that the deployment is set up correctly and meets your requirements.

Finally, you can perform a test deployment from the Azure CLI or Azure PowerShell. A test deployment doesn't create any resources, but it provides you with feedback on what would happen when the deployment runs. You'll perform a test deployment shortly to see the process in action.

Create a resource group

First, we'll create a *resource group* to hold all the things that we need to create. This allows us to administer all the VMs, disks, network interfaces, and other elements that make up our solution as a unit. We can use the Azure CLI to create a resource group with the `az group create` command. It takes a `--name` to give it a unique name in our subscription, and a `--location` to tell Azure what area of the world we want the resources to be located by default.

1. Sign in to the Azure portal.

2. From the menu bar on the top right-hand side, open **Cloud Shell**.

3. Set the resource group name.

   bash

   ```bash
   RESOURCEGROUP=learn-quickstart-vm-rg
   ```

4. Set the location. Replace the eastus value with a location near you.

```bash
LOCATION=eastus
```

The following list has some location values you can use.

- westus2
- southcentralus
- centralus
- eastus
- westeurope
- southeastasia
- japaneast
- brazilsouth
- australiasoutheast
- centralindia

5. Run the following command to create a resource group.

```bash
az group create --name $RESOURCEGROUP --location $LOCATION
```

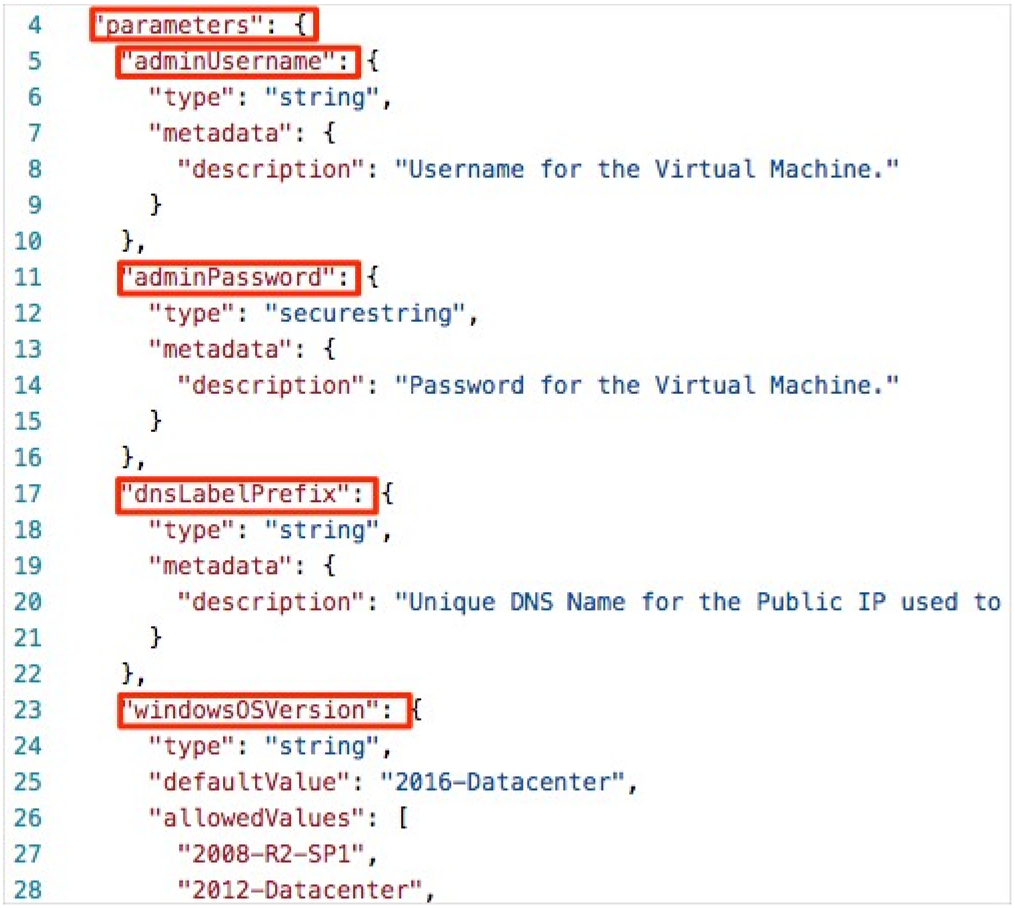
Create template parameters

Recall that a Resource Manager template is divided into sections, one of them being **Parameters**.

Near the start of the template, you see a section named `parameters`. This section defines these five parameters:

- `adminUsername`
- `adminPassword`
- `dnsLabelPrefix`

- `windowsOSVersion`
- `location`



```
"parameters": {
  "adminUsername": {
    "type": "string",
    "metadata": {
      "description": "Username for the Virtual Machine."
    }
  },
  "adminPassword": {
    "type": "securestring",
    "metadata": {
      "description": "Password for the Virtual Machine."
    }
  },
  "dnsLabelPrefix": {
    "type": "string",
    "metadata": {
      "description": "Unique DNS Name for the Public IP used to
    }
  },
  "windowsOSVersion": {
    "type": "string",
    "defaultValue": "2016-Datacenter",
    "allowedValues": [
      "2008-R2-SP1",
      "2012-Datacenter",
```

Two of these parameters – `windowsOSVersion` and `location` – have default values. The default value for `windowsOSVersion` is "2016-Datacenter" and the default value for `location` is the parent resource group's location.

Let's keep these parameters at their default values. For the remaining three parameters, you have two options:

1. Provide the values in a JSON file.
2. Provide the values as command-line arguments.

For learning purposes, here you'll provide the values as command-line arguments. To make the template easy to deploy, you'll start by storing these values as Bash variables.

1. Sign in to the Azure portal.

2. From the menu bar on the top right-hand side, open **Cloud Shell**.

3. From Cloud Shell, create a username. For this example, let's use **azureuser**.

```bash
USERNAME=azureuser
```

4. Run the **openssl** utility to generate a random password.

```bash
PASSWORD=$(openssl rand -base64 32)
```

   There are many ways generate random passwords. The method you choose depends on your workflow and requirements. This method uses the **openssl** utility to generate 32 random bytes and base64 encode the output. Base64 encoding ensures that the result contains only printable characters.

5. Generate a unique DNS label prefix.

```bash
DNS_LABEL_PREFIX=mydeployment-$RANDOM
```

   The DNS label prefix must be unique. The DNS label prefix begins with "mydeployment" followed by a random number. `$RANDOM` is a Bash function that generates a random positive whole number.

   In practice, you would choose a DNS label prefix that fits your requirements.

Validate and launch the template

With your parameters in place, you have everything you need to launch the template.

As a final verification step, you'll begin by validating that the template is syntactically correct.

1. From Cloud Shell, run `az group deployment validate` to validate the template.

Azure CLI

```
az group deployment validate \
  --resource-group $RESOURCEGROUP \
  --template-uri "https://raw.githubusercontent.com/Azure/azure-quick-
start-templates/master/101-vm-simple-windows/azuredeploy.json" \
  --parameters adminUsername=$USERNAME \
  --parameters adminPassword=$PASSWORD \
  --parameters dnsLabelPrefix=$DNS_LABEL_PREFIX
```

The `--template-uri` argument points to the template on GitHub. The template's filename is **azuredeploy.json**. Later, you'll see how to validate and run a template on your local filesystem.

You see a large JSON block as output, which tells you that the template passed validation.

Azure Resource Manager fills in the template parameters and checks whether the template would successfully run in your subscription.

If validation failed, you would see a detailed description of the failure in the output.

2. Run `az group deployment create` to deploy the template.

Azure CLI

```
az group deployment create \
  --name MyDeployment \
  --resource-group $RESOURCEGROUP \
  --template-uri "https://raw.githubusercontent.com/Azure/azure-quick-
start-templates/master/101-vm-simple-windows/azuredeploy.json" \
  --parameters adminUsername=$USERNAME \
  --parameters adminPassword=$PASSWORD \
  --parameters dnsLabelPrefix=$DNS_LABEL_PREFIX
```

This command resembles the previous command, but also includes the `--name` argument to give your deployment a name.

This command takes 4-5 minutes to complete. While you wait, now's a great time to take a closer look at the source code for this template. Remember, the contents of a Resource Manager template will become more familiar to you as you read existing templates and create your own.

When the deployment completes, you see another large JSON block as output that describes the deployment.

Verify the deployment

The deployment succeeded. But let's run a few commands just to verify.

1. Run `az group deployment show` to verify the deployment.

   Azure CLI

   ```
   az group deployment show \
     --name MyDeployment \
     --resource-group $RESOURCEGROUP
   ```

   You see the same JSON block as you did previously. You can run this command later if you ever need these details about the deployment. The output is structured as JSON to make it easier to feed into other tools you might use to track your deployments and cloud usage.

2. Run `az vm list` to see which VMs are running.

   Azure CLI

   ```
   az vm list \
     --resource-group $RESOURCEGROUP \
     --output table
   ```

   Your output resembles this. Your region is shown under the **Location** column.

   bash

   ```
   Name         ResourceGroup                        Location
   Zones
   -----------  -----------------------------------  --------------  ----
   ---
   SimpleWinVM  learn-quickstart-vm-rg  southcentralus
   ```

Recall that the template names the VM "SimpleWinVM". Here you see that this VM exists in your resource group.

Next unit: Add a resource to the template

Continue

✓ 100 XP

Exercise - Deploy a VM using an Azure Quickstart template

10 minutes

Choose which platform you want to run in the cloud

Windows Linux

Here you'll deploy the Quickstart template you explored in the previous part. The Quickstart template brings up a basic virtual machine configuration.

Important

You need your own Azure subscription to run this exercise and you may incur charges. If you don't already have an Azure subscription, create a free account before you begin.

Before you do that, let's briefly review the deployment process.

How do I deploy a Resource Manager template?

You can use automation scripting tools such as the Azure CLI, Azure PowerShell, or even the Azure REST APIs with your favorite programming language to deploy resources from templates. You can also deploy your templates through Visual Studio, Visual Studio Code, and the Azure portal. Shortly, you'll deploy a Resource Manager template using the Azure CLI from Cloud Shell.

Verifying a template

Before you run your template, you might want to verify it first.

A good first step is to process your template with a *linter*, a tool that verifies that the JSON syntax of your template is correct. You can find JSON linting tools that run on the command line, in a browser, or in your favorite code editor.

> **Tip**
>
> Visual Studio Code provides many **built-in features** that make it easier to work with JSON files. Your favorite editor may provide similar built-in features or plugins.

The next step might be to visualize your template. Azure Resource Manager Visualizer enables you to upload your Resource Manager template and graphically see how your resources relate to one another. You can inspect this visualization to verify that the deployment is set up correctly and meets your requirements.

Finally, you can perform a test deployment from the Azure CLI or Azure PowerShell. A test deployment doesn't create any resources, but it provides you with feedback on what would happen when the deployment runs. You'll perform a test deployment shortly to see the process in action.

Create a resource group

First, we'll create a *resource group* to hold all the things that we need to create. This allows us to administer all the VMs, disks, network interfaces, and other elements that make up our solution as a unit. We can use the Azure CLI to create a resource group with the `az group create` command. It takes a `--name` to give it a unique name in our subscription, and a `--location` to tell Azure what area of the world we want the resources to be located by default.

1. Sign in to the Azure portal.

2. From the menu bar on the top right-hand side, open **Cloud Shell**.

3. Set the resource group name.

bash Copy

```bash
RESOURCEGROUP=learn-quickstart-vm-rg
```

4. Set the location. Replace the eastus value with a location near you.

```bash
LOCATION=eastus
```

The following list has some location values you can use.

- westus2
- southcentralus
- centralus
- eastus
- westeurope
- southeastasia
- japaneast
- brazilsouth
- australiasoutheast
- centralindia

5. Run the following command to create a resource group.

```bash
az group create --name $RESOURCEGROUP --location $LOCATION
```

Create template parameters

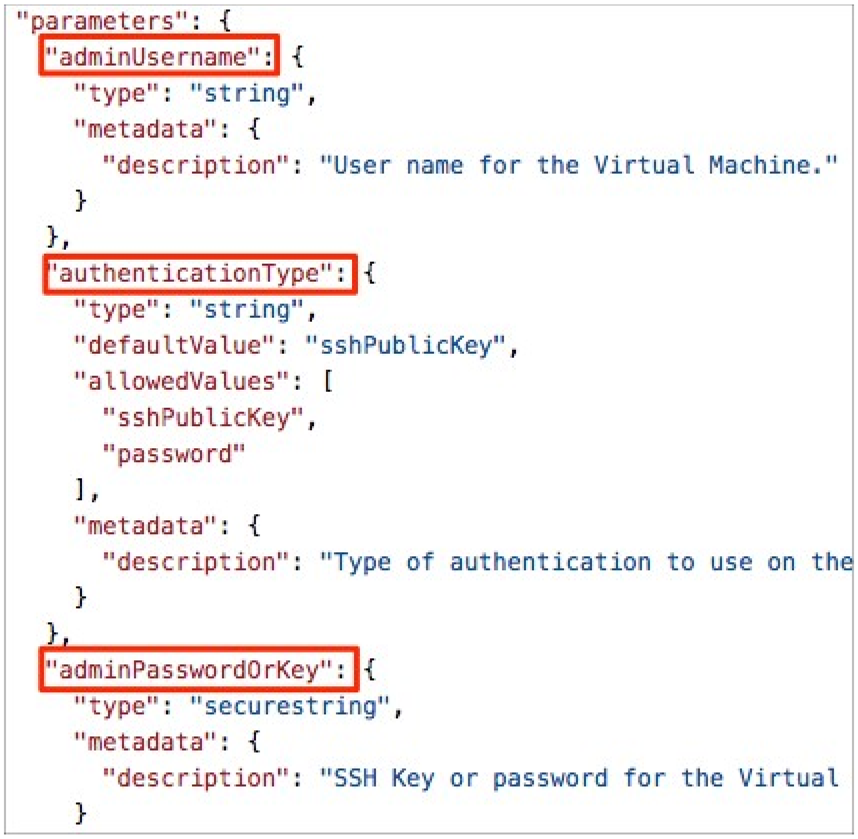
Recall that a Resource Manager template is divided into sections, one of them being **Parameters**.

Near the start of the template, you see a section named `parameters`. This section defines these parameters:

- `adminUsername`
- `authenticationType`
- `adminPasswordOrKey`

- `dnsLabelPrefix`
- `ubuntuOSVersion`
- `location`

This image highlights the first few parameters.



```
"parameters": {
  "adminUsername": {
    "type": "string",
    "metadata": {
      "description": "User name for the Virtual Machine."
    }
  },
  "authenticationType": {
    "type": "string",
    "defaultValue": "sshPublicKey",
    "allowedValues": [
      "sshPublicKey",
      "password"
    ],
    "metadata": {
      "description": "Type of authentication to use on the
    }
  },
  "adminPasswordOrKey": {
    "type": "securestring",
    "metadata": {
      "description": "SSH Key or password for the Virtual
    }
```

The `authenticationType` parameter specifies whether to use password authentication or key-based authentication to connect to the virtual machine. The `adminPasswordOrKey` parameter specifies the password or SSH key. Although key-based authentication is typically more secure than password authentication, here you'll use password authentication for learning purposes.

Some parameters, such as `ubuntuOSVersion` and `location`, have default values. The default value for `ubuntuOSVersion` is "18.04-LTS" and the default value for `location` is the parent resource group's location.

Let's keep these parameters at their default values. For the remaining parameters, you have two options:

1. Provide the values in a JSON file.

2. Provide the values as command-line arguments.

For learning purposes, here you'll provide the values as command-line arguments. To make the template easy to deploy, you'll start by storing these values as Bash variables.

1. Sign in to the Azure portal.
2. From the menu bar on the top right-hand side, open **Cloud Shell**.
3. From Cloud Shell, create a username. For this example, let's use **azureuser**.

   bash Copy

   ```bash
   USERNAME=azureuser
   ```

4. Run the **openssl** utility to generate a random password.

   bash Copy

   ```bash
   PASSWORD=$(openssl rand -base64 32)
   ```

   There are many ways generate random passwords. The method you choose depends on your workflow and requirements. This method uses the **openssl** utility to generate 32 random bytes and base64 encode the output. Base64 encoding ensures that the result contains only printable characters.

5. Generate a unique DNS label prefix.

   bash Copy

   ```bash
   DNS_LABEL_PREFIX=mydeployment-$RANDOM
   ```

   The DNS label prefix must be unique. The DNS label prefix begins with "mydeployment" followed by a random number. `$RANDOM` is a Bash function that generates a random positive whole number.

   In practice, you would choose a DNS label prefix that fits your requirements.

Validate and launch the template

With your parameters in place, you have everything you need to launch the template.

As a final verification step, you'll begin by validating that the template is syntactically correct.

1. From Cloud Shell, run `az group deployment validate` to validate the template.

   Azure CLI

   ```
   az group deployment validate \
     --resource-group $RESOURCEGROUP \
     --template-uri "https://raw.githubusercontent.com/Azure/azure-quick-start-templates/master/101-vm-simple-linux/azuredeploy.json" \
     --parameters adminUsername=$USERNAME \
     --parameters authenticationType=password \
     --parameters adminPasswordOrKey=$PASSWORD \
     --parameters dnsLabelPrefix=$DNS_LABEL_PREFIX
   ```

   The `--template-uri` argument points to the template on GitHub. The template's filename is **azuredeploy.json**. Later, you'll see how to validate and run a template on your local filesystem.

   You see a large JSON block as output, which tells you that the template passed validation.

   Azure Resource Manager fills in the template parameters and checks whether the template would successfully run in your subscription.

   If validation failed, you would see a detailed description of the failure in the output.

2. Run `az group deployment create` to deploy the template.

   Azure CLI

   ```
   az group deployment create \
     --name MyDeployment \
     --resource-group $RESOURCEGROUP \
     --template-uri "https://raw.githubusercontent.com/Azure/azure-quick-
   ```

```
start-templates/master/101-vm-simple-linux/azuredeploy.json" \
  --parameters adminUsername=$USERNAME \
  --parameters authenticationType=password \
  --parameters adminPasswordOrKey=$PASSWORD \
  --parameters dnsLabelPrefix=$DNS_LABEL_PREFIX
```

This command resembles the previous command, but also includes the `--name` argument to give your deployment a name.

This command takes 2-3 minutes to complete. While you wait, now's a great time to take a closer look at the source code for this template. Remember, the contents of a Resource Manager template will become more familiar to you as you read existing templates and create your own.

When the deployment completes, you see another large JSON block as output that describes the deployment.

Verify the deployment

The deployment succeeded. But let's run a few commands just to verify.

1. Run `az group deployment show` to verify the deployment.

   Azure CLI

   ```
   az group deployment show \
     --name MyDeployment \
     --resource-group $RESOURCEGROUP
   ```

   You see the same JSON block as you did previously. You can run this command later if you ever need these details about the deployment. The output is structured as JSON to make it easier to feed into other tools you might use to track your deployments and cloud usage.

2. Run `az vm list` to see which VMs are running.

   Azure CLI

```
az vm list \
  --resource-group $RESOURCEGROUP \
  --output table
```

Your output resembles this. Your region is shown under the **Location** column.

bash

Copy

```
Name        ResourceGroup                         Location        Zones
----------  ------------------------------------  --------------  -----
--
simpleLinuxVM  learn-quickstart-vm-rg    southcentralus
```

Recall that the template names the VM "MyUbuntuVM". Here you see that this VM exists in your resource group.

Next unit: Add a resource to the template

Continue >

200 XP

Add a resource to the template

5 minutes

Choose which platform you want to run in the cloud

Windows Linux

When you create a VM for your financial analysts, you typically include a web server that provides a dashboard for the analyst to visualize and collect the results of the run. Using a Resource Manager template to bring up a VM is a good start. But how can you extend the template to configure web server software inside the VM?

Using the Azure CLI to configure a VM to run a web server requires only a few basic commands. But what happens when:

- You need to create and delete VMs continuously?
- Your deployments require additional components such as storage and networking, increasing their complexity?

Managing individual resources through the Azure CLI is a good start. But it quickly becomes tedious and error prone because you still need to correctly connect resources together. You need a more automated solution.

Here you'll examine the requirements for your VM's web server and learn how to build resources you can add to your templates.

Note

The code and commands you see on this page are for illustration. For now, just follow along. You don't need to modify any files or run any commands just yet.

What's the Custom Script Extension?

The Custom Script Extension is an easy way to download and run scripts on your Azure VMs. It's just one of the many ways you can configure a VM once it's up and running.

You can store your scripts in Azure storage or in a public location such as GitHub. You can run scripts manually or as part of a more automated deployment. Here, you'll define a resource that you'll soon add to your Resource Manager template. The resource will use the Custom Script Extension to download a PowerShell script from GitHub and execute that script on your VM. The script enables the IIS-WebServerRole feature and configures a basic home page.

How do I extend a Resource Manager template?

Your Resource Manager template creates a Windows VM. There are a few ways to extend the template to also enable IIS when the VM starts.

One way to extend your template is to create multiple templates, each defining one piece of the system. You then *link* or *nest* them together to build a more complete system. As you create your own templates, you can build a library of smaller, more granular templates and combine them how you need.

Another way is to modify an existing template to suit your needs. You'll do that in this module because that's often the fastest way to get started writing your own templates.

Build the template resource

Here you'll learn how to build template resources, using the Custom Script Extension as an example. You'll use the reference documentation as a starting point and then define the resource properties you need.

Let's start by reviewing your requirements.

Gather requirements

Here's a brief summary of what you want to accomplish through your template.

1. Create a VM.
2. Open port 80 through the network firewall.
3. Install and configure web server software on your VM.

The Resource Manager template you used in the previous part already covers the first two requirements. For the third, let's start by taking a look at an Azure CLI command we could run manually from the command line to enable IIS on your VM using the Custom Script Extension.

Azure CLI

```
az vm extension set \
  --resource-group $RESOURCEGROUP \
  --vm-name SimpleWinVM \
  --name CustomScriptExtension \
  --publisher Microsoft.Compute \
  --version 1.9 \
  --settings '{"fileUris":["https://raw.githubusercontent.com/Mi-
crosoftDocs/mslearn-welcome-to-azure/master/configure-iis.ps1"]}' \
  --protected-settings '{"commandToExecute": "powershell -ExecutionPolicy
Unrestricted -File configure-iis.ps1"}'
```

This command uses the Custom Script Extension to run a PowerShell script on your VM that installs IIS. You can examine the PowerShell script from a separate browser tab if you'd like.

You need a way to map this command to use the Resource Manager template syntax. The `az vm extension set` example shown above is a command you'd enter on the command line. What you need is the declarative JSON in template format.

Locate the resource schema

To discover how to use the Custom Script Extension in your template, one approach is to learn by example. For example, you might find an Azure Quickstart template that does something similar and adapt it to your needs.

Another approach is to look up the resource you need in the reference documentation. In this case, searching the documentation would reveal Microsoft.Compute virtualMachines/extensions template reference.

You can start by copying the schema to a temporary document, like this.

JSON

```json
{
  "name": "string",
  "type": "Microsoft.Compute/virtualMachines/extensions",
  "apiVersion": "2018-10-01",
  "location": "string",
  "tags": {},
  "properties": {
    "publisher": "string",
    "type": "string",
    "typeHandlerVersion": "string",
    "autoUpgradeMinorVersion": boolean,
    "settings": {},
    "protectedSettings": {},
    "instanceView": {
      "name": "string",
      "type": "string",
      "typeHandlerVersion": "string",
      "substatuses": [
        {
          "code": "string",
          "level": "string",
          "displayStatus": "string",
          "message": "string",
          "time": "string"
        }
      ],
      "statuses": [
        {
          "code": "string",
          "level": "string",
          "displayStatus": "string",
          "message": "string",
          "time": "string"
        }
      ]
    }
  }
}
```

Specify required properties

The schema shows all of the properties you can provide. Some properties are required; others are optional. You might start by identifying all of the required properties. Locate these below the schema definition on the reference page.

Here are the required parameters.

- `name`
- `type`
- `apiVersion`
- `location`
- `properties`

After you remove all the parameters that aren't required, your resource definition might look like this.

JSON

```json
{
  "name": "[concat(variables('vmName'), '/', 'ConfigureIIS')]",
  "type": "Microsoft.Compute/virtualMachines/extensions",
  "apiVersion": "2018-06-01",
  "location": "[parameters('location')]",
  "properties": { }
}
```

The values for the `type` and `apiVersion` properties come directly from the documentation. `properties` is required but for now can be empty.

You know that your existing VM template has a parameter named `location`. This example uses the built-in `parameters` function to read that value.

The `name` property follows a special convention. This example uses the built-in `concat` function to concatenate, or combine, multiple strings. The complete string contains the VM name followed by a slash `/` character followed by a name you choose (here, it's "ConfigureIIS"). The VM name helps the template identify which VM resource to run the script on.

Specify additional properties

Next, you might add in any additional properties that you need. Referring back to the CLI example earlier, you need the location, or URI, of the script file and the PowerShell command to execute on the VM to run that script. As a recommended practice, you might also include the resource's publisher name, its type, and version.

Referring back to the documentation, you need these values, shown here using "dot" notation.

- `properties.publisher`
- `properties.type`
- `properties.typeHandlerVersion`
- `properties.autoUpgradeMinorVersion`
- `properties.settings.fileUris`
- `properties.protectedSettings.commandToExecute`

Your Custom Script Extension resource now looks like this.

JSON

Copy

```json
{
  "name": "[concat(variables('vmName'), '/', 'ConfigureIIS')]",
  "type": "Microsoft.Compute/virtualMachines/extensions",
  "apiVersion": "2018-06-01",
  "location": "[parameters('location')]",
  "properties": {
    "publisher": "Microsoft.Compute",
    "type": "CustomScriptExtension",
    "typeHandlerVersion": "1.9",
    "autoUpgradeMinorVersion": true,
    "settings": {
      "fileUris": [
        "https://raw.githubusercontent.com/MicrosoftDocs/mslearn-welcome-to-
azure/master/configure-iis.ps1"
      ]
    },
    "protectedSettings": {
      "commandToExecute": "powershell -ExecutionPolicy Unrestricted -File
configure-iis.ps1"
    }
  }
}
```

Specify dependent resources

You can't run the Custom Script Extension until the VM is available. All template resources provide a `dependsOn` property. This property helps Resource Manager determine the correct order to apply resources.

Here's what your template resource might look like after you add the `dependsOn` property.

JSON

```json
{
  "name": "[concat(variables('vmName'), '/', 'ConfigureIIS')]",
  "type": "Microsoft.Compute/virtualMachines/extensions",
  "apiVersion": "2018-06-01",
  "location": "[parameters('location')]",
  "properties": {
    "publisher": "Microsoft.Compute",
    "type": "CustomScriptExtension",
    "typeHandlerVersion": "1.9",
    "autoUpgradeMinorVersion": true,
    "settings": {
      "fileUris": [
        "https://raw.githubusercontent.com/MicrosoftDocs/mslearn-welcome-to-
azure/master/configure-iis.ps1"
      ]
    },
    "protectedSettings": {
      "commandToExecute": "powershell -ExecutionPolicy Unrestricted -File
configure-iis.ps1"
    }
  },
  "dependsOn": [
    "[resourceId('Microsoft.Compute/virtualMachines/', varia-
bles('vmName'))]"
  ]
}
```

The bracket `[ ]` syntax means that you can provide an array, or list, of resources that must exist before applying this resource.

There are multiple ways to define a resource dependency. You can provide its name, such as "SimpleWinVM", it's full name (including its namespace, type, and name), such as "Microsoft.Compute/virtualMachines/SimpleWinVM", or by its resource ID.

This example uses the built-in `resourceId` function to get the VM's resource ID using its full name. This helps clarify which resource you're referring to and can help avoid ambiguity when more than one resource has a similar name.

The existing template provides a `vmName` variable that defines the VM's name. This example uses the built-in `variables` function to read it.

Summary

You now have everything you need to define the Custom Script Extension resource in your template. In the next part, you'll extend your template to use it.

Check your knowledge

1. How can you help Resource Manager determine the correct order to apply resources?

- ○ Use the `dependsOn` element to define when one resource must exist before another can be deployed.
- ○ In your template, define resources in the order they need to be deployed.
- ○ Set the `order` property on each resource.

Check your answers

200 XP

Add a resource to the template

5 minutes

Choose which platform you want to run in the cloud

Windows Linux

When you create a VM for your financial analysts, you typically include a web server that provides a dashboard for the analyst to visualize and collect the results of the run. Using a Resource Manager template to bring up a VM is a good start. But how can you extend the template to configure web server software inside the VM?

Using the Azure CLI to configure a VM to run a web server requires only a few basic commands. But what happens when:

- You need to create and delete VMs continuously?
- Your deployments require additional components such as storage and networking, increasing their complexity?

Managing individual resources through the Azure CLI is a good start. But it quickly becomes tedious and error prone because you still need to correctly connect resources together. You need a more automated solution.

Here you'll examine the requirements for your VM's web server and learn how to build resources you can add to your templates.

Note

The code and commands you see on this page are for illustration. For now, just follow along. You don't need to modify any files or run any commands just yet.

What's the Custom Script Extension?

The Custom Script Extension is an easy way to download and run scripts on your Azure VMs. It's just one of the many ways you can configure a VM once it's up and running.

You can store your scripts in Azure storage or in a public location such as GitHub. You can run scripts manually or as part of a more automated deployment. Here, you'll define a resource that you'll soon add to your Resource Manager template. The resource will use the Custom Script Extension to download a Bash script from GitHub and execute that script on your VM. The script installs and configures the Nginx web server software package and configures a basic home page.

How do I extend a Resource Manager template?

Your Resource Manager template creates an Ubuntu Linux VM. There are a few ways to extend the template to also install Nginx when the VM starts.

One way to extend your template is to create multiple templates, each defining one piece of the system. You then *link* or *nest* them together to build a more complete system. As you create your own templates, you can build a library of smaller, more granular templates and combine them how you need.

Another way is to modify an existing template to suit your needs. You'll do that in this module because that's often the fastest way to get started writing your own templates.

Build the template resource

Here you'll learn how to build template resources, using the Custom Script Extension as an example. You'll use the reference documentation as a starting point and then define the resource properties you need.

Let's start by reviewing your requirements.

Gather requirements

Here's a brief summary of what you want to accomplish through your template.

1. Create a VM.
2. Open port 80 through the network firewall.
3. Install and configure web server software on your VM.

The Resource Manager template you used in the previous part already covers the first two requirements. For the third, let's start by taking a look at an Azure CLI command we could run manually from the command line to install and configure Nginx on your VM using the Custom Script Extension.

Azure CLI

```
az vm extension set \
  --resource-group $RESOURCEGROUP \
  --vm-name simpleLinuxVM \
  --name customScript \
  --publisher Microsoft.Azure.Extensions \
  --version 2.0 \
  --settings '{"fileUris":["https://raw.githubusercontent.com/Mi-
crosoftDocs/mslearn-welcome-to-azure/master/configure-nginx.sh"]}' \
  --protected-settings '{"commandToExecute": "./configure-nginx.sh"}'
```

This command uses the Custom Script Extension to run a Bash script on your VM that installs Nginx. You can examine the Bash script from a separate browser tab if you'd like.

You need a way to map this command to use the Resource Manager template syntax. The `az vm extension set` example shown above is a command you'd enter on the command line. What you need is the declarative JSON in template format.

Locate the resource schema

To discover how to use the Custom Script Extension in your template, one approach is to learn by example. For example, you might find an Azure Quickstart template that does something similar and adapt it to your needs.

Another approach is to look up the resource you need in the reference documentation. In this case, searching the documentation would reveal Microsoft.Compute virtualMachines/extensions template reference.

You can start by copying the schema to a temporary document, like this.

JSON

Copy

```json
{
  "name": "string",
  "type": "Microsoft.Compute/virtualMachines/extensions",
  "apiVersion": "2018-10-01",
  "location": "string",
  "tags": {},
  "properties": {
    "publisher": "string",
    "type": "string",
    "typeHandlerVersion": "string",
    "autoUpgradeMinorVersion": boolean,
    "settings": {},
    "protectedSettings": {},
    "instanceView": {
      "name": "string",
      "type": "string",
      "typeHandlerVersion": "string",
      "substatuses": [
        {
          "code": "string",
          "level": "string",
          "displayStatus": "string",
          "message": "string",
          "time": "string"
        }
      ],
      "statuses": [
        {
          "code": "string",
          "level": "string",
          "displayStatus": "string",
          "message": "string",
          "time": "string"
        }
      ]
    }
  }
}
```

Specify required properties

The schema shows all of the properties you can provide. Some properties are required; others are optional. You might start by identifying all of the required properties. Locate these below the schema definition on the reference page.

Here are the required parameters.

- `name`
- `type`
- `apiVersion`
- `location`
- `properties`

After you remove all the parameters that aren't required, your resource definition might look like this.

JSON

```json
{
  "name": "[concat(parameters('vmName'), '/', 'ConfigureNginx')]",
  "type": "Microsoft.Azure.Extensions/virtualMachines/extensions",
  "apiVersion": "2018-06-01",
  "location": "[parameters('location')]",
  "properties": { }
}
```

The values for the `type` and `apiVersion` properties come directly from the documentation. `properties` is required but for now can be empty.

You know that your existing VM template has a parameter named `location`. This example uses the built-in `parameters` function to read that value.

The `name` property follows a special convention. This example uses the built-in `concat` function to concatenate, or combine, multiple strings. The complete string contains the VM name followed by a slash `/` character followed by a name you choose (here, it's "ConfigureNginx"). The VM name helps the template identify which VM resource to run the script on.

Specify additional properties

Next, you might add in any additional properties that you need. Referring back to the CLI example earlier, you need the location, or URI, of the script file and the Bash command to execute on the VM to run that script. As a recommended practice, you might also include the resource's publisher name, its type, and version.

Referring back to the documentation, you need these values, shown here using "dot" notation.

- `properties.publisher`
- `properties.type`
- `properties.typeHandlerVersion`
- `properties.autoUpgradeMinorVersion`
- `properties.settings.fileUris`
- `properties.protectedSettings.commandToExecute`

Your Custom Script Extension resource now looks like this.

JSON

Copy

```json
{
  "name": "[concat(parameters('vmName'), '/', 'ConfigureNginx')]",
  "type": "Microsoft.Compute/virtualMachines/extensions",
  "apiVersion": "2018-06-01",
  "location": "[parameters('location')]",
  "properties": {
    "publisher": "Microsoft.Azure.Extensions",
    "type": "customScript",
    "typeHandlerVersion": "2.0",
    "autoUpgradeMinorVersion": true,
    "settings": {
      "fileUris": [
        "https://raw.githubusercontent.com/MicrosoftDocs/mslearn-welcome-to-
azure/master/configure-nginx.sh"
      ]
    },
    "protectedSettings": {
      "commandToExecute": "./configure-nginx.sh"
    }
  }
}
```

Specify dependent resources

You can't run the Custom Script Extension until the VM is available. All template resources provide a `dependsOn` property. This property helps Resource Manager determine the correct order to apply resources.

Here's what your template resource might look like after you add the `dependsOn` property.

JSON

Copy

```json
{
  "name": "[concat(parameters('vmName'), '/', 'ConfigureNginx')]",
  "type": "Microsoft.Compute/virtualMachines/extensions",
  "apiVersion": "2018-06-01",
  "location": "[parameters('location')]",
  "properties": {
    "publisher": "Microsoft.Azure.Extensions",
    "type": "customScript",
    "typeHandlerVersion": "2.0",
    "autoUpgradeMinorVersion": true,
    "settings": {
      "fileUris": [
        "https://raw.githubusercontent.com/MicrosoftDocs/mslearn-welcome-to-
azure/master/configure-nginx.sh"
      ]
    },
    "protectedSettings": {
      "commandToExecute": "./configure-nginx.sh"
    }
  },
  "dependsOn": [
    "[resourceId('Microsoft.Compute/virtualMachines/', parame-
ters('vmName'))]"
  ]
}
```

The bracket `[ ]` syntax means that you can provide an array, or list, of resources that must exist before applying this resource.

There are multiple ways to define a resource dependency. You can provide its name, such as "simpleLinuxVM", it's full name (including its namespace, type, and name), such as "Microsoft.Compute/virtualMachines/simpleLinuxVM", or by its resource ID.

This example uses the built-in `resourceId` function to get the VM's resource ID using its full name. This helps clarify which resource you're referring to and can help avoid ambiguity when more than one resource has a similar name.

The existing template provides a `vmName` parameter that defines the VM's name. This example uses the built-in `parameters` function to read it.

Summary

You now have everything you need to define the Custom Script Extension resource in your template. In the next part, you'll extend your template to use it.

Check your knowledge

1. How can you help Resource Manager determine the correct order to apply resources?

- ○ Use the `dependsOn` element to define when one resource must exist before another can be deployed.
- ○ In your template, define resources in the order they need to be deployed.
- ○ Set the `order` property on each resource.

Check your answers

100 XP

Exercise - Extend the Quickstart template to deploy a basic web site

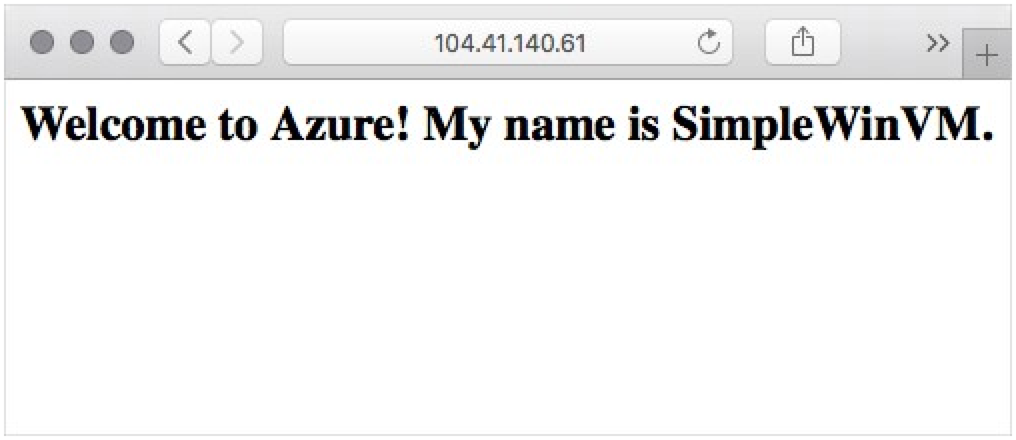
7 minutes

Choose which platform you want to run in the cloud

Windows Linux

Now that you've defined the template resource for the Custom Script Extension that configures IIS on your VM, let's add it to the existing VM template and run it.

Here's what the IIS configuration will look like.



Welcome to Azure! My name is SimpleWinVM.

Build the template

Here you'll download the template and modify it.

1. From Cloud Shell, run `curl` to download the template you used previously from GitHub.

```bash
curl https://raw.githubusercontent.com/Azure/azure-quickstart-tem-
plates/master/101-vm-simple-windows/azuredeploy.json > azuredeploy.json
```

2. Open **azuredeploy.json** through the Cloud Shell editor.

```bash
code azuredeploy.json
```

3. In the file, locate the `resources` section. Add the Custom Script Extension resource you built in the previous part to the top of this section.

   Here's the code as a refresher.

```JSON
{
  "name": "[concat(variables('vmName'),'/', 'ConfigureIIS')]",
  "type": "Microsoft.Compute/virtualMachines/extensions",
  "apiVersion": "2018-06-01",
  "location": "[parameters('location')]",
  "properties": {
    "publisher": "Microsoft.Compute",
    "type": "CustomScriptExtension",
    "typeHandlerVersion": "1.9",
    "autoUpgradeMinorVersion": true,
    "settings": {
      "fileUris": [
        "https://raw.githubusercontent.com/MicrosoftDocs/mslearn-wel-
come-to-azure/master/configure-iis.ps1"
      ]
    },
    "protectedSettings": {
       "commandToExecute": "powershell -ExecutionPolicy Unrestricted -
File configure-iis.ps1"
    }
  },
  "dependsOn": [
    "[resourceId('Microsoft.Compute/virtualMachines/', varia-
bles('vmName'))]"
```

```
    ]
  },
```

Note the comma `,` character at the end, which is needed to separate resources. The order you define resources doesn't matter, but here you add it to the top for simplicity.

4. In the file, locate the `securityRules` under the `resources` section. Add in a section to open port 80.

JSON

```json
        {
          "name": "allow_80",
          "properties": {
            "priority": 101,
            "access": "Allow",
            "direction": "Inbound",
            "destinationPortRange": "80",
            "protocol": "Tcp",
            "sourcePortRange": "*",
            "sourceAddressPrefix": "Internet",
            "destinationAddressPrefix": "*"
          }
        }
```

5. If you get stuck or want to compare your work, you can download the resulting file from GitHub.

bash

```bash
curl https://raw.githubusercontent.com/MicrosoftDocs/mslearn-build-
azure-vm-templates/master/windows/azuredeploy.json > azuredeploy.json
```

6. You're all done editing files. Select the ellipses in the corner and **Save**.

7. To close the editor, click the ellipses in the corner and then select **Close Editor**.

Verify the template

Here you'll validate the template from the CLI.

In practice, you might run lint tests or run your template through the Azure Resource Manager Visualizer before you run a test deployment.

Similar to what you did previously, run `az group deployment validate` to validate your template.

Azure CLI

```
az group deployment validate \
  --resource-group $RESOURCEGROUP \
  --template-file azuredeploy.json \
  --parameters adminUsername=$USERNAME \
  --parameters adminPassword=$PASSWORD \
  --parameters dnsLabelPrefix=$DNS_LABEL_PREFIX
```

Notice that this time you use the `--template-file` argument, and not `--template-uri`, because you are referencing a local file.

If you see errors, refer back to the previous part or compare your code to the reference implementation.

Deploy the template

Here you'll run a command that's similar to the one you ran earlier to deploy the template. Because you haven't modified any existing resources – the VM, its network settings, or the storage account – Resource Manager won't take any action on those resources. It will only apply the resource you just added that runs the Custom Script Extension that installs IIS on your VM.

Run `az group deployment create` to update your deployment.

Azure CLI

```
az group deployment create \
  --name MyDeployment \
  --resource-group $RESOURCEGROUP \
  --template-file azuredeploy.json \
  --parameters adminUsername=$USERNAME \
  --parameters adminPassword=$PASSWORD \
  --parameters dnsLabelPrefix=$DNS_LABEL_PREFIX
```

Again, notice that this time you use the `--template-file` argument because you are referencing a local file.

The command takes a couple minutes to complete. You see a large block of JSON as output, which describes the deployment.

Verify the deployment

The deployment succeeded, so let's see the resulting configuration in action.

1. Run `az vm show` to get the VM's IP address and store the result in a Bash variable.

   Azure CLI

   ```
   IPADDRESS=$(az vm show \
     --name SimpleWinVM \
     --resource-group $RESOURCEGROUP \
     --show-details \
     --query [publicIps] \
     --output tsv)
   ```

2. Run `curl` to access your web server.

   bash

   ```
   curl $IPADDRESS
   ```

   You see this.

   HTML

   ```
   <html><body><h2>Welcome to Azure! My name is SimpleW-
   inVM.</h2></body></html>
   ```

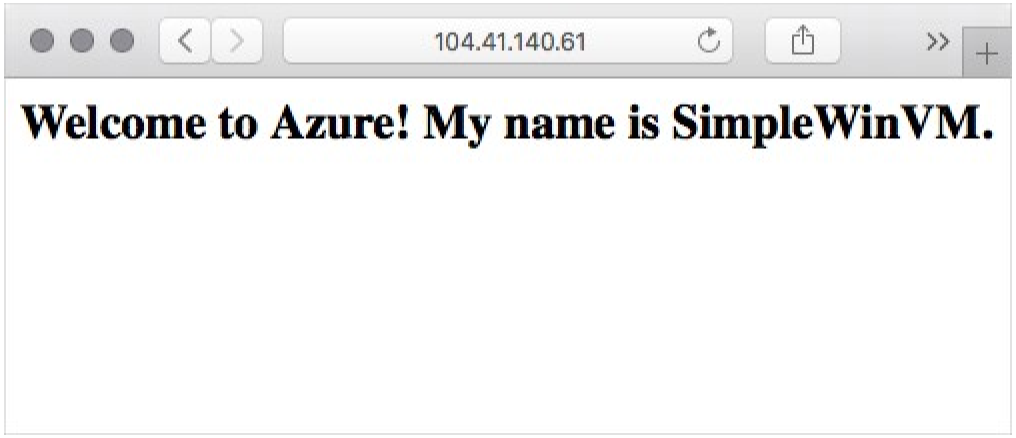
3. From a separate browser tab, navigate to your web site.

   First, print the IP address.

   bash

   ```bash
   echo $IPADDRESS
   ```

   Navigate to the IP address you see from a separate browser tab. You see this.

   Welcome to Azure! My name is SimpleWinVM.

Great work! With the Custom Script Extension resource in place, you're able to extend your deployment to do more.

Next unit: Summary

Continue >

✓ 100 XP

Exercise - Extend the Quickstart template to deploy a basic web site

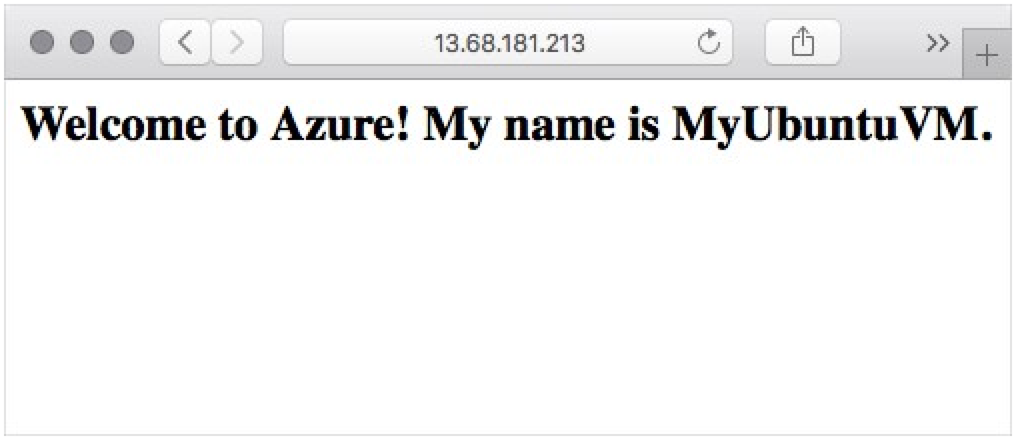
7 minutes

Choose which platform you want to run in the cloud

Windows Linux

Now that you've defined the template resource for the Custom Script Extension that configures Nginx on your VM, let's add it to the existing VM template and run it.

Here's what the Nginx configuration will look like.



Welcome to Azure! My name is MyUbuntuVM.

Build the template

Here you'll download the template and modify it.

1. From Cloud Shell, run `curl` to download the template you used previously from GitHub.

bash

```bash
curl https://raw.githubusercontent.com/Azure/azure-quickstart-tem-
plates/master/101-vm-simple-linux/azuredeploy.json > azuredeploy.json
```

2. Open **azuredeploy.json** through the Cloud Shell editor.

bash

```bash
code azuredeploy.json
```

3. In the file, locate the `resources` section. Add the Custom Script Extension resource you built in the previous part to the top of this section, then save the file.

   Here's the code as a refresher.

JSON

```json
{
  "name": "[concat(parameters('vmName'),'/', 'ConfigureNginx')]",
  "type": "Microsoft.Compute/virtualMachines/extensions",
  "apiVersion": "2018-06-01",
  "location": "[parameters('location')]",
  "properties": {
    "publisher": "Microsoft.Azure.Extensions",
    "type": "customScript",
    "typeHandlerVersion": "2.0",
    "autoUpgradeMinorVersion": true,
    "settings": {
      "fileUris": [
        "https://raw.githubusercontent.com/MicrosoftDocs/mslearn-wel-
come-to-azure/master/configure-nginx.sh"
      ]
    },
    "protectedSettings": {
       "commandToExecute": "./configure-nginx.sh"
    }
  },
  "dependsOn": [
    "[resourceId('Microsoft.Compute/virtualMachines/', parame-
ters('vmName'))]"
  ]
},
```

Note the comma `,` character at the end, which is needed to separate resources. The order you define resources doesn't matter, but here you add it to the top for simplicity.

4. In the file, under `resources` locate the "Microsoft.Network/networkSecurityGroups" section. Under "securityRules, add a new rule to open port 80. Use the following rule:

   JSON

   ```json
   {
     "name":"allow_80",
     "properties": {
     "priority": 101,
     "protocol": "TCP",
     "access": "Allow",
     "direction": "Inbound",
     "sourceAddressPrefix": "Internet",
     "sourcePortRange": "*",
     "destinationAddressPrefix": "*",
     "destinationPortRange": "80"
     }
   },
   ```

5. If you get stuck or want to compare your work, you can download the resulting file from GitHub.

   bash

   ```bash
   curl https://raw.githubusercontent.com/MicrosoftDocs/mslearn-build-azure-vm-templates/master/linux/azuredeploy.json > azuredeploy.json
   ```

6. You're all done editing files. Select the ellipses in the corner and **Save**.

7. To close the editor, click the ellipses in the corner and then select **Close Editor**.

Verify the template

Here you'll validate the template from the CLI.

In practice, you might run lint tests or run your template through the Azure Resource Manager Visualizer before you run a test deployment.

Similar to what you did previously, run `az group deployment validate` to validate your template.

Azure CLI

Copy

```
az group deployment validate \
  --resource-group $RESOURCEGROUP \
  --template-file azuredeploy.json \
  --parameters adminUsername=$USERNAME \
  --parameters authenticationType=password \
  --parameters adminPasswordOrKey=$PASSWORD \
  --parameters dnsLabelPrefix=$DNS_LABEL_PREFIX
```

Notice that this time you use the `--template-file` argument, and not `--template-uri`, because you are referencing a local file.

If you see errors, refer back to the previous part or compare your code to the reference implementation.

Deploy the template

Here you'll run a command that's similar to the one you ran earlier to deploy the template. Because you haven't modified any existing resources – the VM, its network settings, or the storage account – Resource Manager won't take any action on those resources. It will only apply the resource you just added that runs the Custom Script Extension that installs Nginx on your VM.

Run `az group deployment create` to update your deployment.

Azure CLI

Copy

```
az group deployment create \
  --name MyDeployment \
  --resource-group $RESOURCEGROUP \
  --template-file azuredeploy.json \
  --parameters adminUsername=$USERNAME \
  --parameters authenticationType=password \
  --parameters adminPasswordOrKey=$PASSWORD \
  --parameters dnsLabelPrefix=$DNS_LABEL_PREFIX
```

Again, notice that this time you use the `--template-file` argument because you are referencing a local file.

The command takes a couple minutes to complete. You see a large block of JSON as output, which describes the deployment.

Verify the deployment

The deployment succeeded, so let's see the resulting configuration in action.

1. Run `az vm show` to get the VM's IP address and store the result in a Bash variable.

   Azure CLI

   ```
   IPADDRESS=$(az vm show \
     --name simpleLinuxVM \
     --resource-group $RESOURCEGROUP \
     --show-details \
     --query [publicIps] \
     --output tsv)
   ```

2. Run `curl` to access your web server.

   bash

   ```
   curl $IPADDRESS
   ```

   You see this.

   HTML

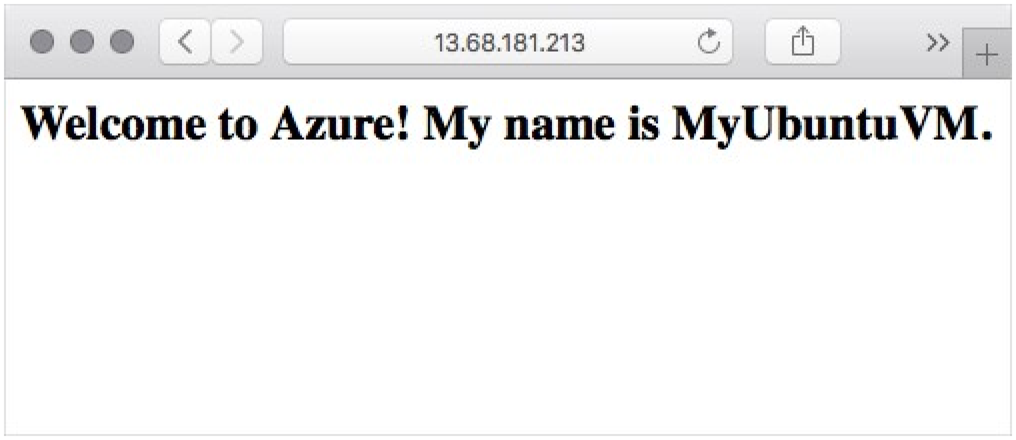
   ```
   <html><body><h2>Welcome to Azure! My name is simpleLi-
   nuxVM.</h2></body></html>
   ```

3. From a separate browser tab, navigate to your web site.

First, print the IP address.

```bash
echo $IPADDRESS
```

4. Navigate to the IP address you see from a separate browser tab. You see something like the following message:



Welcome to Azure! My name is MyUbuntuVM.

Great work! With the Custom Script Extension resource in place, you're able to extend your deployment to do more.

Next unit: Summary

Continue >

✓ 200 XP

Summary

2 minutes

In this module, you defined what Azure Resource Manager means and what's in a Resource Manager template.

Resource Manager templates are *declarative*, meaning that you define *what* resources you need and let Resource Manager handle the deployment details.

You used an existing Azure Quickstart template to deploy a VM. Quickstart templates are a great way to jump-start your deployments. They also demonstrate recommended practices you can learn from as you author your own templates.

You also extended your Quickstart template to configure web server software on your VM. Extending an existing template is a great way to get something running quickly and understand how the pieces fit together.

Resource Manager templates are also *composable*. As you build out your deployments, you can write smaller templates that each define a piece of the system and then combine them to create a complete system.

Think about the analysts you support at your financial services company. As you build your library of Resource Manager templates, you'll be able to assemble deployments for each analyst much more quickly. After each financial model completes, you need to run only a single Azure CLI command to tear down the deployment.

Clean up

When you're working in your own subscription, it's a good idea at the end of a project to identify whether you still need the resources you created. Resources left running can cost you money. You can delete resources individually or delete the resource group to delete the entire set of resources.

Learn more

A great way to learn is by doing. Try writing a Resource Manager template that automates one of your deployments.

Refer to the Resource Manager on Azure documentation to learn more about how to create, deploy, manage, audit, and troubleshoot your templates.

Here are some resources that go into more detail on what you learned in this module.

- Azure Quickstart Templates
- Azure Resource Manager Visualizer
- Deploy a simple Windows VM
- Deploy a simple Ubuntu Linux VM

Here are some topics that we did not discuss here, but you might be interested in as you explore and build your own templates.

- Manage access using RBAC and Azure Resource Manager templates
- Use condition in Azure Resource Manager templates
- Integrate Azure Key Vault in Resource Manager Template deployment
- Create linked Azure Resource Manager templates
- Best Practices For Using Azure Resource Manager Templates
- Troubleshoot common Azure deployment errors with Azure Resource Manager

Check your knowledge

1. Say you want to create a reusable template that uses the Custom Script Extension to configure web content on a VM. What's the best way to enable deployments to specify the script that configures web content?

[✓ Provide a parameter that specifies the script location.] ✓

Parameters promote reuse. Their values are filled in when the template runs.

○ Provide a variable that specifies the script location.

○

Provide a default script location in your template and then use a nested template to override that location.

Module complete:

Unlock achievement