Introduction

- 3 minutes

Creating administration scripts is a powerful way to optimize your work flow. You can automate common, repetitive tasks. Once a script has been verified, it will run consistently, likely reducing errors.

Suppose you work at a company that uses Azure Virtual Machines (VMs) to test your Customer Relationship Management (CRM) software. The VMs are built from images that include a web front end, a web service that implements business logic, and an SQL database.

You've been executing multiple rounds of tests on a single VM, but you've noticed that changes in the database and configuration files can cause inconsistent results. In one case, a bug created a phone call record with no corresponding customer in the database. The orphaned record caused subsequent integration tests to fail, even after you fixed the bug. You plan to solve this problem by using a fresh VM deployment for each testing cycle. You want to automate the VM creation setup because it will be executed many times per week.

Here, you'll learn how to manage Azure resources using Azure PowerShell. You'll use Azure PowerShell interactively for one-off tasks, and write scripts to automate repeated tasks.

Learning objectives

In this module, you will:

- Decide if Azure PowerShell is the right tool for your Azure administration tasks
- Install Azure PowerShell on Linux, macOS, and/or Windows
- Connect to an Azure subscription using Azure PowerShell
- Create Azure resources using Azure PowerShell

Prerequisites

- Experience with a command-line interface such as PowerShell or Bash

- Knowledge of basic Azure concepts such as resource groups and Virtual Machines
- Experience administering Azure resources using the Azure portal

---

Next unit: Decide if Azure PowerShell is right for your tasks

Decide if Azure PowerShell is right for your tasks

Completed100 XP

- 5 minutes

Suppose you need to choose a tool to administer the Azure resources you'll use to test your Customer Relationship Management (CRM) system. Your tests require you to create resource groups and provision virtual machines (VMs).

You want something that is easy for administrators to learn, but powerful enough to automate the installation and setup of multiple virtual machines, or script a full application environment. There are multiple tools available, and you need to find the best one for your people and tasks.

What tools are available?

Azure provides three administration tools:

- The Azure portal
- The Azure CLI
- Azure PowerShell

They all offer approximately the same amount of control; any task that you can do with one of the tools, you can likely do with the other two. All three are cross-platform, running on Windows, macOS, and Linux. They differ in syntax, setup requirements, automation support.

Here, we'll describe each of the three options and provide some guidance on how to decide among them.

What is the Azure portal?

The Azure portal is a website that lets you create, configure, and alter the resources in your Azure subscription. The portal is a Graphical User Interface (GUI) that makes it convenient to locate the resource you need and execute any required changes. It also guides you through complex administrative tasks by providing wizards and tooltips.

The portal doesn't provide any way to automate repetitive tasks. For example, to set up 15 VMs, you would need to create them one by one, completing the wizard for each VM. This method can be time consuming, and is error prone for complex tasks.

What is the Azure CLI?

The Azure CLI is a cross-platform command-line program to connect to Azure and execute administrative commands on Azure resources. For example, to create a VM, you could use the following command:

Azure CLICopy

```
az vm create \
  --resource-group CrmTestingResourceGroup \
  --name CrmUnitTests \
  --image UbuntuLTS
  ...
```

The Azure CLI is available two ways: inside a browser via the Azure Cloud Shell, or with a local install on Linux, Mac, or Windows. In both cases, you can use it interactively, or use it with scripts to automate tasks. For interactive use, you'd first launch a shell such as `cmd.exe` on Windows, or Bash on Linux or macOS, then issue the commands at the shell prompt. To automate repetitive tasks, you'd assemble the commands into a shell script using the script syntax of your chosen shell, then execute the script.

What is Azure PowerShell?

Azure PowerShell is a module you add to PowerShell to let you connect to your Azure subscription and manage resources. Azure PowerShell requires PowerShell to function. PowerShell provides services like the shell window, command parsing, and so on. The Azure Az PowerShell module adds Azure-specific commands.

For example, Azure PowerShell provides the **New-AzVM** command, which creates a virtual machine for you in your Azure subscription. To use it, you would launch the PowerShell application, then issue the following command:

Azure PowerShellCopy
```
New-AzVm `
    -ResourceGroupName "CrmTestingResourceGroup" `
    -Name "CrmUnitTests" `
    -Image "UbuntuLTS"
    ...
```

Azure PowerShell is also available two ways: inside a browser via the Azure Cloud Shell, or with a local install on Linux, Mac, or Windows. In both cases, you have two modes to choose from. You can use it in interactive mode, in which you manually issue one command at a time. Or, in scripting mode, where you execute a script that consists of multiple commands.

How to Choose an administrative tool

There's approximate parity between the portal, the Azure CLI, and Azure PowerShell with respect to the Azure objects they can administer and the configurations they can create. They're also all cross-platform. Typically, you'll consider several other factors when making your choice:

- **Automation**: Do you need to automate a set of complex or repetitive tasks? Azure PowerShell and the Azure CLI support automation, while Azure portal doesn't.
- **Learning curve**: Do you need to complete a task quickly without learning new commands or syntax? The Azure portal doesn't require you to learn syntax or memorize commands. In Azure PowerShell and the Azure CLI, you must know the detailed syntax for each command you use.
- **Team skillset**: Does your team have existing expertise? For example, your team may have used PowerShell to administer Windows. If so, they'll quickly become comfortable using Azure PowerShell.

Example

Recall that you're choosing an administrative tool to create the test environments for your CRM application. Your administrators have two specific Azure tasks they need to complete:

1. Create one resource group for each category of testing (unit, integration, and acceptance).
2. Create multiple VMs in each resource group before every round of testing.

To create the resource groups, the Azure portal is a reasonable choice. These tasks are one-offs, so you don't need scripts to complete them.

Finding the best tool to create the VMs is a more challenging decision. You need to create several VMs, and you need to create them repeatedly, likely several times each week. For these tasks, you'll want automation, so the Azure portal isn't a good choice. In this case, either Azure PowerShell or the Azure CLI will meet your needs. If your team members have some existing PowerShell knowledge, Azure PowerShell will likely be the best match. Azure PowerShell is available on the operating systems your admin team uses, it supports automation, and should be quick for your team to learn.

Most administrators' first experience with Azure is in the Portal. It's a great place to start as it provides a clean, well-structured graphical interface, but provides limited options for automation. When you need automation, Azure gives you two options: Azure PowerShell for admins with PowerShell experience and the Azure CLI for everyone else.

In practice, businesses typically have a mix of one-off and repetitive tasks. So, it's common to use both the Portal and a scripting solution. In our CRM example, it's appropriate to create the resource groups via the Portal and automate the VM creation with PowerShell.

The rest of this module focuses on installing and using Azure PowerShell.

Next unit: Install PowerShell

Install PowerShell

Completed 100 XP

- 10 minutes

Suppose you've chosen Azure PowerShell as your automation solution. Your administrators prefer to run their scripts locally rather than in the Azure Cloud Shell. The team uses machines that run Linux, macOS, and Windows. You need to get Azure PowerShell working on all their devices.

What must be installed?

We'll go through the actual installation instructions in the next unit, but let's look at the two components that make up Azure PowerShell.

- **The base PowerShell product** This comes in two variants: Windows PowerShell and PowerShell 7.x, which can be installed on Windows, macOS, and Linux.
- **The Azure Az PowerShell module** This extra module must be installed to add the Azure-specific commands to PowerShell.

Tip

PowerShell 7.0.6 LTS, PowerShell 7.1.3, or higher is the recommended version of PowerShell for use with the Azure Az PowerShell module on all platforms.

Once you've installed the base product, you'll then add the Azure PowerShell module to your installation.

How to install PowerShell

On both Linux and macOS, you'll use a package manager to install PowerShell Core. The recommended package manager differs by OS and distribution.

Note

PowerShell is available in the Microsoft repository, so you'll first need to add that repository to your package manager.

Linux

On Linux, the package manager will change based on the Linux distribution you choose.

Distribution(s)	**Package manager**
Ubuntu, Debian	`apt-get`
Red Hat, CentOS	`yum`
OpenSUSE	`zypper`
Fedora	`dnf`

Mac

On macOS, you'll use `Homebrew` to install PowerShell.

In the next section, you'll go through the detailed installation steps for some common platforms.

Next unit: Exercise - Install Azure PowerShell

Exercise - Install Azure PowerShell

Completed 100 XP

- 10 minutes

Choose your platform

Linux Mac [✓ Windows]

In this unit, you'll learn how to check the version of **PowerShell** installed on your local machine and install the latest version.

Note

This exercise guides you through creating a local installation of PowerShell tools. The remainder of this module uses the Azure Cloud Shell, so you can leverage the free subscription support in Microsoft Learn. If you prefer, consider this exercise as an optional activity and just review the instructions.

Windows

Windows PowerShell is included with the Windows operating system; however, we recommend installing PowerShell 7.0.6 LTS, PowerShell 7.1.3, or higher for use with Azure Az PowerShell module PowerShell. You can check which version is installed using the following steps:

1. In the **System tray search box**, type **PowerShell**. You may have multiple shortcut links:
    - PowerShell 7 (x64) - The 64-bit version. Generally, you should choose this shortcut.

    - Windows PowerShell - The 64-bit version included with Windows.
    - Windows PowerShell (x86) - A 32-bit version installed on 64-bit Windows.
    - Windows PowerShell ISE - The Integrated Scripting Environment (ISE) is used for writing scripts in Windows PowerShell.
    - Windows PowerShell ISE (x86) - A 32-bit version of the ISE on Windows.
2. Select the best match PowerShell icon.
3. Type the following command to determine the version of PowerShell installed.

   PowerShellCopy
   ```
   $PSVersionTable.PSVersion
   ```

   or

   PowerShellCopy
   ```
   pwsh -ver
   ```

   If the major version number is lower than 7, follow the instructions to upgrade existing Windows PowerShell. It's important to install the SDK to support .NET tools, as well.

   You need the .NET SDK installed to run this command.

   PowerShellCopy
   ```
   dotnet tool install --global PowerShell
   ```

   After the .NET tool is installed, run the PowerShell version command again to verify your installation.

You'll also need to set up your local machine(s) to support PowerShell. In the next unit, we'll review commands you can add, including the Azure Az PowerShell module.

Next unit: Create an Azure Resource using scripts in Azure PowerShell

Create an Azure Resource using scripts in Azure PowerShell

Completed100 XP

- 5 minutes

Choose your platform

○ Linux ○ Mac [✓ ● Windows]

Ins **interactive mode**, PowerShell lets you write commands and execute them immediately.

Recall that the overall goal in the Customer Relationship Management (CRM) example is to create three test environments containing Virtual Machines. You'll use resource groups to ensure the VMs are organized into separate environments: one for unit testing, one for integration testing, and one for acceptance testing. You only need to create the resource groups once, so using the PowerShell interactive mode in this use case is a good choice.

When you enter a command into PowerShell, PowerShell matches the command to a *cmdlet*, and PowerShell then performs the requested action. We'll look at some common commands you can use, then we'll look into installing the Azure support for PowerShell.

What are PowerShell cmdlets?

A PowerShell command is called a **cmdlet** (pronounced "command-let"). A cmdlet is a command that manipulates a single feature. The term **cmdlet** is intended to imply "small command". By convention, cmdlet authors are encouraged to keep cmdlets simple and single-purpose.

The base PowerShell product ships with cmdlets that work with features such as sessions and background jobs. You can add modules to your PowerShell installation to get cmdlets that manipulate other features. For example, there are third-party modules to work with ftp, administer your operating system, access the file system, and so on.

Cmdlets follow a verb-noun naming convention; for example, `Get-Process`, `Format-Table`, and `Start-Service`. There's also a convention for verb choice: "get" to retrieve data, "set" to insert or update data, "format" to format data, "out" to direct output to a destination, and so on.

Cmdlet authors are encouraged to include a help file for each cmdlet. The `Get-Help` cmdlet displays the help file for any cmdlet. For example, to get help on the `Get-ChildItem` cmdlet, enter the following statement in a Windows PowerShell session:

PowerShellCopy

```
Get-Help -Name Get-ChildItem -Detailed
```

What is a PowerShell module?

Cmdlets are shipped in *modules*. A PowerShell Module is a DLL that includes the code to process each available cmdlet. You'll load cmdlets into PowerShell by loading the module in which they're contained. You can get a list of loaded modules using the `Get-Module` command:

PowerShellCopy

```
Get-Module
```

This command will output something like:

OutputCopy

```
ModuleType Version    Name                               ExportedCommands
---------- -------    ----                               ----------------
Manifest   3.1.0.0    Microsoft.PowerShell.Management    {Add-Computer, Add-Content,
Checkpoint-Computer, Clear-Con...
Manifest   3.1.0.0    Microsoft.PowerShell.Utility       {Add-Member, Add-Type,
Clear-Variable, Compare-Object...}
Binary     1.0.0.1    PackageManagement                  {Find-Package, Find-
PackageProvider, Get-Package, Get-Pack...
Script     1.0.0.1    PowerShellGet                      {Find-Command, Find-
DscResource, Find-Module, Find-RoleCap...
Script     2.0.0      PSReadline                         {Get-PSReadLineKeyHandler,
Get-PSReadLineOption, Remove-PS...
```

What is the Az PowerShell module?

Az is the formal name for the Azure PowerShell module, which contains cmdlets to work with Azure features. It contains hundreds of cmdlets that let you control nearly every aspect of every Azure resource. You can work with resource groups, storage, virtual

machines, Azure Active Directory, containers, machine learning, and so on.
The **Az** module is an open-source component available on GitHub.

Note

You might have seen or used Azure PowerShell commands that used a `-AzureRM` format. Because Az PowerShell modules now have all the capabilities of AzureRM PowerShell modules and more, we'll retire AzureRM PowerShell modules on 29 February 2024. To avoid service interruptions, **update your scripts** that use AzureRM PowerShell modules to use Az PowerShell modules by 29 February 2024. To automatically update your scripts, follow the **quickstart guide**.

Install the Az PowerShell module

The Az PowerShell module is available from a global repository called the PowerShell Gallery. You can install the module onto your local machine through the `Install-Module` cmdlet.

To install the latest Azure Az PowerShell module, run the following commands:

1. Open the **Start** menu and enter **PowerShell**.
2. Select the **PowerShell** icon.
3. Enter the following command, and then press `Enter`.

   PowerShellCopy

   ```
   Install-Module -Name Az -Scope CurrentUser -Repository PSGallery
   ```

The previous command installs the module for your current user (controlled by the `Scope` parameter).

The command relies on NuGet to retrieve components. So depending on the version you've installed, you might be prompted to download and install the latest version of NuGet.

OutputCopy

```
NuGet provider is required to continue
PowerShellGet requires NuGet provider version '2.8.5.201' or newer to interact with
NuGet-based repositories. The NuGet
 provider must be available in 'C:\Program
Files\PackageManagement\ProviderAssemblies' or
'C:\Users\<username>\AppData\Local\PackageManagement\ProviderAssemblies'. You can
also install the NuGet provider by running
```

```
'Install-PackageProvider -Name NuGet -MinimumVersion 2.8.5.201 -Force'. Do you want
PowerShellGet to install and import
 the NuGet provider now?
 [Y] Yes  [N] No  [S] Suspend  [?] Help (default is "Y"):
```

Enter **Y** and press `Enter`.

By default, the PowerShell Gallery isn't configured as a trusted repository for PowerShellGet. Each time you perform an installation from an untrusted repository, you'll be prompted to confirm you want to install the module with following output:

OutputCopy

```
You are installing the modules from an untrusted repository. If you trust this
repository, change its
InstallationPolicy value by running the Set-PSRepository cmdlet. Are you sure you
want to install the modules from
'PSGallery'?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is
"N"):
```

Enter **Y** or **A**, then press `Enter`.

Script execution failed

Depending on your security configuration, `Import-Module` might fail with something like the following output:

OutputCopy

```
import-module : File C:\Program Files\PowerShell\Modules\az\6.3.0\Az.psm1 cannot be
loaded
because running scripts is disabled on this system. For more information, see
about_Execution_Policies at
https:/go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ import-module Az
+ ~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : SecurityError: (:) [Import-Module], PSSecurityException
    + FullyQualifiedErrorId :
UnauthorizedAccess,Microsoft.PowerShell.Commands.ImportModuleCommand
```

It might also fail by not responding at all. In this case, press `Ctrl+C` to stop the program.

Both behaviors typically indicate that the execution policy is "Restricted", meaning you can't run modules that you download from an external source, including the PowerShell

Gallery. You can check by running the cmdlet `Get-ExecutionPolicy`. If it returns "Restricted", then:

1. Use the `Set-ExecutionPolicy` cmdlet to change the policy to "RemoteSigned":

   PowerShellCopy

   ```
   Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
   ```

   You'll be prompted for permission:

   OutputCopy

   ```
   The execution policy helps protect you from scripts that you do not
   trust. Changing the execution policy might expose
   you to the security risks described in the about_Execution_Policies help
   topic at
   https:/go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the
   execution policy?
   [Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
   (default is "N"): Y
   ```

2. Enter **Y** or **A**, then press `Enter`.
3. At the command prompt, use the up arrow on your keyboard and rerun the `Install-Module` command for Azure.

You should be able to see the Az PowerShell module loading. After it completes, you'll be able to use `Import-Module` to load the cmdlets.

Update a PowerShell module

You may get a warning or error message that indicates a version of the Azure PowerShell module is already installed. If so, you can issue the following command to update to the *latest* version.

PowerShellCopy

```
Update-Module -Name Az
```

As with the `Install-Module` cmdlet, answer **Yes** or **Yes to All** when prompted to trust the module. You can also use the `Update-Module` command to reinstall a module if you're having trouble with it.

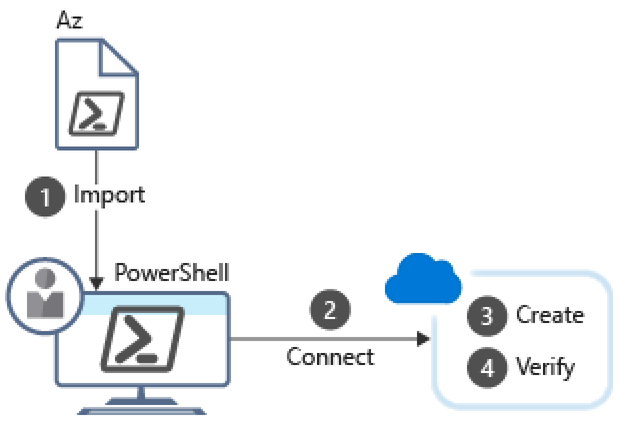
Example: How to create a resource group with Azure PowerShell

Once you've installed the Azure module, you can begin working with Azure. Let's do a common task: creating a Resource Group. As you know, we use resource groups to administer related resources together. Creating a new resource group is one of the first tasks you'll do when starting a new Azure solution.

There are four steps you need to perform:

1. Import the Azure cmdlets.
2. Connect to your Azure subscription.
3. Create the resource group.
4. Verify that creation was successful.

The following illustration shows an overview of these steps.



Each step corresponds to a different cmdlet.

Import the Azure cmdlets

Beginning with PowerShell 3.0, modules are loaded automatically when you use a cmdlet within the module. It's no longer necessary to manually import PowerShell modules unless you've changed the default module autoloading settings.

Connect

When you're working with a local install of Azure PowerShell, you'll need to authenticate before you can execute Azure commands. The `Connect-AzAccount` cmdlet prompts for your Azure credentials, then connects to your Azure subscription. It has many optional parameters, but if all you need is an interactive prompt, you don't need any parameters:

Azure PowerShellCopy

```
Connect-AzAccount
```

Work with subscriptions

If you're new to Azure, you probably only have a single subscription. But if you've been using Azure for a while, you might have created multiple Azure subscriptions. You can configure Azure PowerShell to execute commands against a particular subscription.

You can only be in one subscription at a time. Use the `Get-AzContext` cmdlet to determine which subscription is active. If it's not the correct one, you can change subscriptions using another cmdlet.

1. Get a list of all subscription names in your account with the `Get-AzSubscription` command.
2. Change the subscription by passing the name of the one to select.

Azure PowerShellCopy

```
Set-AzContext -Subscription '00000000-0000-0000-0000-000000000000'
```

If you need to look up the **Subscription ID**, open Azure and select **Subscriptions** on the home page.

Get a list of all resource groups

You can retrieve a list of all Resource Groups in the active subscription.

Azure PowerShellCopy

```
Get-AzResourceGroup
```

To get a more concise view, you can send the output from the `Get-AzResourceGroup` to the `Format-Table` cmdlet using a pipe '|'.

Azure PowerShellCopy

```
Get-AzResourceGroup | Format-Table
```

The output will look something like this:

OutputCopy

```
ResourceGroupName                  Location       ProvisioningState Tags TagsTable
ResourceId
-----------------                  --------       ----------------- ---- --------- --
--------
cloud-shell-storage-southcentralus southcentralus Succeeded
/subscriptions/00000000-0000-0000...
ExerciseResources                  eastus         Succeeded
/subscriptions/00000000-0000-0000...
```

Create a resource group

As you know, when you're creating resources in Azure, you'll always place them into a resource group for management purposes. A resource group is often one of the first things you'll create when starting a new application.

You can create resource groups by using the `New-AzResourceGroup` cmdlet. You must specify a name and location. The name must be unique within your subscription. The location determines where the metadata for your resource group will be stored (which may be important to you for compliance reasons). You use strings like "West US", "North Europe", or "West India" to specify the location. As with most of the Azure cmdlets, `New-AzResourceGroup` has many optional parameters. However, the core syntax is:

PowerShellCopy

```
New-AzResourceGroup -Name <name> -Location <location>
```

Note

Remember, we will be working in an active Azure sandbox, which creates the Resource Group for you. Use the previous command if you prefer to work in your own subscription.

Verify the resources

The `Get-AzResource` lists your Azure resources, which is useful here to verify the resources were created and the resource group creation was successful.

PowerShellCopy

```
Get-AzResource
```

Like the `Get-AzResourceGroup` command, you can get a more concise view through the `Format-Table` cmdlet:

PowerShellCopy

```
Get-AzResource | Format-Table
```

You can also filter it to specific resource groups to only list resources associated with that group:

PowerShellCopy

```
Get-AzResource -ResourceGroupName ExerciseResources
```

Create an Azure Virtual Machine

Another common task you can do with PowerShell is to create VMs.

Azure PowerShell provides the `New-AzVm` cmdlet to create a virtual machine. The cmdlet has many parameters to let it handle the large number of VM configuration settings. Most of the parameters have reasonable default values, so we only need to specify five things:

- **ResourceGroupName**: The resource group into which the new VM will be placed.
- **Name**: The name of the VM in Azure.
- **Location**: Geographic location where the VM will be provisioned.
- **Credential**: An object containing the username and password for the VM admin account. We'll use the `Get-Credential` cmdlet. This cmdlet will prompt for a username and password and package it into a credential object.
- **Image**: The operating system image to use for the VM, which is typically a Linux distribution or Windows Server.

PowerShellCopy

```
New-AzVm
    -ResourceGroupName <resource group name>
    -Name <machine name>
    -Credential <credentials object>
    -Location <location>
    -Image <image name>
```

You can supply these parameters directly to the cmdlet as shown in the previous example. Alternatively, you can use other cmdlets to configure the virtual machine, such as `Set-AzVMOperatingSystem`, `Set-AzVMSourceImage`, `Add-AzVMNetworkInterface`, and `Set-AzVMOSDisk`.

Here's an example that strings the `Get-Credential` cmdlet together with the `-Credential` parameter:

PowerShellCopy

```
New-AzVM -Name MyVm -ResourceGroupName ExerciseResources -Credential (Get-Credential)
...
```

The `AzVM` suffix is specific to VM-based commands in PowerShell. There are several others you can use:

Command	**Description**
`Remove-AzVM`	Deletes an Azure VM.
`Start-AzVM`	Start a stopped VM.
`Stop-AzVM`	Stop a running VM.
`Restart-AzVM`	Restart a VM.
`Update-AzVM`	Updates the configuration for a VM.

Example: Getting information for a VM

You can list the VMs in your subscription using the `Get-AzVM -Status` command. This command also supports entering a specific VM by including the `-Name` property. Here, we'll assign it to a PowerShell variable:

PowerShellCopy

```
$vm = Get-AzVM  -Name MyVM -ResourceGroupName ExerciseResources
```

The interesting thing is that now your VM is an *object* with which you can interact. For example, you can make changes to that object, then push changes back to Azure by using the `Update-AzVM` command:

PowerShellCopy

```
$ResourceGroupName = "ExerciseResources"
$vm = Get-AzVM  -Name MyVM -ResourceGroupName $ResourceGroupName
$vm.HardwareProfile.vmSize = "Standard_DS3_v2"
```

```
Update-AzVM -ResourceGroupName $ResourceGroupName  -VM $vm
```

The interactive mode in PowerShell is appropriate for one-off tasks. In our example, we'll likely use the same resource group for the lifetime of the project, so creating it interactively is reasonable. Interactive mode is often quicker and easier for this task than writing a script and executing that script exactly once.

Next unit: Exercise - Create an Azure Resource using scripts in Azure PowerShell

Exercise - Create an Azure Resource using scripts in Azure PowerShell

Completed100 XP

- 15 minutes

This module requires a sandbox to complete.

A **sandbox** gives you access to free resources. Your personal subscription will not be charged. The sandbox may only be used to complete training on Microsoft Learn. Use for any other reason is prohibited, and may result in permanent loss of access to the sandbox.

Microsoft provides this lab experience and related content for educational purposes. All presented information is owned by Microsoft and intended solely for learning about the covered products and services in this Microsoft Learn module.

Activate sandbox

Recall our original scenario: creating VMs to test our CRM software. When a new build is available, we want to spin up a new VM so we can test the full install experience from a clean image. When we're finished, we want to delete the VM.

Let's try the commands you would use to create a VM.

Create a Linux VM with Azure PowerShell

Because we're using the Azure sandbox, you won't have to create a resource group. Instead, use the resource group **[sandbox resource group name]**. In addition, be aware of the location restrictions.

Let's create a new Azure VM with PowerShell.

1. Use the `New-AzVm` cmdlet to create a VM.
    - Use the resource group **[sandbox resource group name]**.
    - Give the VM a name. Typically, you want to use something meaningful that identifies the purposes of the VM, location, and (if there's more than one) instance number. We'll use "testvm-eus-01" for "Test VM in East US, instance 1". Come up with your own name based on where you'll place the VM.
    - Select a location close to you from the following list, available in the Azure sandbox. Make sure to change the value in the following example command if you're using copy and paste.
        - westus2
        - southcentralus
        - centralus
        - eastus
        - westeurope
        - southeastasia
        - japaneast
        - brazilsouth
        - australiasoutheast
        - centralindia
    - Use "Canonical:0001-com-ubuntu-server-focal:20_04-lts:latest" for the image. This image is Ubuntu Linux.
    - Use the `Get-Credential` cmdlet and feed the results into the `Credential` parameter.

      Important

      See the **Linux VM FAQ** for username and password limitations. Passwords must be 12 - 123 characters in length, and meet three of the following four complexity requirements:

        - Have lowercase characters
        - Have uppercase characters
        - Have a digit
        - Have a special character (Regex match [\W_])

   - Add the `-OpenPorts` parameter and pass "22" as the port. This port will let us SSH into the machine.
   - Create a public IP address name. You'll use this name to create and find your static IP address to sign in to the machine.

   PowerShellCopy
   ```
   New-AzVm -ResourceGroupName [sandbox resource group name] -Name "testvm-
   eus-01" -Credential (Get-Credential) -Location "eastus" -Image
   Canonical:0001-com-ubuntu-server-focal:20_04-lts:latest -OpenPorts 22 -
   PublicIpAddressName "testvm-01"
   ```
   Tip

   You can use the **Copy** button to copy commands to the clipboard. To paste, right-click on a new line in the Cloud Shell terminal and select **Paste**, or use the Shift+Insert keyboard shortcut (⌘+V on macOS).

2. Create a username and password, then press `Enter`. PowerShell will start creating your VM.
3. The VM creation takes a few minutes to complete. After completion, you can query it and assign the VM object to a variable (`$vm`).

   PowerShellCopy
   ```
   $vm = (Get-AzVM -Name "testvm-eus-01" -ResourceGroupName [sandbox
   resource group name])
   ```

4. Query the value to dump out the information about the VM.

   PowerShellCopy
   ```
   $vm
   ```

   You should see something like the following output:

   PowerShellCopy
   ```
   ResourceGroupName : [sandbox resource group name]
   Id                : /subscriptions/00000000-0000-0000-0000-
   000000000000/resourceGroups/[sandbox resource group
   name]/providers/Microsoft.Compute/virtualMachines/testvm-eus-01
   VmId              : 00000000-0000-0000-0000-000000000000
   Name              : testvm-eus-01
   Type              : Microsoft.Compute/virtualMachines
   Location          : eastus
   Tags              : {}
   HardwareProfile   : {VmSize}
   NetworkProfile    : {NetworkInterfaces}
   OSProfile         : {ComputerName, AdminUsername, LinuxConfiguration,
   Secrets}
   ProvisioningState : Succeeded
   ```

```
StorageProfile     : {ImageReference, OsDisk, DataDisks}
```

5. You can reach into complex objects through a dot (".") notation. For example, to see the properties in the `VMSize` object associated with the HardwareProfile section, run the following command:

   PowerShellCopy
   ```
   $vm.HardwareProfile
   ```

6. Or, to get information on one of the disks, run the following command:

   PowerShellCopy
   ```
   $vm.StorageProfile.OsDisk
   ```

7. You can even pass the VM object into other cmdlets. For example, running the following command will show you all available sizes for your VM:

   PowerShellCopy
   ```
   $vm | Get-AzVMSize
   ```

8. Now, run the following command to get your public IP address:

   PowerShellCopy
   ```
   Get-AzPublicIpAddress -ResourceGroupName [sandbox resource group name] -
   Name "testvm-01"
   ```

9. With the IP address, you can connect to the VM with SSH. For example, if you used the username "bob", and the IP address is "205.22.16.5", running this command would connect to the Linux machine:

   PowerShellCopy
   ```
   ssh bob@205.22.16.5
   ```

   Sign out by entering `exit`.

Delete a VM

To try out some more commands, let's delete the VM. We'll shut it down first:

PowerShellCopy
```
Stop-AzVM -Name $vm.Name -ResourceGroupName $vm.ResourceGroupName
```

Now, let's delete the VM by running the `Remove-AzVM` cmdlet:

PowerShellCopy
```
Remove-AzVM -Name $vm.Name -ResourceGroupName $vm.ResourceGroupName
```

Run this command to list all the resources in your resource group:

PowerShellCopy
```
Get-AzResource -ResourceGroupName $vm.ResourceGroupName | Format-Table
```

You should see a bunch of resources (disks, virtual networks, and so on) that all still exist.

OutputCopy
```
Microsoft.Compute/disks
Microsoft.Network/networkInterfaces
Microsoft.Network/networkSecurityGroups
Microsoft.Network/publicIPAddresses
Microsoft.Network/virtualNetworks
```

The `Remove-AzVM` command *just deletes the VM*. It doesn't clean up any of the other resources. At this point, we'd likely just delete the resource group itself and be done with it. However, let's run through the exercise to clean it up manually. You should see a pattern in the commands.

1. Delete the network interface:

   PowerShellCopy
   ```
   $vm | Remove-AzNetworkInterface –Force
   ```

2. Delete the managed OS disks and storage account:

   PowerShellCopy
   ```
   Get-AzDisk -ResourceGroupName $vm.ResourceGroupName -DiskName
   $vm.StorageProfile.OSDisk.Name | Remove-AzDisk -Force
   ```

3. Next, delete the virtual network:

   PowerShellCopy
   ```
   Get-AzVirtualNetwork -ResourceGroupName $vm.ResourceGroupName | Remove-
   AzVirtualNetwork -Force
   ```

4. Delete the network security group:

   PowerShellCopy
   ```
   Get-AzNetworkSecurityGroup -ResourceGroupName $vm.ResourceGroupName |
   Remove-AzNetworkSecurityGroup -Force
   ```

5. And finally, delete the public IP address:

   PowerShellCopy
   ```
   Get-AzPublicIpAddress -ResourceGroupName $vm.ResourceGroupName | Remove-
   AzPublicIpAddress -Force
   ```

We should have caught all the created resources. Check the resource group just to be sure. We performed many manual commands here, but a better approach would have been to write a *script*. Then we could reuse this logic later to create or delete a VM. Let's look at scripting with PowerShell.

Next unit: Create and save scripts in Azure PowerShell

Create and save scripts in Azure PowerShell

Completed 100 XP

- 10 minutes

Complex or repetitive tasks often take a great deal of administrative time. Organizations prefer to automate these tasks to reduce costs and avoid errors.

Automation is important in the Customer Relationship Management (CRM) company example. There, you're testing your software on multiple Linux Virtual Machines (VMs) that you need to continuously delete and recreate. You want to use a PowerShell script to automate the creation of the VMs versus creating them manually each time.

Beyond the core operation of creating a VM, you have a few more requirements for your script:

- You'll create multiple VMs, so you want to put the creation inside a loop
- You need to create VMs in three different resource groups, so the name of the resource group should be passed to the script as a parameter

In this section, you'll see how to write and execute an Azure PowerShell script that meets these requirements.

What is a PowerShell script?

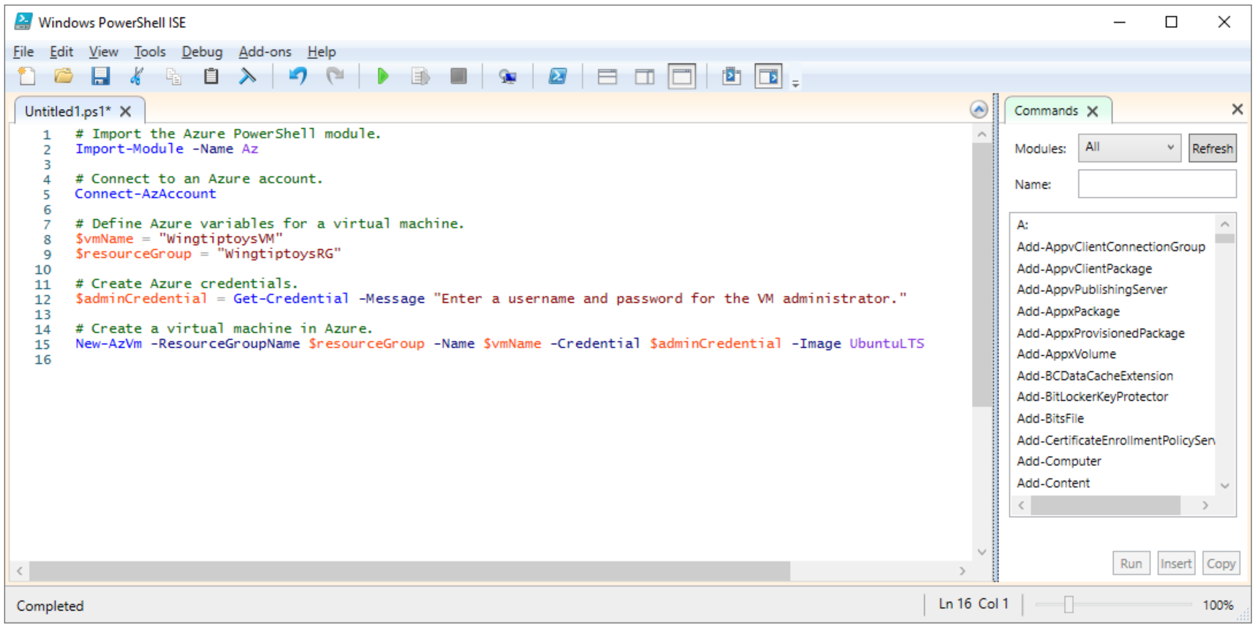
A PowerShell script is a text file containing commands and control constructs. The commands are invocations of cmdlets. The control constructs are programming features like loops, variables, parameters, comments, etc., supplied by PowerShell.

PowerShell script files have a `.ps1` file extension. You can create and save these files with any text editor.

Tip

If you're writing PowerShell scripts under Windows, you can use the Windows PowerShell Integrated Scripting Environment (ISE). This editor provides features such as syntax coloring and a list of available cmdlets.

The following screenshot shows the Windows PowerShell Integrated Scripting Environment (ISE) with a sample script to connect to Azure and create a virtual machine in Azure.



```powershell
# Import the Azure PowerShell module.
Import-Module -Name Az

# Connect to an Azure account.
Connect-AzAccount

# Define Azure variables for a virtual machine.
$vmName = "WingtiptoysVM"
$resourceGroup = "WingtiptoysRG"

# Create Azure credentials.
$adminCredential = Get-Credential -Message "Enter a username and password for the VM administrator."

# Create a virtual machine in Azure.
New-AzVm -ResourceGroupName $resourceGroup -Name $vmName -Credential $adminCredential -Image UbuntuLTS
```

Once you've written the script, execute it from the PowerShell command line by passing the name of the file preceded by a dot and a backslash:

PowerShellCopy

```
.\myScript.ps1
```

PowerShell techniques

PowerShell has many features found in typical programming languages. You can define variables, use branches and loops, capture command-line parameters, write functions, add comments, and so on. We'll need three features for our script: variables, loops, and parameters.

Variables

In the last unit, you saw that PowerShell supports variables. Use `$` to declare a variable and `=` to assign a value. For example:

PowerShellCopy

```
$loc = "East US"
$iterations = 3
```

Variables can hold objects. For example, the following definition sets the **adminCredential** variable to the object returned by the **Get-Credential** cmdlet.

PowerShellCopy

```
$adminCredential = Get-Credential
```

To obtain the value stored in a variable, use the `$` prefix and its name, as in the following:

PowerShellCopy

```
$loc = "East US"
New-AzResourceGroup -Name "MyResourceGroup" -Location $loc
```

Loops

PowerShell has several loop structures, including `For`, `Do...While`, and `For...Each`. The `For` loop is the best match for our needs because we'll execute a cmdlet a fixed number of times.

The following example shows the core syntax. The example runs for two iterations and prints the value of **i** each time. The comparison operators are written `-lt` for "less than", `-le` for "less than or equal", `-eq` for "equal", `-ne` for "not equal", etc.

PowerShellCopy

```
For ($i = 1; $i -lt 3; $i++)
{
    $i
}
```

Parameters

When you execute a script, you can pass arguments on the command line. You can provide names for each parameter to help the script extract the values. For example:

PowerShellCopy

```
.\setupEnvironment.ps1 -size 5 -location "East US"
```

Inside the script, you'll capture the values into variables. In this example, the parameters are matched by name:

PowerShellCopy

```
param([string]$location, [int]$size)
```

You can omit the names from the command line. For example:

PowerShellCopy

```
.\setupEnvironment.ps1 5 "East US"
```

Inside the script, you'll rely on position for matching when the parameters are unnamed:

PowerShellCopy

```
param([int]$size, [string]$location)
```

We could take these parameters as input and use a loop to create a set of VMs from the given parameters. We'll try that next.

The combination of PowerShell and Azure PowerShell gives you all the tools you need to automate Azure. In our CRM example, we'll be able to create multiple Linux VMs using a parameter to keep the script generic and a loop to avoid repeated code. This script allows us to execute a formerly complex operation in a single step.

Next unit: Exercise - Create and save scripts in Azure PowerShell

Exercise - Create and save scripts in Azure PowerShell

Completed100 XP

- 10 minutes

Sandbox activated! Time remaining:
46 min

You have used 1 of 10 sandboxes for today. More sandboxes will be available tomorrow.

In this unit, you'll continue with the example of a company that makes Linux admin tools. Recall that you plan to use Linux VMs to let potential customers test your software. You have a resource group ready, and now it's time to create the VMs.

Your company has paid for a booth at a large Linux trade show. You plan a demo area containing three terminals each connected to a separate Linux VM. At the end of each day, you want to delete the VMs and recreate them, so they start fresh every morning. Creating the VMs manually after work when you're tired would be error prone. You want to write a PowerShell script to automate the VM creation process.

Write a script to create virtual machines

Follow these steps in Cloud Shell on the right to write the script:

1. Switch to your home folder in Cloud Shell.

   PowerShellCopy
   ```
   cd $HOME\clouddrive
   ```

2. Create a new text file, named **ConferenceDailyReset.ps1**.

   PowerShellCopy
   ```
   touch "./ConferenceDailyReset.ps1"
   ```

3. Open the integrated editor, and select the **ConferenceDailyReset.ps1** file.

PowerShellCopy
```
code "./ConferenceDailyReset.ps1"
```
Tip

The integrated Cloud Shell also supports vim, nano, and emacs if you'd prefer to use one of those editors.

4. Start by capturing the input parameter in a variable. Add the following line to your script.

   PowerShellCopy
   ```
   param([string]$resourceGroup)
   ```
   Note

   Normally, you'd have to authenticate with Azure using your credentials using `Connect-AzAccount`, and you could do so in the script. However, in Cloud Shell environment you will already be authenticated, so this is unnecessary.

5. Prompt for a username and password for the VM's admin account and capture the result in a variable:

   PowerShellCopy
   ```
   $adminCredential = Get-Credential -Message "Enter a username and password
   for the VM administrator."
   ```

6. Create a loop that executes three times:

   PowerShellCopy
   ```
   For ($i = 1; $i -le 3; $i++)
   {

   }
   ```

7. In the loop body, create a name for each VM and store it in a variable, and output it to the console:

   PowerShellCopy
   ```
   $vmName = "ConferenceDemo" + $i
   Write-Host "Creating VM: " $vmName
   ```

8. Next, create a VM using the `$vmName` variable:

   PowerShellCopy

```
New-AzVm -ResourceGroupName $resourceGroup -Name $vmName -Credential
$adminCredential -Image Canonical:0001-com-ubuntu-server-focal:20_04-
lts:latest
```

9. Save the file. You can use the "..." menu at the top right corner of the editor. There are also common accelerator keys for *Save*, like `Ctrl-S`.

The completed script should look like the following code:

PowerShellCopy
```
param([string]$resourceGroup)

$adminCredential = Get-Credential -Message "Enter a username and password for the VM
administrator."

For ($i = 1; $i -le 3; $i++)
{
    $vmName = "ConferenceDemo" + $i
    Write-Host "Creating VM: " $vmName
    New-AzVm -ResourceGroupName $resourceGroup -Name $vmName -Credential
$adminCredential -Image Canonical:0001-com-ubuntu-server-focal:20_04-lts:latest
}
```

Run the script

1. Save the file, and close the editor using the "..." context menu on the top right of the editor.
2. Run the script.

   PowerShellCopy
   ```
   ./ConferenceDailyReset.ps1 learn-1f85e320-cee8-4f25-9023-894a7396fb3e
   ```

   The script will take several minutes to complete. When it's finished, verify it ran successfully by looking at the resources you now have in your resource group:

   PowerShellCopy
   ```
   Get-AzResource -ResourceType Microsoft.Compute/virtualMachines
   ```

You should see three VMs, each with a unique name.

You wrote a script that automated the creation of three VMs in the resource group indicated by a script parameter. The script is short and simple, but automates a process that would take a long time to complete manually with the Azure portal.

Next unit: Summary

Summary

Completed200 XP

- 3 minutes

In this module, we wrote a script to automate the creation of multiple VMs. Even though the script was relatively short, you can see the potential power when you combine loops, variables, and functions from PowerShell with cmdlets from Azure PowerShell.

Azure PowerShell is a good automation choice for admins with PowerShell experience. The combination of clean syntax and a powerful scripting language also makes it worth considering even if you're new to PowerShell. This level of automation for time-consuming and error-prone tasks should help you reduce administrative time and increase quality.

Clean up

The sandbox automatically cleans up your resources when you're finished with this module.

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 that you leave running can cost you money. You can delete resources individually or delete the resource group to delete the entire set of resources.

When you're running in your own subscription, you can use the following PowerShell cmdlet to delete the resource group (and all related resources).

PowerShellCopy
```
Remove-AzResourceGroup -Name MyResourceGroupName
```

When you're asked to confirm the delete, answer **Yes**, or you can add the `-Force` parameter to skip the prompt. The command may take several minutes to complete.

Check your knowledge

1.

True or false: The Azure portal, the Azure CLI, and Azure PowerShell offer significantly different services, so it's unlikely that all three will support the operation you need.

○

True

○

False

The three tools offer almost the same set of services. Generally, services aren't a factor in deciding which tool is best for your tasks.

2.

Suppose you're building a video-editing application that will offer online storage for user-generated video content. You'll store the videos in Azure Blobs, so you need to create an Azure storage account to contain the blobs. Once the storage account is in place, it's unlikely you would remove and recreate it because all the user videos would be deleted. Which tool is likely to offer the quickest and easiest way to create the storage account?

○

Azure portal

The portal is a good choice for one-off operations like creating a long-lived storage account. The portal gives you a GUI containing all the storage-account properties and provides tool tips to help you select the right options for your needs.

○

Azure CLI

○

Azure PowerShell

3.

What needs to be installed on your machine to let you execute Azure PowerShell cmdlets locally?

○

The Azure Cloud Shell

○

The base PowerShell product and the Az PowerShell module

You need both the base PowerShell product and the Az PowerShell module. The base product gives you the shell itself, a few core commands, and programming constructs like loops, variables, etc. The Az PowerShell module adds the cmdlets you need to work with Azure resources.

○ The Azure CLI and Azure PowerShell

Module complete: