13

Managing Azure

In this chapter, we cover the following recipes:

* Getting started using Azure with PowerShell
* Creating Azure resources
* Exploring the Azure storage account
* Creating an Azure SMB file share
* Creating an Azure website
* Creating an Azure Virtual Machine

# Introduction

Azure is Microsoft's cloud computing platform and is a competitor to Amazon's Amazon Web Services (AWS) and other public cloud providers. Azure provides you with access to a vast and constantly growing range of features. It enables any organization to move some, most, or even their entire on-premises infrastructure into the cloud.

Azure features come from three levels:

* Infrastructure as a Service (IaaS)
* Platform as a Service (PaaS)
* Software as a Service (SaaS)

IaaS is, in effect, an instant computing infrastructure that you can provision, manage, and use over the internet or via a private network connection. IaaS includes essential computing infrastructure components (servers, storage, networking, firewalls, and security), plus the physical plant you require to run these components (for example, power and air conditioning). In an IaaS environment, the servers are Azure virtual machines (effectively Hyper-V VMs) and interact with the networking, security, and storage components.

PaaS is a complete deployment environment in the cloud, including the operating system, storage, and other infrastructure. One key PaaS offering in Azure is the Azure SQL Database. Things like the OS and SQL server patching, which you would have to deal with if you deployed SQL in an IaaS environment, are all managed by Azure. The Azure SQL offering provides a (nearly) complete SQL service, all managed by Azure. You can do a few things in a full SQL server implementation that the SQL PaaS offering does not provide. These are generally actions that only the platform owner is allowed to perform. For example, with SQL running inside an IaaS Azure VM, you cannot use SQL database mirroring. If you need SQL services that the Azure SQL offering does not provide, you can create a VM and install and manage a SQL Server in the VM.

With SaaS, you just use an application that the vendor has placed in the cloud. A key example of SaaS is Office 365 (O365), which bundles Exchange Online, SharePoint Online, Teams, and more. Strictly speaking, Office 365 is not an Azure offering – you purchase it directly from either the Office 365 website or via a Microsoft Partner. In terms of purchase, Office 365 is a single offering with many different plans (combinations of services that include a downloadable version of the Office applications, such as Word and Excel). Using PowerShell to manage Office 365, each of the included applications has a unique operational approach. With Exchange Online, for example, you use PowerShell Implicit Remoting to manage the exchange component of your Office 365 subscription. Other commands run locally but make use of REST API calls to Azure across the Internet.

To provide authentication for software running within Azure and other SaaS applications, you can use the Azure Active Directory (AAD). With AAD, you can create a cloud-only directory or synchronize the AAD with your on-premises Active Directory. AAD can also be used to provide authentication for a range of other third-party SaaS applications. Full details on managing AAD and Office 365 are outside the scope of this chapter.

We begin this chapter with the first recipe, Getting started using Azure with PowerShell, in which you create an environment that allows you to manage Azure and the Office 365 SaaS components. This recipe also shows you how to download the cmdlets you need.

The Creating Azure resourcesrecipe guides you through creating a few of the core resources you need to create and manage other Azure resources. These include a resource group and a storage account. You create all Azure resources within a resource group.

You create and store any required storage, such as VHD files for an Azure VM, in a storage group. While the recipes in this chapter use a single resource group and a single storage account for simplicity, in large-scale Azure deployments, you may require multiple instances of these resources.

In the Exploring the Azure storage account recipe, we look at setting up Azure storage using the Azure storage account we created earlier. The Creating an Azure SMB file share recipe shows you how you can create an SMB3 file share that you can access from client applications across the internet. Instead of having an application point to an on-premises file share, you can now host the share in Azure. This feature might be useful if you use Azure IaaS VM to host an application that utilizes a shared folder for its data. You could also use it as a file share in the cloud.

The Creating an Azure websiterecipe shows you how you can set up a simple website. The recipe creates a free Azure App Plan which supports an IIS website. With this app plan, you can set up a simple website, say for a short-term marketing campaign. You can scale this to deliver internet-scale websites that you can have Azure scale dynamically according to load.

The final recipe, Creating an Azure Virtual Machine, examines how to create an Azure VM and access it via RDP. Although Azure uses a Hyper-V variant to run its VMs, you cannot use the Hyper-V cmdlets you saw in Chapter 11 to manage Azure VMs. Azure has its own set of commands, as you see in this recipe.

This chapter is only a taster for using Azure with PowerShell. There is so much more you can do that could not fit into this chapter.

# Getting started using Azure with PowerShell

There are two fundamental things you need to do before you can start managing Azure features using PowerShell. The first is to obtain an Azure subscription. The second is to obtain the cmdlets you use to access Azure and Office 365's features.

Azure is a commercial service – each feature you use potentially has a real-world cost attached, which Microsoft bases on your usage of Azure resources. For example, with an Azure VM, you would pay to have the VM running, and there are additional charges for the VM's virtual disk storage and network traffic in and out of your VM.

The charges for Office 365 and other SaaS offerings, on the other hand, are user-based – a given user can use lots of emails, for example, without incurring any additional charges. For details on costs for Azure, see https://azure.microsoft.com/pricing/, and for details on Microsoft 365 charges, see https://www.microsoft.com/microsoft-365/buy/compare-all-microsoft-365-products.

There are many ways you can get an Azure subscription, including via a Visual Studio subscription (https://visualstudio.microsoft.com/vs/pricing/), via an Action Pack subscription (https://docs.microsoft.com/partner-center/mpn-get-action-pack), or outright purchase on a pay as you go basis.

Microsoft also provides a one-month free trial subscription that helps you test out the recipes in this chapter. The trial subscription provides you with full access to Azure features up to a financial limit, which is $200 US dollars or similar in other currencies at the time of writing. These limits may have changed by the time you read this book. The trial subscription should be sufficient to enable you to learn how to use PowerShell with Azure.

To get a trial subscription, navigate to https://azure.microsoft.com/free/, and fill in the forms. Note that a free trial requires you to submit a credit card number. There is no charge for the subscription; the credit card number is used only for identity verification – and it keeps the lawyers happier.

If you take out an Azure trial and wish to keep your Azure resources running after the trial expires, you have to move it to a pay as you go subscription. You should receive an email shortly before the trial expires to transition it.

To use PowerShell with Azure's various features, you need to obtain cmdlets that Microsoft does not provide in Windows Server 2022/Windows 10, Windows PowerShell 5.0/5.1, or PowerShell 7. You get the relevant modules from the PowerShell Gallery. You use the cmdlets in the PowerShellGet module to find and download the necessary modules.

As a word of warning, these cmdlets change regularly. For the most part, these changes add functions and fix bugs, but you may find that new versions of a module bring breaking changes that could affect your scripts. The Azure team provide reasonable notice of breaking changes, and you usually have plenty of notice and flexibility over when you deploy any updates.

## Getting ready

You run this recipe on SRV1, a domain-joined server.

## How to do it...

1. Finding core Azure module on the PS Gallery

Find-Module -Name Az |

  Format-Table -Wrap -Autosize

1. Installing Az module

Install-Module -Name Az -Force

1. Discovering Azure modules and how many cmdlets each contains

$HT = @{ Label ='Cmdlets'

         Expression = {(Get-Command -module $\_.name).count}}

Get-Module Az\* -ListAvailable |

    Sort-Object {(Get-command -Module $\_.Name).Count} -Descending |

       Format-Table -Property Name,Version,Author,$HT -AutoSize

1. Finding Azure AD cmdlets

Find-Module AzureAD |

  Format-Table -Property Name, Version, Author -AutoSize -Wrap

1. Installing the Azure AD module

Install-Module -Name AzureAD -Force

1. Discovering the Azure AD Module

$FTHT = @{

    Property = 'Name', 'Version', 'Author', 'Description'

    AutoSize = $true

    Wrap     = $true

}

Get-Module -Name AzureAD -ListAvailable |

  Format-Table @FTHT

1. Logging into Azure

$CredAZ  = Get-Credential     # Enter your Azure Credential details

$Account = Connect-AzAccount -Credential $CredAZ

$Account

1. Getting Azure account name

$AccountM = $Account.Context.Account.Id

"Azure Account   : $AccountM"

1. Viewing Azure subscription

$SubID = $ACCOUNT.Context.Subscription.Id

Get-AzSubscription -SubscriptionId $SubId |

  Format-List -Property \*

1. Counting Azure locations

$AZL = Get-AzLocation

$LOC = $AZL | Sort-Object Location

"Azure locations:  [{0}]" -f $LOC.Count

1. Viewing Azure locations

$LOC |

  Format-Table Location, DisplayName

1. Getting Azure environments

Get-AzEnvironment |

    Format-Table -Property name, ManagementPortalURL

## How it works...

In step 1, you use the Find-Module command to find the Az module on the PS Gallery. The output of this step should resemble this:

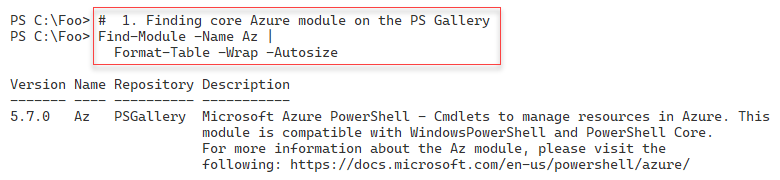


Figure 13.1: Finding the AZ module

**Insert image B42024\_13\_01.png**

In step 2, you install the Az module on SRV1. This step installs all of the individual sub-modules that you can use to manage Azure. Although this step produces no console output, you may see popup progress indicators as PowerShell installs the individual modules.

In step 3, you discover the individual modules and how many cmdlets each one contains. The output looks like this:

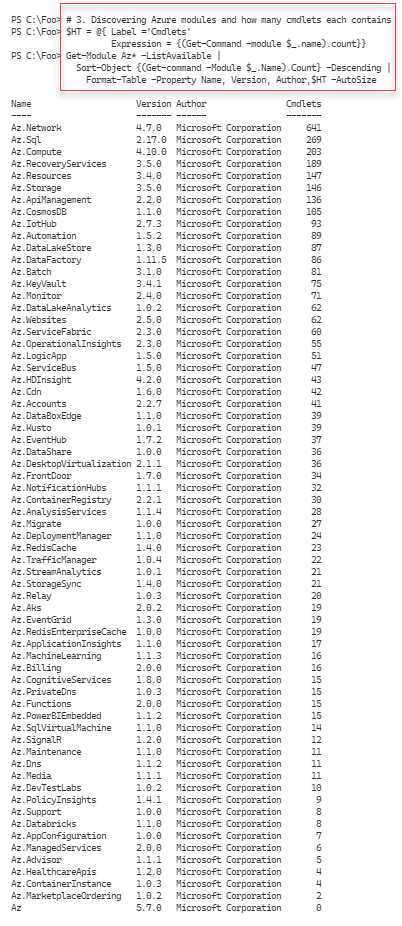
­­

Figure 13.2: Viewing Azure modules

**Insert image B42024\_13\_02.png**

In step 4, you find the Azure AD module on the PS Gallery. The output of this step is as follows:

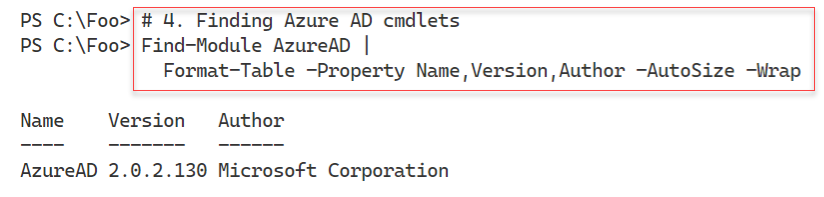


Figure 13.3: Finding the Azure AD module

**Insert image B42024\_13\_03.png**

In step 5, you install the Azure AD module, which generates no output to the console. In step 6, you look at more information about this module, with output like this:

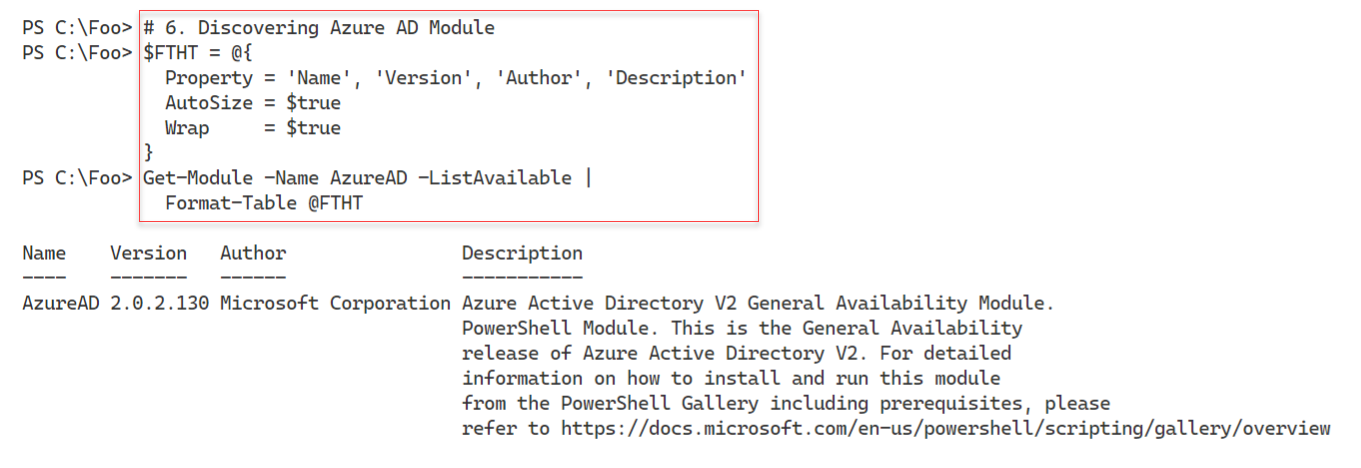


Figure 13.4: Viewing more information on the Azure AD module

**Insert image B42024\_13\_05.png**

In step 7, you use Connect-AzAccount to log in to Azure. The output of this command looks like this:

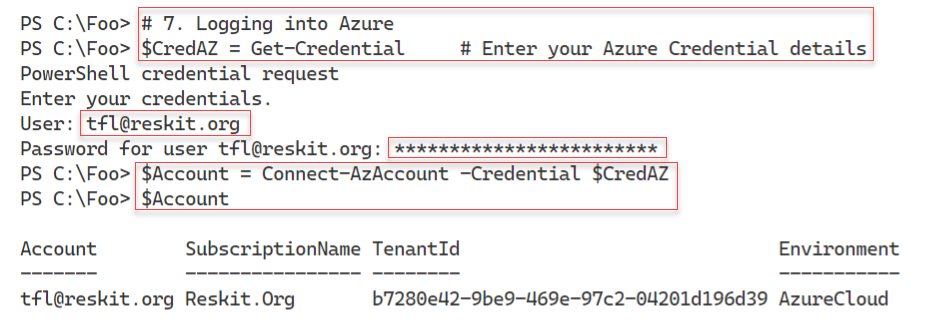


Figure 13.5: Logging into Azure

**Insert image B42024\_13\_06.png**

Once you have logged in successfully, in step 8, you view the account name for this subscription, with output like this:

Figure 13.6: Viewing the Azure account name

**Insert image B42024\_13\_07.png**

In step 9, you view details of the Azure subscription, with output like this:



Figure 13.7: Viewing details of the Azure subscription

**Insert image B42024\_13\_08.png**

In step 10, you use the Get-AzLocation cmdlet to discover and count the Azure locations of Azure's worldwide collection of data centers. The output, at least at the time of writing, looks like this:

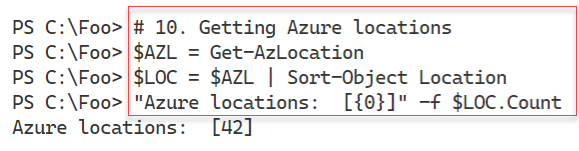


Figure 13.8: Counting Azure global locations

**Insert image B42024\_13\_09.png**

In step 11, you view the Azure locations, with output like this:



Figure 13.9: Viewing Azure global locations

**Insert image B42024\_13\_10.png**

Microsoft has created several independent Azure environments, each with an independent management portal and sets of services. The Az cmdlets work with any environment you can access, although some services may not exist in all environments. In step 12, you view the current Azure environments, with output like this:

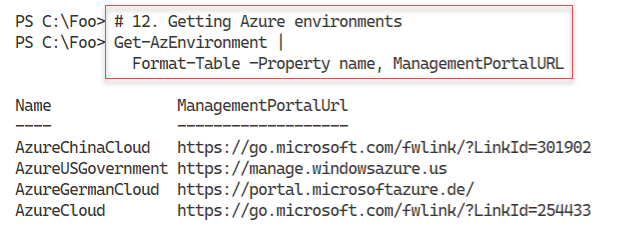


Figure 13.10: Viewing Azure cloud environments

**Insert image B42024\_13\_11.png**

## There's more...

In step 3, you enumerate the Azure modules loaded in the previous step, and for each one, you display the name and the number of commands in the module. Some of the modules are large and contain a wealth of commands; others are small. While the AZ.Network module contains over 600 commands, the Az.MarketPlaceOrdering module contains just 2 commands. By the time you read this book, the number of modules and the commands in each has probably changed. The core modules and cmdlets you use in this chapter should not have changed, but you never know.

In step 4 through step 6, you find, download, and examine the Azure AD module. You use these cmdlets to manage the Azure AD service. This chapter does not cover using these cmdlets.

In step 7, you log in to Azure. In production, using a simple user and password, even a long one, is not secure. A best practice is to use multi-factor authentication (MFA). In which case, you could use Connect-AzAccount without parameters and log in via Microsoft's GUI. This recipe shows how to log in to Azure using only a credential object.

In step 10 and step 11, you count and view the Azure locations around the world. Each Azure location is one, and sometimes more than one, physical data center containing a large number of compute and data servers that deliver Azure services. Not every Azure location delivers all Azure service offerings, especially as Microsoft rolls out new features. Microsoft is constantly investing in new locations, so there may be even more locations by the time you are reading this chapter.

Microsoft maintains several separate independently-run and operated Azure environments. In addition to the public Azure Cloud environment, Microsoft provides three additional parallel environments: China, Germany, and the US government. In step 12, you view the publicly acknowledged environments – there may be more (or not).

# Creating Azure resources

In the previous recipe, you created and used the basic Azure management environment by downloading the key modules, logging in to Azure, and looking at the environment. In this recipe, you create certain key Azure assets, including a resource group, a storage account, and tags.

You create all Azure resources within a resource group. A resource group is a grouping of Azure resources. Resource groups are fundamental to managing Azure resources.

Any storage you create within Azure resides in a storage account, a fundamental building block within Azure. You create a storage account within one of the Azure locations you saw in the Getting started using Azure with PowerShell recipe. When you create your storage account, you specify the level of resiliency and durability that you wish Azure to provide. There are several levels of replication provided within Azure. These levels provide for multiple copies of the data, which Microsoft replicates automatically. You can have Azure store these replicas in the local Azure data region or other Azure locations. These data replicas provide extra resilience and an increased likelihood of recovery should the unthinkable happen and an entire data center somehow fails in a catastrophic way.

You can provision a storage account as either standard or premium. A standard storage account allows you to store any data (as you see in the Exploring the Azure storage account recipe). A premium storage account provides extra features but at a cost.

Tags are name/value pairs that allow you to organize your resources within your subscription. For more details on how you can use tags to organize your Azure resources, see https://docs.microsoft.com/azure/azure-resource-manager/resource-group-using-tags/.

## Getting ready

You run this recipe on SRV1, on which you have installed the AZ module(s).

## How to do it...

1. Setting key variables

$Locname    = 'uksouth'     # Location name

$RgName     = 'packt\_rg'    # Resource group we are using

$SAName     = 'packt42sa'   # A unique storage account name

1. Logging into your Azure account

$CredAZ  = Get-Credential

$Account = Connect-AzAccount

1. Creating a resource group and tagging it

$RGTag  = [Ordered] @{Publisher='Packt'

                      Author='Thomas Lee'}

$RGHT = @{

    Name     = $RgName

    Location = $Locname

    Tag      = $RGTag

}

$RG = New-AzResourceGroup @RGHT

1. Viewing the resource group with tags

Get-AzResourceGroup -Name $RGName |

    Format-List -Property \*

1. Testing to see if an SA name is taken

Get-AzStorageAccountNameAvailability $SAName

In step 5, you ensure that the storage account name is unique, which means the name is available for you to use. If the name is not unique, you should change the value of $SAName and retry the creation of the storage account before proceeding.

1. Creating a new storage account

$SAHT = @{

  Name              = $SAName

  SkuName           = 'Standard\_LRS'

  ResourceGroupName = $RgName

  Tag               = $RGTag

  Location          = $Locname

}

New-AzStorageAccount @SAHT

1. Getting an overview of the storage account in this resource group

$SA = Get-AzStorageAccount -ResourceGroupName $RgName

$SA |

  Format-List -Property \*

1. Getting primary endpoints for the storage account

$SA.PrimaryEndpoints

1. Reviewing the SKU

$SA.Sku

1. Viewing the storage account's context property

$SA.Context

## How it works...

In step 1, you define key variables, which produces no console output. In step 2, you log in to your Azure account, entering your credentials via the GUI. In the Getting started using Azure with PowerShell recipe, you logged in using your username and password. In this recipe, you use the GUI to sign in, using MFA if you have that configured. When you complete the login process, you should see this in the default browser:

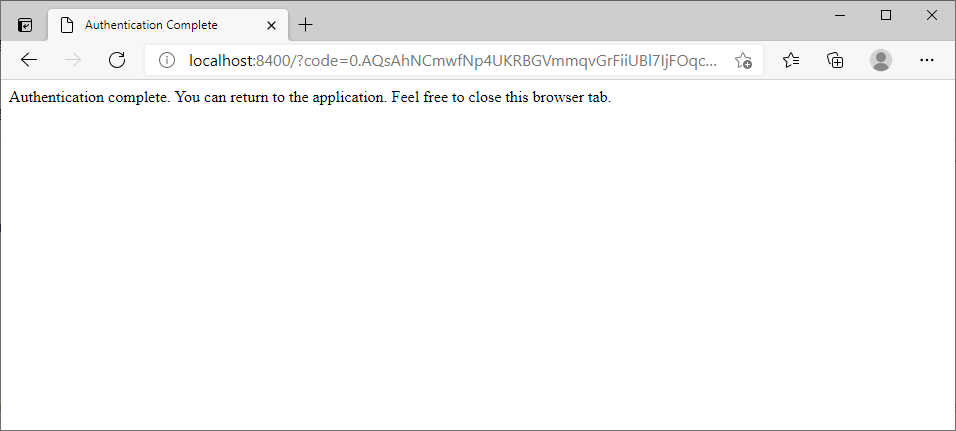


Figure 13.11: Logging in to Azure

**Insert image B42024\_13\_12.png**

In step 3, you create a new Azure resource group with two tags. This step creates no output to the console.

In step 4, you view the newly created resource group, with output like this:

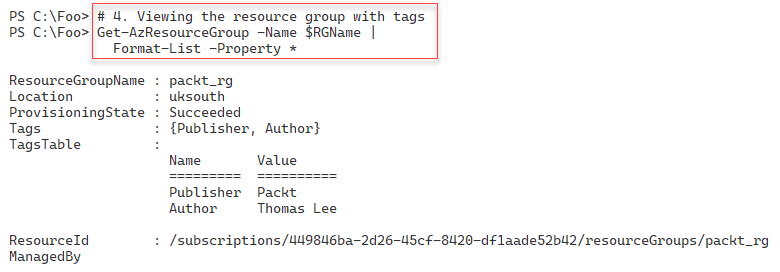
0

Figure 13.12: Viewing new Azure resource group

**Insert image B42024\_13\_13.png**

With Azure, a storage account name must be globally unique. In step 5, you use the Get‑AzureAccountNameAvailability cmdlet to test whether the name you want to use for the storage account is available. The output looks like this:

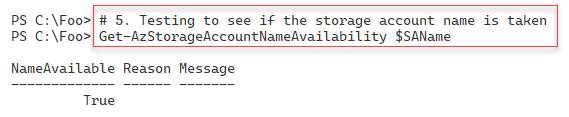


Figure 13.13: Testing the availability of the storage account name

**Insert image B42024\_13\_14.png**

In step 6, you create a new storage account, which generates the following output:

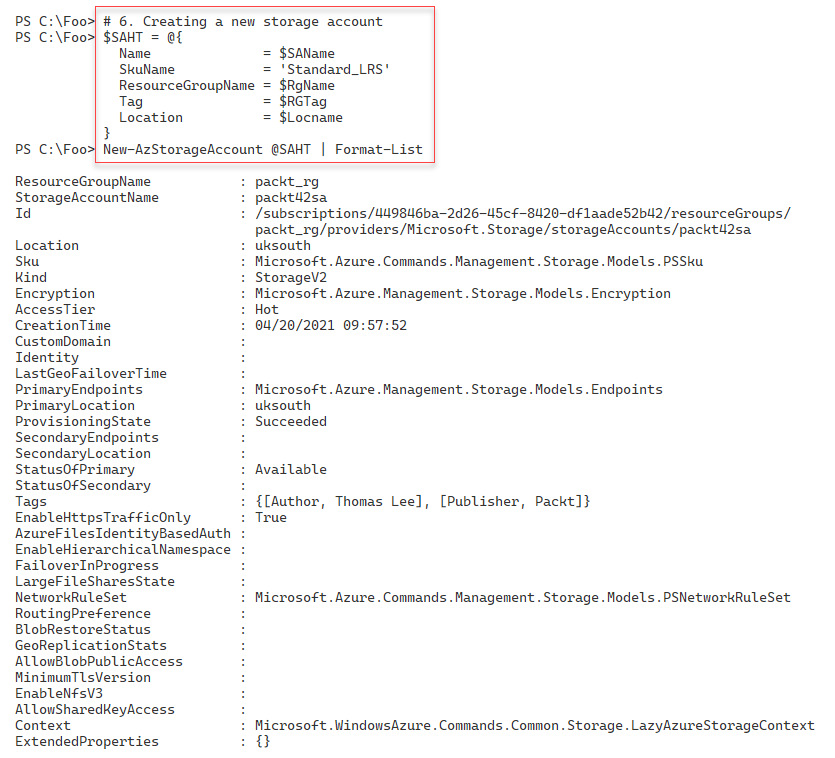


Figure 13.14: Creating a new storage account

**Insert image B42024\_13\_15.png**

In step 7, you retrieve and view the storage account details, with output like this:

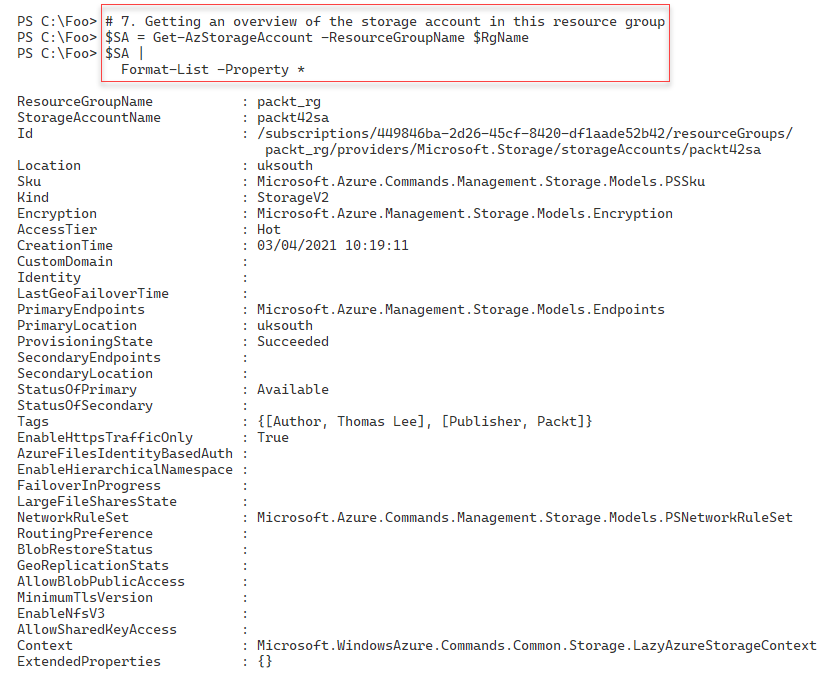


Figure 13.15: Viewing your new storage account

**Insert image B42024\_13\_16.png**

Each of the Azure storage endpoint types (such as a blob, file, or queue) are prefixed with your storage account name followed by a standardized primary endpoint suffix. You view these endpoints in step 8, with output like this:

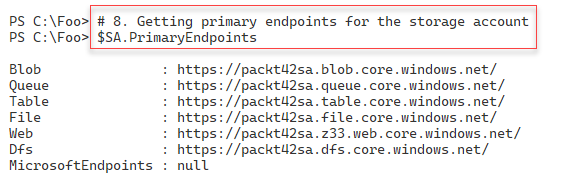


Figure 13.16: Viewing primary Azure storage endpoint suffixes

**Insert image B42024\_13\_17.png**

In step 9, you view the Shipping Kit Unit (SKU) for this storage group, with output like this:

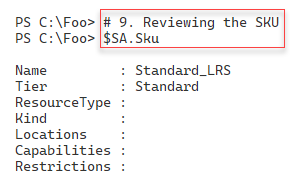


Figure 13.17: Viewing Azure storage group's SKU

**Insert image B42024\_13\_18.png**

In the final step in this recipe, step 10, you view the storage account's context property, with output like this:

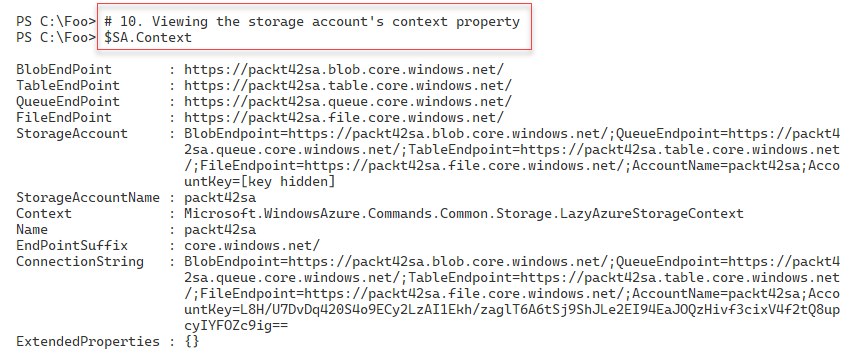


Figure 13.18: Viewing the storage group's context property

**Insert image B42024\_13\_19.png**

## There's more...

In step 1, you define variables that hold important values you use later in the recipe. If you are performing this recipe, you need to change the values you use in this step to reflect your unique names for your resource group and storage account.

In step 5, you test to check whether the proposed storage account name is available. Someone else may have used the storage group’s name, meaning it is not available to you. If your choice of storage account name has already been taken, you need to adjust the name to be unique. You could use a GUID for the account name, which you create using the New-Guid command as one alternative, although that may be less easy to type.

In step 9, you examined the storage account's SKU. The SKU represents the specific resource tier in which you have created the storage account. Each tier, each SKU, has different capabilities and different prices.

In step 10, you examine the storage account's context. Azure uses context objects to hold subscription and authentication information. For more information on context objects in Azure, see https://docs.microsoft.com/powershell/azure/context-persistence.

# Exploring the Azure storage account

Many Azure features use Azure Storage. When you create an Azure VM, for example, you store the VHD file in Azure Storage. Azure storage accounts can hold a variety of data, with different mechanisms for managing each data type. Additionally, the storage account provides both scalability and data durability and resiliency.

Azure Storage manages five distinct types of data:

* Binary large object (blob)
* Table
* Queue
* File
* Disk

A blob is unstructured data that you store in Azure. Blob storage can hold any type of data in any form – MP4 movies, ISO images, VHD drives, JPG files, and more. Individual blobs reside within blob containers, equivalent to file store folders, but with very limited nesting capability.

Blobs come in three types: block blobs, append blobs, and page blobs. Block blobs are physically optimized for storing documents to the cloud and for streaming applications. Append blobs are optimized for append operations and are useful for logging. Page blobs are optimized for read/write operations – Azure VHDs, for example, are always of the page blob type. For more information about blob types, take a look at https://docs.microsoft.com/azure/storage/blobs/storage-blobs-introduction.

An Azure table is a non-relational storage system that utilizes key-value pairs. You can use Azure tables for storing unstructured or semi-structured data. Azure tables are not the same as SQL tables, which hold highly normalized data. An Azure table consists of a grouping of entities. See https://azure.microsoft.com/services/storage/tables/ for more information about Azure table storage.

An Azure queue is a durable message queuing feature that you can use to implement massively scalable applications. With message queues, one part of an application can write a transaction to the queue for another part to process. A queue enables you to decouple application components for independent scaling and to provide greater resiliency. For more details on Azure queues, see https://azure.microsoft.com/services/storage/queues/.

The Azure Files feature provides cross-platform file storage that you can access using SMB. Azure Files allow you to create and use SMB file shares in the cloud and access them, just like you would access on-premises SMB shares. Azure Files support SMB 3.0, making it simple and easy to migrate legacy applications that rely on file shares. For more information on Azure Files, see https://azure.microsoft.com/services/storage/files/.

Azure's disk storage provides persistent, highly secure disk options, particularly for Azure VMs. Azure has designed its disk storage service for low latency and high throughput. You can create disks using both traditional spinning disks as well as SSD disks. SSD disks provide better I/O performance for I/O intensive applications but are generally more expensive. For more details on Azure disk storage, see https://azure.microsoft.com/services/storage/disks/.

Azure's storage features continue to evolve, with more options available as time goes by. For more details on Azure storage as a whole, see https://docs.microsoft.com/azure/storage/common/storage-introduction.

As we noted earlier, Azure names storage accounts based on a global naming scheme that uses HTTPS URLs. The Azure REST API relies on these URLs to manage the Azure resources in your resource groups. The HTTP URL contains your storage account followed by the Azure data type and then ".corewindows.net", like this:

https://<storageaccountname>.<datatype>.core.windows.net/...

## Getting ready

This recipe uses SRV1, which you have used to access Azure so far in this chapter.

## How to do it...

1. Setting key variables

$Locname    = 'uksouth'         # location name

$RgName     = 'packt\_rg'        # resource group we are using

$SAName     = 'packt42sa'       # storage account name

$CName      = 'packtcontainer'  # a blob container name

$CName2     = 'packtcontainer2' # a second blob container name

1. Logging into your Azure account

$Account = Connect-AzAccount

1. Getting and displaying the storage account key

$SAKHT = @{

    Name              = $SAName

    ResourceGroupName = $RgName

}

$Sak = Get-AzStorageAccountKey  @SAKHT

$Sak

1. Extracting the first key's "password"

$Key = ($Sak | Select-Object -First 1).Value

1. Getting the storage account context which encapsulates credentials for the storage account

$SCHT = @{

    StorageAccountName = $SAName

    StorageAccountKey = $Key

}

$SACon = New-AzStorageContext @SCHT

$SACon

1. Creating 2 blob containers

$CHT = @{

  Context    = $SACon

  Permission = 'Blob'

}

New-AzStorageContainer -Name $CName @CHT

New-AzStorageContainer -Name $CName2 @CHT

1. Viewing blob containers

Get-AzStorageContainer -Context $SACon |

    Select-Object -ExpandProperty CloudBlobContainer

1. Creating a blob

'This is a small Azure blob!!' | Out-File .\azurefile.txt

$BHT = @{

    Context = $SACon

    File = '.\azurefile.txt'

    Container = $CName

}

$Blob = Set-AzStorageBlobContent  @BHT

$Blob

1. Constructing and displaying the blob name

$BlobUrl = "$($Blob.Context.BlobEndPoint)$CName/$($Blob.name)"

$BlobUrl

1. Downloading and viewing the blob

$OutFile = 'C:\Foo\Test.Txt'

Start-BitsTransfer -Source $BlobUrl -Destination $OutFile

Get-Content -Path $OutFile

## How it works...

In step 1, you set values for variables you use later in this recipe, creating no console output. If you discover, in the Getting started using Azure with PowerShell recipe, that you cannot acquire the storage account name, you need to create the storage with a different name. Don’t forget to change the name in this step (and in subsequent recipes in this chapter.

In step 2, you connect to your Azure account using the GUI. As with other recipes, once you finish, you should see this in the default browser:

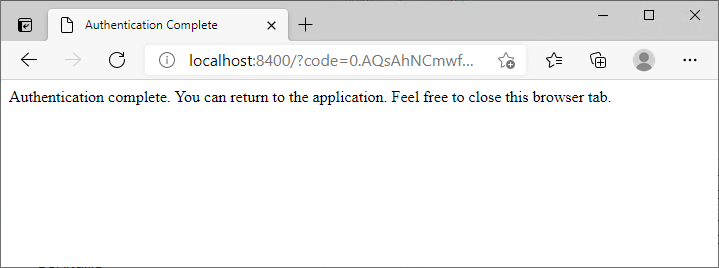


Figure 13.19: Logging into Azure

**Insert image B42024\_13\_20.png**

In step 3, you use the Get-AzStorageAccountKey cmdlet to retrieve your storage account's key data, with output like this:

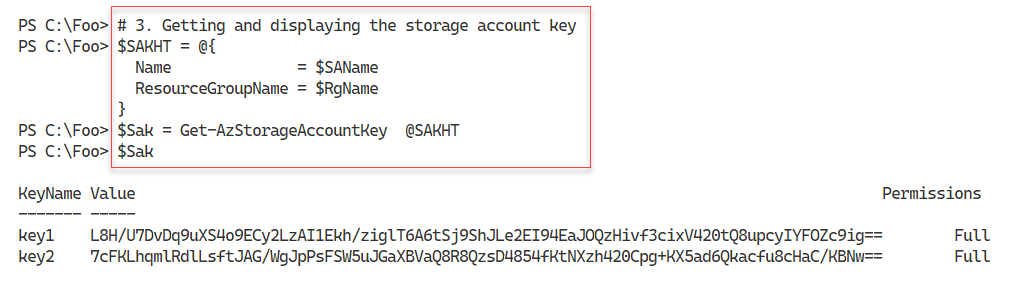


Figure 13.20: Getting and viewing your storage account key

**Insert image B42024\_13\_21.png**

In step 4, you extract the first key to later serve as a password. This step creates no output to the console. In step 5, you view the storage account's context, which looks like this:

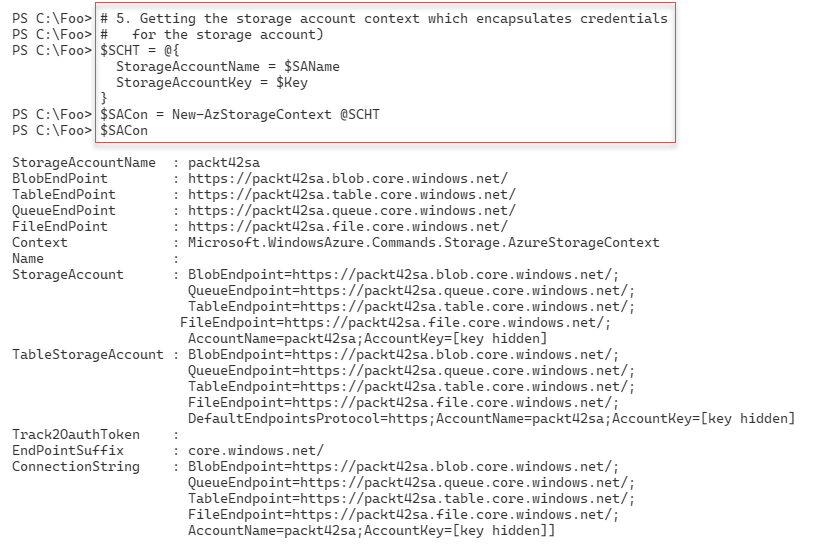


Figure 13.21: Getting and viewing your storage account's context object

**Insert image B42024\_13\_22.png**

In step 6, you create two new Azure storage blob containers, with output like this:

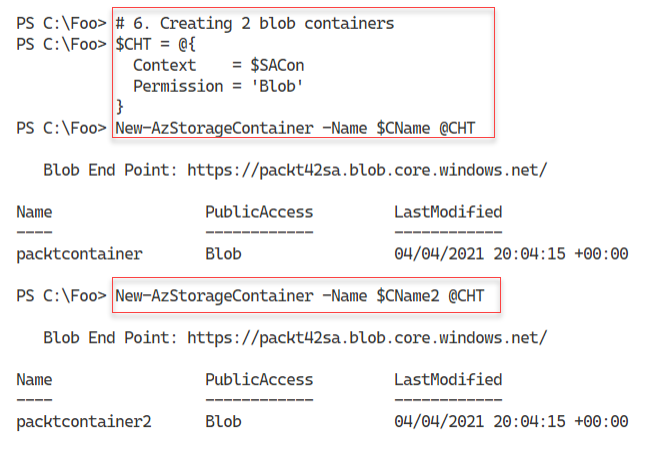


Figure 13.22: Creating two Azure storage blob containers

**Insert image B42024\_13\_23.png**

In step 7, you use the Get-AzStorageContainer cmdlet to get details about the Azure blob containers within the storage account, with output like this:

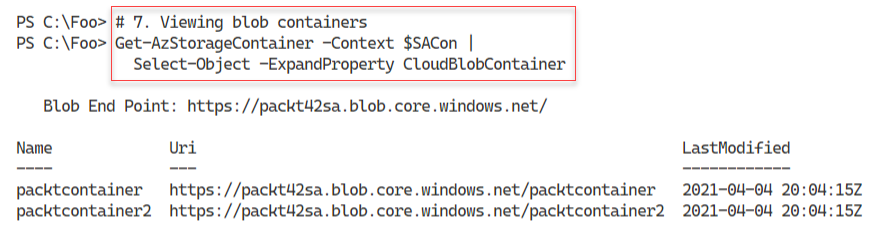


Figure 13.23: Viewing Azure storage blob containers

**Insert image B42024\_13\_24.png**

In step 8, you create a new Azure blob with output like this:

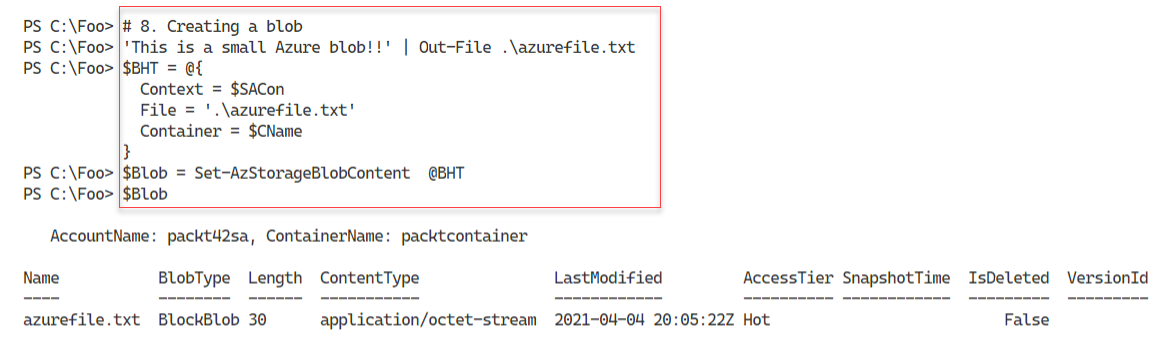


Figure 13.24: Creating an Azure blob

**Insert image B42024\_13\_25.png**

In step 9, you display the blob's URL, with output like this:

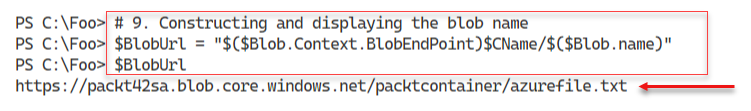


Figure 13.25: Displaying the blob's URL

**Insert image B42024\_13\_26.png**

In step 10, you use the URL of the new blob to download the file and view it, like this:

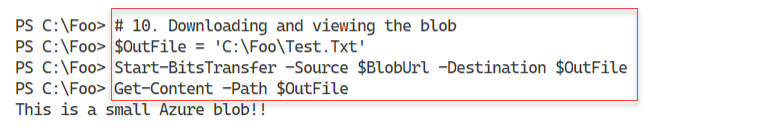


Figure 13.26: Viewing the blob in your default browser

**Insert image B42024\_13\_27.png**

## There's more...

In step 3, you display your storage account's account key, which contains two passwords for your account.

In step 10, you use the Background Intelligent Transfer Service (BITS) to download the blob to a local file. You could have also used your default browser and navigated to the blob's URL, or used other means to download the data you stored in Azure.

# Creating an Azure SMB file share

Azure provides you with the ability to create SMB shares with an Azure storage account. These SMB shares act the same as the local on-premises SMB shares you used in Chapter 10, Managing Shared Data. The key difference is how you create them and the credentials that you use to access the shares.

Before an SMB client can access data held in an SMB share, the SMB client needs to authenticate with the SMB server. With Windows-based shares, you either specify a user credential object (user ID and password) or, in a domain environment, the SMB client utilizes Kerberos to authenticate. With Azure, you use the storage account name as the user ID and the storage account key as the password.

The storage account key contains two properties, imaginatively named key1 and key2. The values of these two properties are valid passwords for Azure SMB file shares. Having two keys enables you to do regular key rotation. If your application uses the value of key1, you can reconfigure your application to use the key2 value as the share's password and then regenerate the key1 value. At a later time, you can repeat this, changing the application to use the now updated key1's value and then regenerating key2. Key rotation provides you with an immediate key update where you need it. Armed with either key value, you can create SMB shares that you can directly access across the internet.

Azure SMB file shares differ from Azure blobs in terms of how you access the data. You access a blob via HTTP, whereas you access an Azure file share via the standard SMB 3 networking protocol that you used in, for example, Chapter 10, Managing Shared Data. Also, with Azure blob storage, you can only have a single level of a folder (that is, the blob container). With Azure Files, on the other hand, you can have as many folders as you need.

When using Azure SMB shares, the storage account key is the password for the share, and the storage account name is the user ID. As with all credentials, you should exercise caution when including the account key in your PowerShell scripts.

In this recipe, you use the resource group and storage account we created earlier (in the Create Azure resources recipe). This recipe also checks to ensure that these exist and creates them if they are not available.

## Getting ready

This recipe uses SRV1, which you have used to access Azure so far in this chapter.

## How to do it...

1. Defining variables

$Locname   = 'uksouth'      # location name

$RgName    = 'packt\_rg'     # resource group we are using

$SAName    = 'packt42sa'    # storage account name

$ShareName = 'packtshare'   # must be lower case!

1. Logging in to your Azure account

$Account = Connect-AzAccount

1. Getting storage account, account key, and context

$SA = Get-AzStorageAccount -ResourceGroupName $Rgname

$SAKHT = @{

    Name              = $SAName

    ResourceGroupName = $RgName

}

$Sak = Get-AzStorageAccountKey @SAKHT

$Key = ($Sak | Select-Object -First 1).Value

$SCHT = @{

    StorageAccountName = $SAName

    StorageAccountKey  = $Key

}

$SACon = New-AzStorageContext @SCHT

1. Adding credentials to the local credentials store

$T = "$SAName.file.core.windows.net"

cmdkey /add:$T /user:"AZURE\$SAName" /pass:$Key

1. Creating an Azure share

New-AzStorageShare -Name $ShareName -Context $SACon

1. Checking that the share is reachable

$TNCHT = @{

  ComputerName = "$SAName.file.core.windows.net"

  Port         = 445

}

Test-NetConnection @TNCHT

1. Mounting the share as M:

$Mount = 'M:'

$Rshare = "\\$SaName.file.core.windows.net\$ShareName"

$SMHT = @{

    LocalPath  = $Mount

    RemotePath = $Rshare

    UserName   = $SAName

    Password   = $Key

}

New-SmbMapping @SMHT

1. Viewing the share in Azure

Get-AzStorageShare -Context $SACon  |

    Format-List -Property \*

1. Viewing local SMB mappings

Get-SmbMapping -LocalPath M:

1. Creating a file in the share

New-Item -Path M:\Foo -ItemType Directory | Out-Null

'Azure and PowerShell 7 Rock!!!' |

   Out-File -FilePath M:\Foo\Recipe.Txt

1. Retrieving details about the share contents

Get-ChildItem -Path M:\ -Recurse |

    Format-Table -Property FullName, Mode, Length

1. Getting the content from the file

Get-Content -Path M:\Foo\Recipe.txt

1. Cleaning up –  removing data from the share

Remove-Item -Path M: -Force -Recurse -ErrorAction SilentlyContinue

1. Cleaning up – removing SMB mapping

Get-SmbMapping -LocalPath M: |

  Remove-SmbMapping -Force

1. Removing the Azure share

Get-AzStorageShare -Name $ShareName -Context $SACon |

  Remove-AzStorageShare

## How it works...

With step 1, you define variables to hold important Azure names you use in this recipe, which generates no output. In step 2, you log in using the Azure GUI and MFA. Once you have logged in successfully, you see the following:

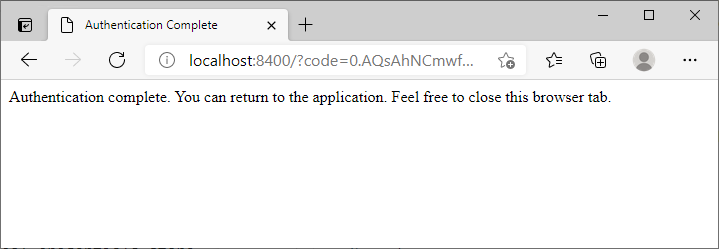


Figure 13.27: Logging in to your Azure Account

**Insert image B42024\_13\_28.png**

In step 3, you retrieve the storage account, account key, and context, which generates no output to the console. To simplify your use of SMB shares, in step 4, you add credentials in the local credential store. Windows uses these credentials when connecting to the share. The output of this step is as follows:

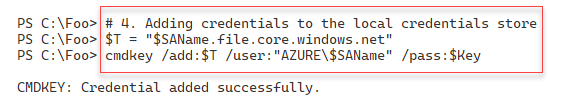


Figure 13.28: Storing credentials in the local Windows credential store

**Insert image B42024\_13\_29.png**

In step 5, you create an Azure SMB file share, with output like this:

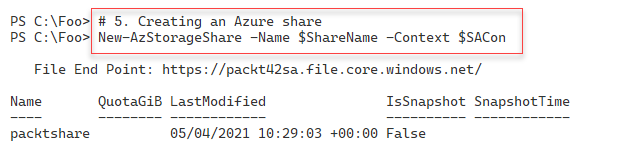


Figure 13.29: Creating an Azure SMB file share

**Insert image B42024\_13\_30.png**

With step 6, you use Test-NetConnection to verify you can reach the Azure storage host that holds your newly created SMB share, with output like this:

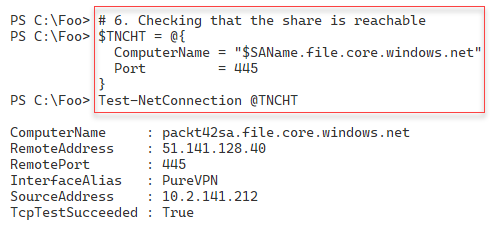


Figure 13.30: Testing connection to Azure file share

**Insert image B42024\_13\_31.png**

In step 7, you create an SMB drive mapping, mapping the local M: drive to your Azure file share. This step produces output like this:

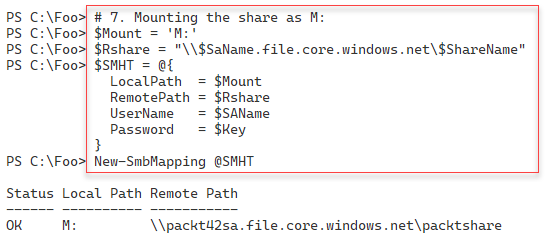


Figure 13.31: Creating a new SMB drive mapping

**Insert image B42024\_13\_32.png**

With step 8, you use the Get-AzStorageShare cmdlet to view the share's properties from within Azure. The output from this step looks like this:

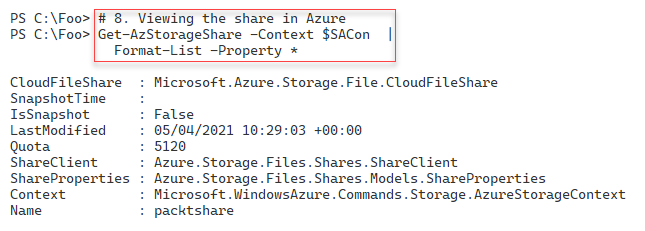


Figure 13.32: Viewing Azure file share

**Insert image B42024\_13\_33.png**

In step 9, you view the local SMB mapping, with output like this:

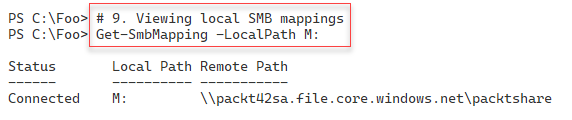


Figure 13.33: Viewing the local SMB share mapping

**Insert image B42024\_13\_34.png**

In step 10, you create a new file in the Azure file share, generating no output. In step 11, you view details of the share contents, including the new file you just created. The output of this step looks like this:

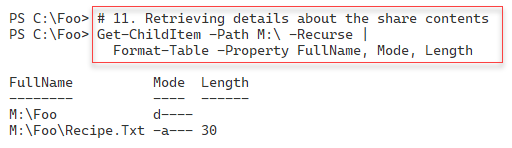


Figure 13.34: Retrieving details about the share contents

**Insert image B42024\_13\_35.png**

In step 12, you examine the contents of the file you retrieved from Azure, with output like this:

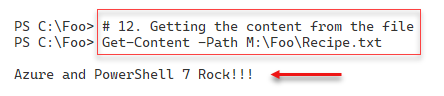


Figure 13.35: Getting the content from the file

**Insert image B42024\_13\_35a.png**

In step 13, you begin the cleanup process by removing all the data in the SMB share. In step 14, you remove the local SMB mapping. In step 15, you remove the share from Azure. These final three steps produce no output.

## There's more...

In step 2, you log in to your Azure account. If you have just completed working with any of the Azure recipes in this chapter, Azure may still have you logged in. In that case, you can skip this step.

In step 5, you create the Azure file share, which you then view in step 6. Depending on your internet connection and your Internet router, you may have issues here. Some small business routers, by default, block SMB traffic outbound. You may be able to reconfigure the router. If not, you can always use a VPN. You could use one of the popular Internet VPN suppliers and use any VPN into the Internet. Once you have your VPN connection established and routing packets onto the Internet, your access to this share should be successful. The screenshot taken in this chapter relied on using a VPN from PureVPN. See https://www.purevpn.com/ for more details on this firm's offerings.

In step 13, step 14, and step 15, you tidy up and remove the SMB share from Azure, including the data (stored in Azure) and the local SMB mapping. If you intend to keep the share, then omit these final steps.

# Creating an Azure website

Azure provides many ways for you to create rich web and mobile applications in the cloud. You could set up your virtual machines, install IIS, and add your web application. If your application needs to store data, you can create a separate SQL Server VM (or use Azure's SQL database PaaS offering).

A simpler way is to create an Azure App Service. Azure App Services enable you to build, deploy, and manage rich websites and web applications. You can use frameworks such as .NET, Node.js, PHP, and Python in these applications and any database software that's appropriate for your needs. You can also take advantage of its DevOps capabilities, such as continuous deployment from Azure DevOps, GitHub, Docker Hub, and other sources, package management, staging environments, custom domain, and TLS/SSL certificates.

An Azure App Service can be a simple static HTML site or rich multi-tier application providing both web and mobile platforms. You have a lot of choices and features to exploit.

In this recipe, you create a simple single-page static website. You upload the page via FTP. To simplify the content upload, you use the PSFTP third-party module, which you get and install from the PowerShell gallery.

## Getting ready

This recipe uses SRV1, which you have used to access Azure so far in this chapter.

## How to do it...

1. Defining variables

$Locname    = 'uksouth'     # location name

$RgName     = 'packt\_rg'    # resource group we are using

$SAName     = 'packt42sa'   # storage account name

$AppSrvName = 'packt42'

$AppName    = 'packt42website'

1. Logging in to your Azure account

Login-AzAccount -Credential $CredAz

1. Getting the resource group

$RGHT1 = @{

  Name        = $RgName

  ErrorAction = 'Silentlycontinue'

}

$RG = Get-AzResourceGroup @RGHT1

1. Getting the storage account

$SAHT = @{

  Name              = $SAName

  ResourceGroupName = $RgName

  ErrorAction       = 'SilentlyContinue'

}

$SA = Get-AzStorageAccount @SAHT

1. Creating an application service plan

$SPHT = @{

     ResourceGroupName   = $RgName

     Name                = $AppSrvName

     Location            = $Locname

     Tier               =  'Free'

}

New-AzAppServicePlan @SPHT |  Out-Null

1. Viewing the service plan

$PHT = @{

  ResourceGroupName = $RGname

  Name              = $AppSrvName

}

Get-AzAppServicePlan @PHT

1. Creating the new Azure web app

$WAHT = @{

  ResourceGroupName = $RgName

  Name              = $AppName

  AppServicePlan    = $AppSrvName

  Location          = $Locname

}

New-AzWebApp @WAHT |  Out-Null

The web app name, held in the $AppName variable, needs to be unique globally, just like the storage account name. Therefore, this step is only successful if the name is unique. If you get errors from this step, you need to change the web app name until you find one that works.

1. Viewing web app details

$WebApp = Get-AzWebApp -ResourceGroupName $RgName -Name $AppName

$WebApp |

  Format-Table -Property Name, State, Hostnames, Location

1. Checking the website

$SiteUrl = "https://$($WebApp.DefaultHostName)"

Start-Process -FilePath $SiteUrl

1. Installing the PSFTP module

Install-Module PSFTP -Force | Out-Null

Import-Module PSFTP

1. Getting publishing profile XML and extracting FTP upload details

$APHT = @{

  ResourceGroupName = $RgName

  Name              = $AppName

  OutputFile        = 'C:\Foo\pdata.txt'

}

$X = [xml] (Get-AzWebAppPublishingProfile @APHT)

$X.publishData.publishProfile[1]

1. Extracting credentials and site details

$UserName = $X.publishData.publishProfile[1].userName

$UserPwd  = $X.publishData.publishProfile[1].userPWD

$Site     = $X.publishData.publishProfile[1].publishUrl

1. Setting the FTP connection

$FTPSN  = 'FTPtoAzure'

$PS     = ConvertTo-SecureString $UserPWD -AsPlainText -Force

$T      = 'System.Management.automation.PSCredentiaL'

$Cred   = New-Object -TypeName $T -ArgumentList $UserName,$PS

$FTPHT  = @{

  Credentials = $Cred

  Server      = $Site

  Session     = $FTPSN

  UsePassive  = $true

}

Set-FTPConnection @FTPHT

1. Opening an FTP session

$Session = Get-FTPConnection -Session $FTPSN

$Session

1. Creating a web page and uploading it

'My First Azure Web Site' | Out-File -FilePath C:\Foo\Index.Html

$Filename = 'C:\foo\index.html'

$IHT = @{

  Path       = '/'

  LocalPath  = 'C:\foo\index.html'

  Session    = $FTPSN

}

Add-FTPItem @IHT

1. Examining the site using your default browser

$SiteUrl = "https://$($WebApp.DefaultHostName)"

Start-Process -FilePath $SiteUrl

1. Tidying up – removing the webapp

$WebApp | Remove-AzWebApp  -Force

1. Tidying up – removing the service plan

Get-AzAppServicePlan @PHT |

  Remove-AzAppServicePlan -Force

## How it works...

As with other recipes in this chapter, in step 1, you define variables for use with this recipe. This step produces no console output.

In step 2, you log in to your Azure account, using the Azure GUI and MFA. Once you are logged in, you should see the following in your default browser.:

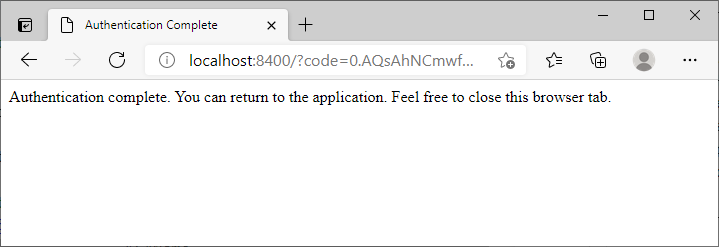


Figure 13.36: Logging into Azure

**Insert image B42024\_13\_36.png**

In step 3, you get the resource group, and in step 4, you get your storage account. You use these objects in this recipe. In step 5, you create the Azure application service plan using the Free resource tier. These three steps create no console output.

In step 6, you view the app service plan with output like this:

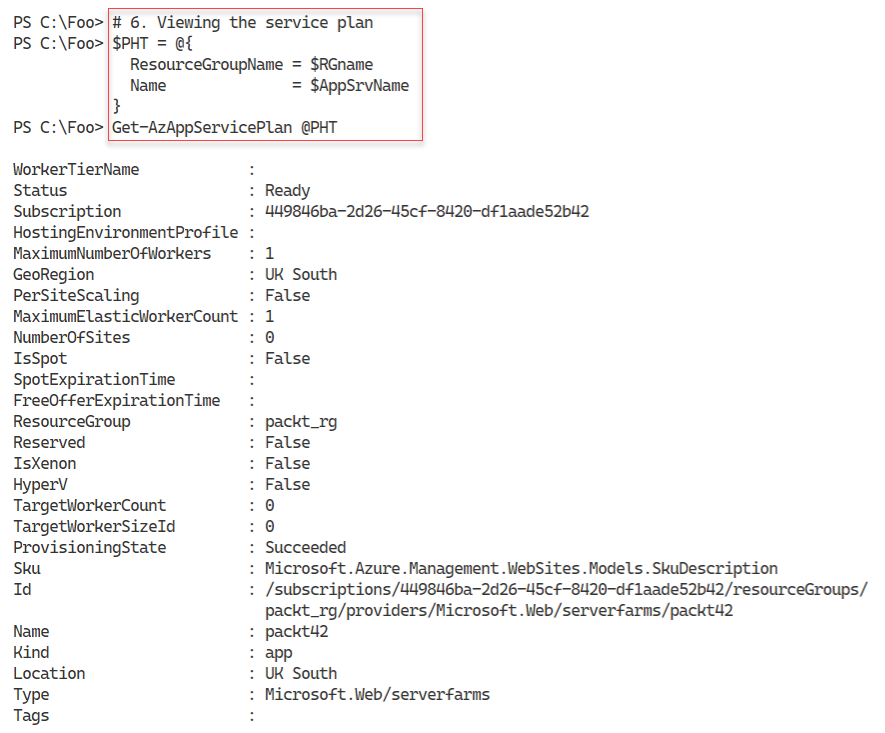


Figure 13.37: Viewing the App Service Plan

**Insert image B42024\_13\_37.png**

In step 7, you create the actual web application (website) using the New-AzWebApp cmdlet. There is no output from this step.

In step 8, you use the Get-AzWebApp command to view details of your new web application with output like this:

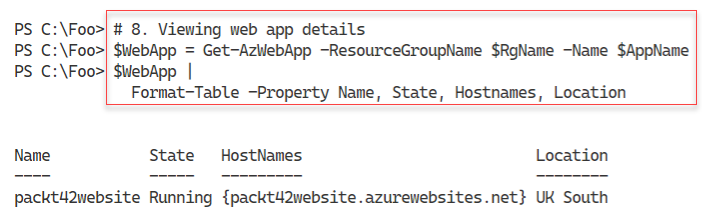


Figure 13.38: Viewing the Azure web application

**Insert image B42024\_13\_38.png**

In step 9, you use your default browser to navigate to the website, with output like this:

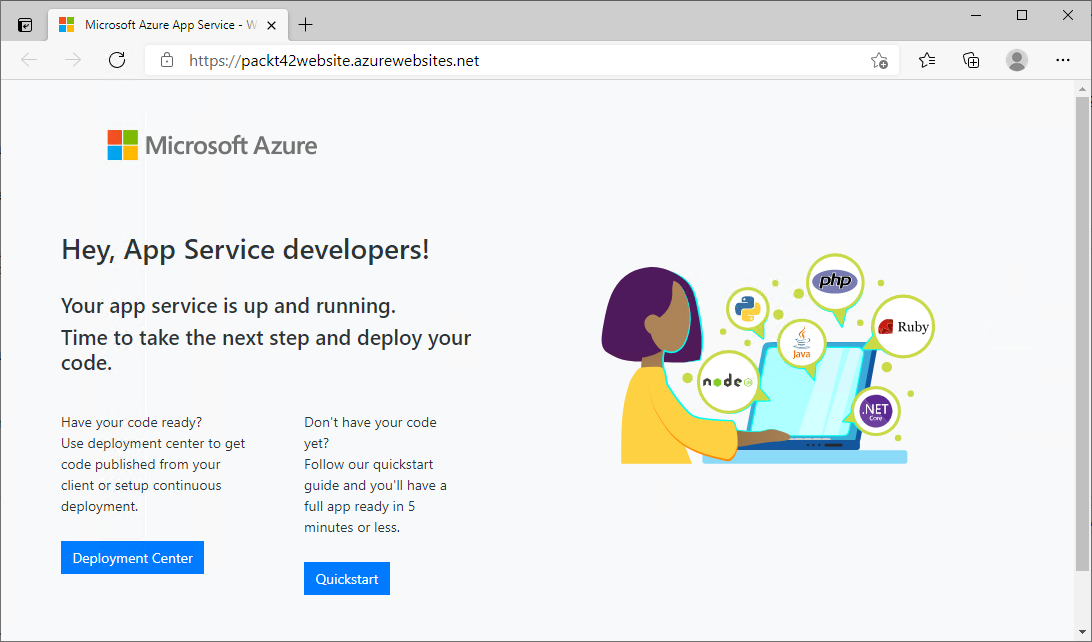


Figure 13.39: Viewing the Azure website

**Insert image B42024\_13\_39.png**

To add content to your site, you need to upload the site contents using FTP. You can use the PSFTP module to do the uploads. You first have to download and install the PSFTP module from the PowerShell gallery which you do in step 10, producing no output.

In step 11, you use Get-AzWebAppPublishingProfile to get the Azure publishing profile. This object contains details on how you can publish your website contents. This step produces the following output:

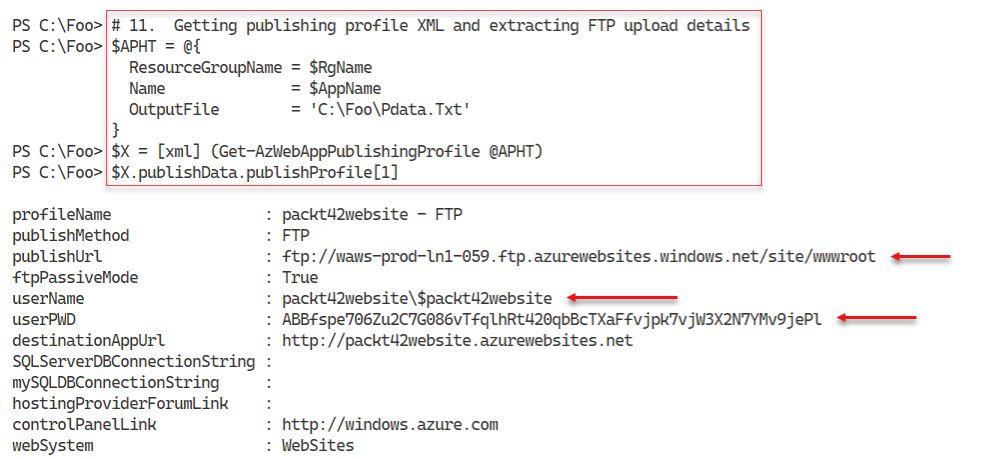


Figure 13.40: Obtaining Azure site publishing details

**Insert image B42024\_13\_40.png**

In step 12, you extract the site, user name, and password for the FTP site to upload site content. This step produces no output.

In step 13, you create an FTP session to upload your site content. The output of this step looks like this:

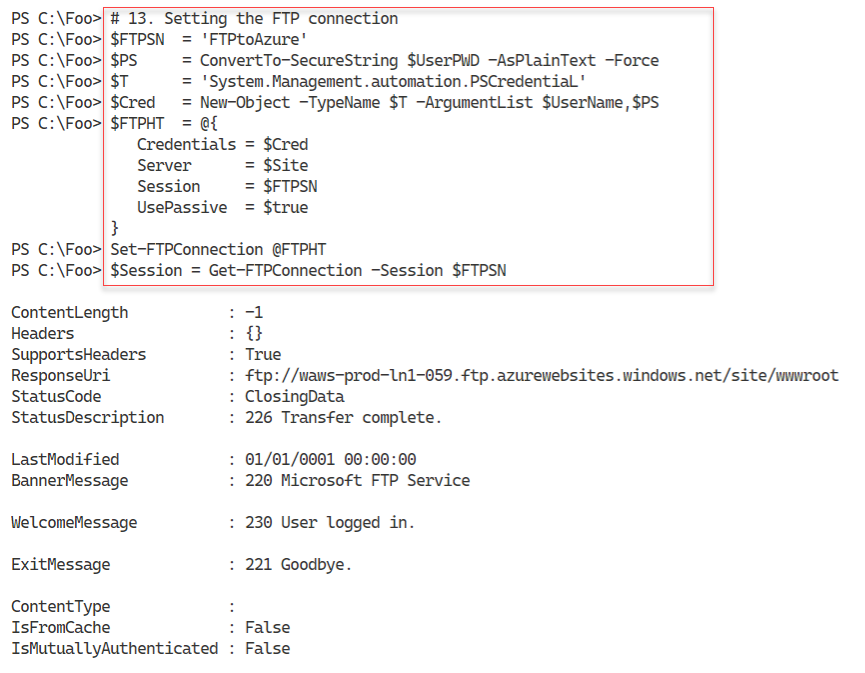


Figure 13.41: Setting an FTP connection

**Insert image B42024\_13\_41.png**

In step 14, you open an FTP session to Azure and display the details, with output like this:

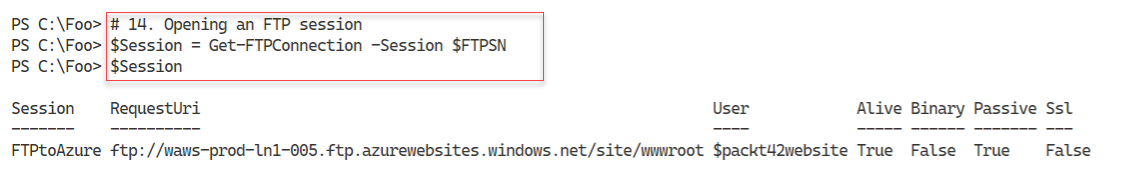


Figure 13.42: Opening an FTP session with Azure

**Insert image B42024\_13\_42.png**

In step 15, you create a simple web page, Index.html, and upload it using the FTP session opened earlier in this recipe. The output from this step looks like this:

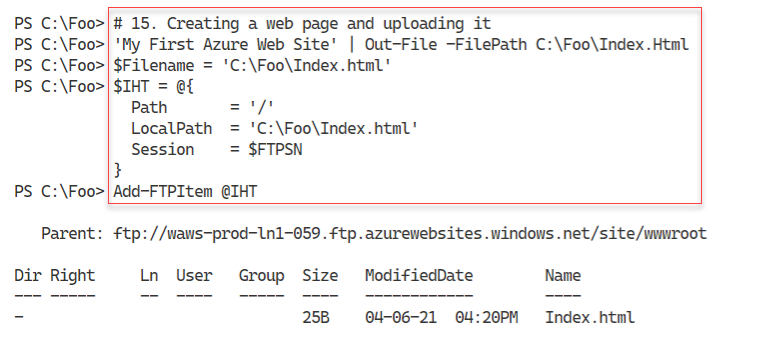


Figure 13.43: Creating and uploading a web page

**Insert image B42024\_13\_43.png**

Now that you have uploaded some content, in step 16, you view your website again, with output like this:

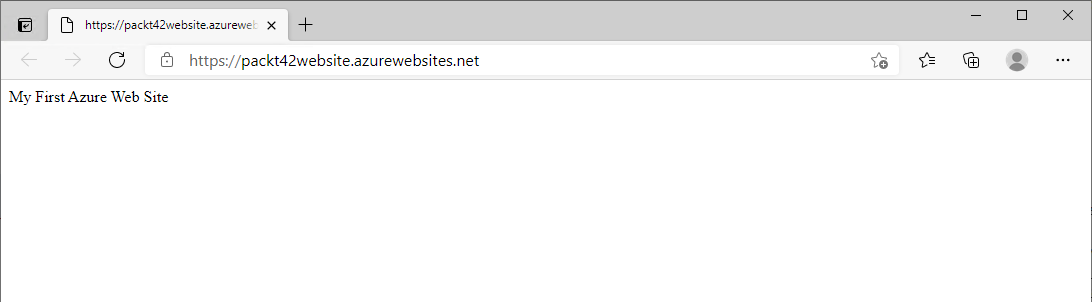


Figure 13.44: Viewing your website

**Insert image B42024\_13\_44.png**

## There's more...

In step 9, you view your newly created Azure website. The output shown in *Figure 13.39* is what Azure displays before you upload actual site content. This graphic tells you that the Azure site is up and running, letting you know you now need to upload some content.

In step 11, you get the publishing profile. This object contains the FTP publishing URL and the username and password for the FTP site. This object also gives you the full URL to the website itself.

In step 15, you create then upload a single HTML page (Index.html), generating no console output.

In step 16, you use your default browser to view this site – this time, you see your site's content. Of course, in production, there would be more pages and other files you would upload. It can take a few seconds before Azure displays the updated content.

In step 17, you begin to tidy up by removing the web app. In step 18, you complete the tidy up by removing the service plan from Azure. These two steps produce no output to the console.

# Creating an Azure Virtual Machine

Azure provides a range of on-demand computing resources, one of which is virtual machines (VMs). An Azure VM is a good solution for more control over the computing environment than you might obtain using a PaaS service.

An Azure VM is essentially a Hyper-V VM that you run within Azure. There are some differences between the Hyper-V VMs you create within Server 2022 (or Windows 10) and Azure VMs, but they are minor. The AZ cmdlets you use to manage Azure VMs are slightly different in style from the Hyper‑V cmdlets, which may mean a bit of a learning curve.

## Getting ready

This recipe uses SRV1, which you have used to access Azure so far in this chapter.

## How to do it...

1. Defining key variables

$Locname = 'uksouth'          # location name

$RgName  = 'packt\_rg'         # resource group name

$SAName  = 'packt42sa'        # Storage account name

$VNName  = 'packtvnet'        # Virtual Network Name

$CloudSN = 'packtcloudsn'     # Cloud subnet name

$NSGName = 'packt\_nsg'        # NSG name

$Ports   = @(80, 3389)        # ports to open in VPN

$IPName  = 'Packt\_IP1'        # Private IP Address name

$User    = 'AzureAdmin'       # User Name

$UserPS  = 'JerryRocks42!'    # User Password

$VMName  = 'Packt42VM'        # VM Name

1. Logging into your Azure account

$CredAZ = Get-Credential     # Enter your Azure Credential details

$Account = Connect-AzAccount -Credential $CredAZ

1. Getting the resource group

$RG = Get-AzResourceGroup -Name $RgName

1. Getting the storage account

$SA = Get-AzStorageAccount -Name $SAName -ResourceGroupName $RgName

1. Creating VM credentials

$T = 'System.Management.Automation.PSCredential'

$P = ConvertTo-SecureString -String $UserPS -AsPlainText -Force

$VMCred = New-Object -TypeName $T -ArgumentList $User, $P

1. Creating a simple VM using defaults

$VMHT = @{

    ResourceGroupName   = $RgName

    Location            = $Locname

    Name                = $VMName

    VirtualNetworkName  = $VNName

    SubnetName          = $CloudSN

    SecurityGroupName   = $NSGName

    PublicIpAddressName = $IPName

    OpenPorts           = $Ports

    Credential          = $VMCred

}

New-AzVm @VMHT

1. Getting the VM's External IP address

$VMIP = Get-AzPublicIpAddress -ResourceGroupName $RGname

$VMIP = $VMIP.IpAddress

"VM Public IP Address: [$VMIP]"

1. Connecting to the VM

mstsc /v:"$VMIP"

1. Tidying up – stopping and removing the Azure VM

Stop-AzVm -Name $VMName -Resourcegroup $RgName -Force |

  Out-Null

Remove-AZVm -Resourcegroup $RgName -Name $VMName -Force |

  Out-Null

1. Tidying up – Removing VM's networking artifacts

Remove-AzNetworkInterface -Resourcegroup $RgName -Name $VMName -Force

Remove-AzPublicIpAddress -Resourcegroup $RgName -Name $IPName -Force

Remove-AzNetworkSecurityGroup -Resourcegroup $RgName -Name $NSGName -Force

Remove-AzVirtualNetwork -Name $VNName -Resourcegroup $RgName -Force

Get-AzNetworkWatcher | Remove-AzNetworkWatcher

Get-AzResourceGroup -Name NetworkWatcherRG  |

  Remove-AzResourceGroup -Force |

    Out-Null

1. Tidying up – Removing the VM's disk

Get-AzDisk | Where-Object name -match "packt42vm" |

  Remove-AzDisk -Force |

    Out-Null

1. Tidying up – Removing the storage account and resource group

Get-AzStorageAccount -StorageAccountName $SAName -ResourceGroupName $RgName |

  Remove-AzStorageAccount -Force

Get-AzResourceGroup -Name $RgName |

  Remove-AzResourceGroup -Force

    Out-Null

## How it works...

In step 1, you create variables to hold important values for this recipe. This step creates no console output.

In step 2, you log into your Azure account, with output like this:

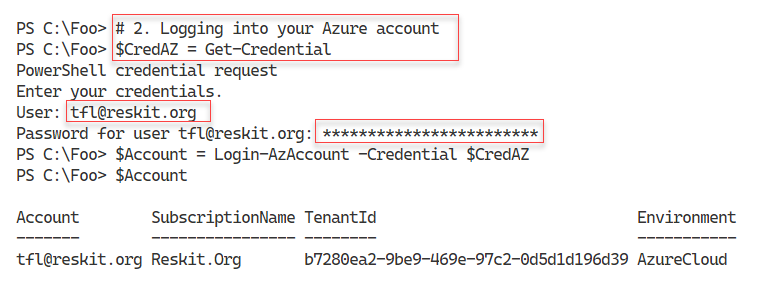


Figure 13.45: Logging into Azure

**Insert image B42024\_13\_45.png**

In step 3, you retrieve the resource group, and in step 4, you retrieve the storage account, creating no console output. You created these in an earlier step.

In step 5, you create a credential object you use with the Azure VM. This step also produces no output.

In step 6, you use the New-AzVm command to create a new Azure VM, which looks like this:

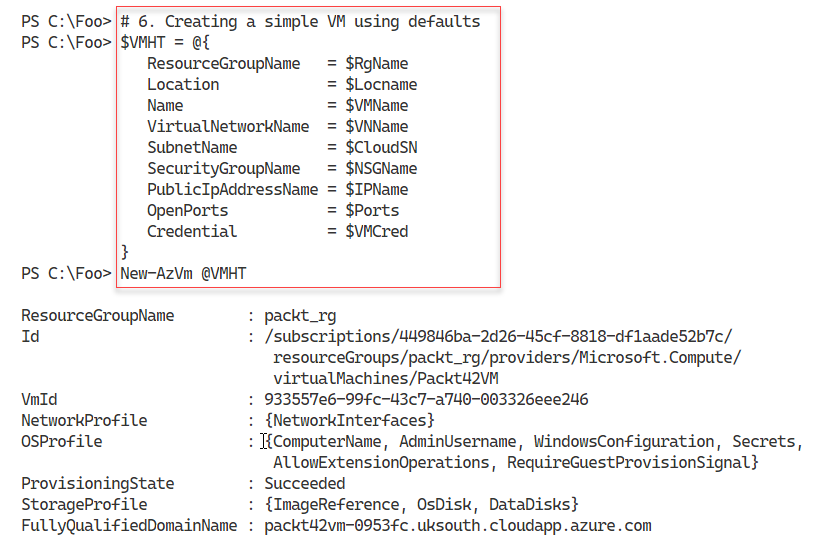


Figure 13.46: Creating an Azure VM

**Insert image B42024\_13\_46.png**

In step 7, you use the Get-AzPublicIpAddress cmdlet to obtain the IPv4 address of the Azure VM. The output should look like this:

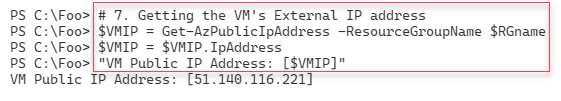


Figure 13.47: Creating an Azure VM

**Insert image B42024\_13\_46a.png**

In step 8, you use the Windows Remote Desktop Protocol (RDP) client and open up a terminal connection with the VM over the Internet. After typing a few commands in the terminal window, the results should look like this:

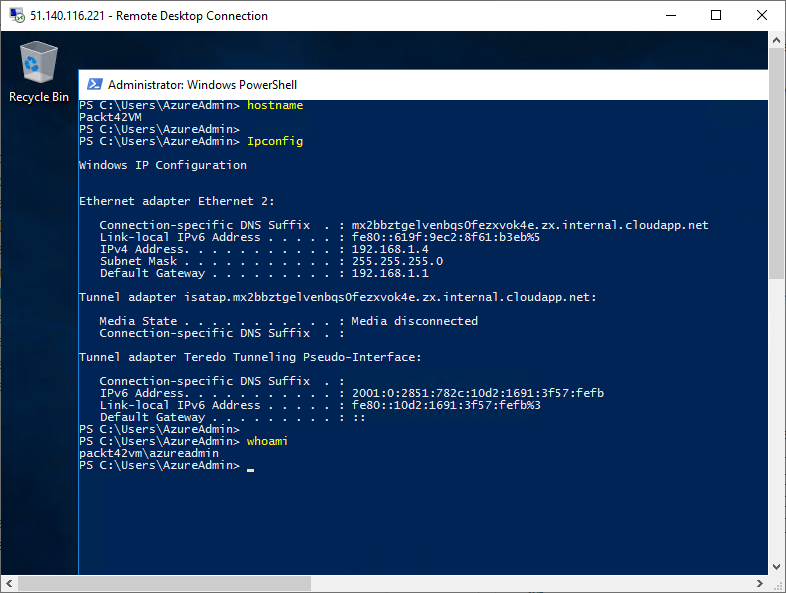


Figure 13.48: Using a terminal session into your Azure VM

**Insert image B42024\_13\_47.png**

In step 9, you begin to tidy up by stopping and then removing the Azure VM. In step 10, you remove the network-related artifacts, including the VM’s network interface, the public IP address, and so on. In step 11, you remove the VM’s virtual disk. In step 12, you remove both the storage account and resource group you have used in the recipes in this chapter. These four steps produce no console output.

## There's more...

In step 1, you create variables to hold important values for this recipe. In production or testing, you would probably need to change some of these values since certain names, such as the storage account or Azure web app name, need to be unique globally. You need to test any such names for uniqueness. If you plan to deploy Azure VMs in production, you should devise a corporate naming convention. In creating your naming convention, you have to balance ease of use versus uniqueness. A 32-character VM storage account name is highly likely to be unique but can be challenging to type at the console.

In step 6, you create a new Azure VM. The creation process can take several minutes to complete since Azure has to create your VM on a compute server, provision a disk image, then complete the Windows Server installation. The performance you observe may well depend on the Azure location and its current workload. Also – when you created the VM, you did not specify the -Size parameter. By default, unless you specify a different VM size, Azure defaults to Standard\_D2s\_V3. For more information on VM sizes, see: https://docs.microsoft.com/azure/virtual-machines/sizes.

In step 8, you use the Windows Remote Desktop Client application to create a terminal session into your new Azure VM. Note that having the RDP port open inbound from the Internet provides good functionality, but it is a security risk. If you need RDP access, you access the RDP session through a VPN.

In the final four steps, you remove the Azure VM, the VM’s network artifacts, the VM disk and finally the storage account and resource group you have used throughout this recipe and chapter.