12

Managing Azure

In this chapter, we cover the following recipes:

* Using PowerShell with Azure
* Creating core Azure resources
* Exploring your storage account
* Creating an Azure SMB file share
* Create an Azure website
* Creating an Azure VM

# Introduction

Azure is Microsoft's cloud computing platform and is a competitor to Amazon's Amazon Web Services and other public cloud providers, Azure provides you with access to a huge range of features. Organizations can literally move 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 the basic computing infrastructure components (servers, storage, networking, firewalls, and security), plus the physical plant that's required to run these components (power, air conditioning, and so on). In an IaaS environment, the servers are all 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 deploy SQL in an IaaS environment, are all managed by Azure. This provides a complete SQL service, all managed by Azure. This, of course, means there are some things you can't do—actions that are reserved for the platform owner (that is, Microsoft). For example, with SQL running inside an IaaS Azure VM, you can use database mirroring—the SQL PaaS service does not provide that feature for you to use.

With SaaS, you just use an application that the vendor has placed in the cloud. One key example of SaaS is Office 365 (O365), which bundles Exchange Online, SharePoint Online, Skype For Business Online, OneDrive for Business, and Microsoft Teams. 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 also include a downloadable version of the Office applications, such as Word and Excel). In terms of using PowerShell to manage Office 365, each of the included applications has its own unique approach. With Exchange Online, for example, you use PowerShell Implicit Remoting to manage the exchange component of your Office 365 subscription.

To provide authentication for software running within Azure and for other SaaS applications, you can make use of Azure Active Directory (AAD). With AAD, you can create a cloud-only directory or you can 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 in terms of managing both AAD and Office 365 components are outside the scope of this chapter.

In this chapter, we begin with the first recipe: Using PowerShell with Azure. In this recipe, we look at setting up a basic environment that we can manage Azure and the Office 365 SaaS components. This recipe also shows how to download the AAD cmdlets.

The Creating core Azure resources recipe 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. Every Azure resource you create with the ARM API must be contained in a resource group. Also, any storage you may require, such as VHD files for an Azure VM, need to be stored in a storage group. While the recipes in this chapter use a single resource group and a single storage account, large-scale Azure deployments may require multiple instances of these key resources.

In the Creating Azure storage recipe, we look at setting up Azure storage using the storage account we created earlier. The Creating and using an Azure SMB file share recipe shows you how you can create an SMB 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 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 and using Azure websites recipe shows you how you can set up a simple website. The recipe sets up a WordPress blog using PowerShell. This feature enables you to set up a simple website, say for a short-term marketing campaign, as well as build internet-scale websites that you can have Azure scale dynamically according to load.

The next recipe, Creating and using Azure virtual machines, examines how to create an Azure VM and access it. This includes creating a virtual network and setting the VM up to enable you to manage it with PowerShell or connect via RDP. This chapter is only a taster for using Azure with PowerShell. There is so much more that you can do that could not fit into this book.

# Using PowerShell with Azure

There are two key 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 get access to the cmdlets you need to be able to access Azure (and Office 365's features).

Azure is a commercial service—each feature you use has a cost attached. Azure charges are based on resource usage. With an Azure VM, for example, you would pay to have the VM running, with additional charges for the storage the VM uses and for any network traffic.

The charges for Office 365, on the other hand, are user-based—a given user can use lots of email, for example, without incurring any additional charges. For details on costs for Azure, see <https://azure.microsoft.com/pricing/>, and for details of Office 365 charges, see <https://products.office.com/business/compare-office-365-for-business-plans>.

To use Azure's IaaS and PaaS features, you need to have an Azure subscription. There are many ways you can get an Azure subscription, including via an MSDN subscription, an Action Pack subscription, or by outright purchase. Naturally, you need to ensure that any systems are properly licensed.

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

If you do not have an existing subscription to Azure, navigate to <https://azure.microsoft.com/free/>, where you can create a trial subscription.

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 to identify verification—plus it keeps the lawyers happier.

If you take out an Azure trial and you want to keep your Azure resources running after the trial expires, you have to move it to a pay as you go subscription. You will receive an email shortly before the trial expires to transition it, which prevents downtime if you are using the trial for production.

To use PowerShell with Azure's various features, you need to obtain cmdlets that Microsoft does not provide in Windows Server 2019, Windows PowerShell 5.0/5.1, or PowerShell Core. You get the relevant modules from the PowerShell Gallery using the cmdlets in the PowerShellGet module to find and download them.

Azure has had PowerShell support almost since the very start of the service. These cmdlet sets have changed as Azure has matured (and expanded in scope). In 2019, the Azure PowerShell team released a new module, the Az module, to serve as the basis for automating Azure operations with PowerShell. The Az module is actually a set of modules containing over 2,500 cmdlets, a few of which you explore in this chapter.

## Getting ready

To run this recipe, and all the recipes in this chapter, on CL1, which you initially configured in the Installing RSAT Tools on Windows 10 and Windows Server 2019 recipe in [Chapter 1](file:///C:\Users\siddh\OneDrive\Desktop\PEN\September%2022%20-%20Parvathy%20-%20Chapter%201-12%20STY\9781789808537\OEBPS\ch01.html), Establishing a PowerShell Administrative Environment.

## How to do it...

1. Find the core AZ module:

Find-Module -Name Az

1. Install the AZ modules:

Install-Module -Name Az -Force

1. Discover Azure modules and how many cmdlets each contains:

$HT = @{ Label ='Cmdlets'

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

Get-Module Az\* -ListAvailable |

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

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

1. Find the Azure AD cmdlets:

Find-Module AzureAD |

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

1. Download and install the AzureAD module:

Install-Module -Name AzureAD -Force

1. Discover the AzureAD module:

$FTHT = @{

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

AutoSize = $true

Wrap = $true

}

Get-Module -Name AzureAD -ListAvailable |

Format-Table @FTHT

1. Log in to Azure:

$CredAZ = Get-Credential

$Subscription = Login-AzAccount -Credential $CredAZ

1. Obtain the Azure subscription details:

$SubID = $Subscription.Context.Subscription.SubscriptionId

Get-AzSubscription -SubscriptionId $SubId |

Format-List -Property \*

1. Get Azure locations:

Get-AzLocation | Sort-Object Location |

Format-Table Location, Displayname

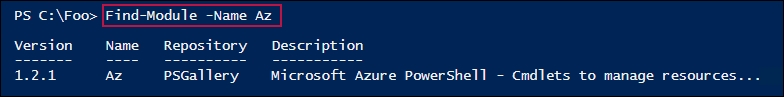
1. Get Azure environments:

Get-AzEnvironment |

Format-Table -Property name, ManagementPortalURL

## How it works...

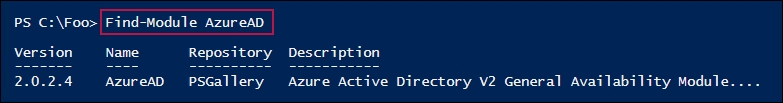
In step 1, you used the Find-Module cmdlet to search the PowerShell Gallery for the AZ module, which produces the following output:



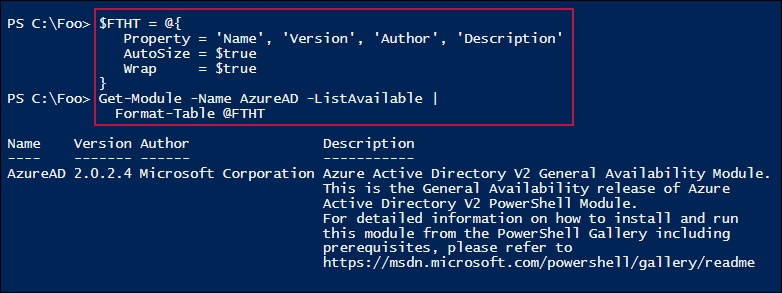
In step 2, you downloaded and installed the AZ modules, which generates no output. In step 3, you looked at the downloaded modules, which look like this:



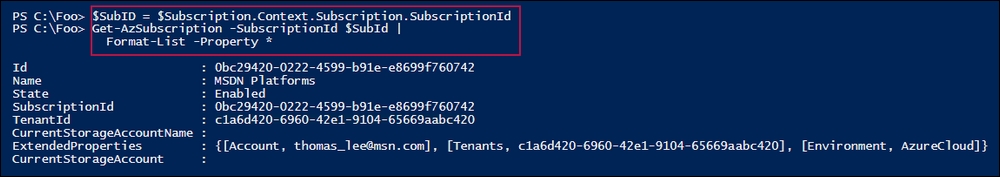
In step 4, you found the Azure AD module, which generates output like this:



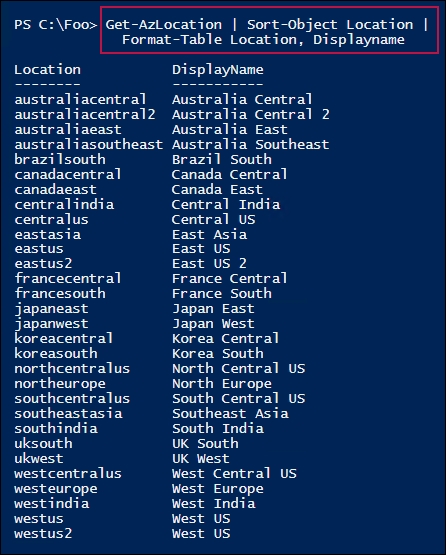
In step 5, you installed the Azure AD module, which generates no output. After you installed the module, in step 6, you examined the module, which generated the following output:



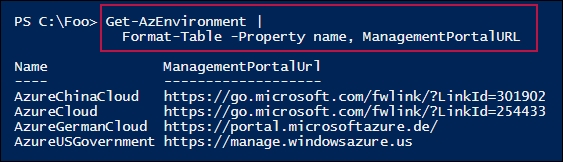
In step 7, you used your subscription logon information and logged on to Azure. This step produced output. In step 8, you got the details of your subscription, which looks like this:



In step 9, you used the Get-AzLocation cmdlet to discover the current Azure locations, which produces output like this:



In the final step, step 10, you viewed the different public Azure clouds that are run by Microsoft. The output of this step looks like this:



## There's more...

In step 3, you examined the modules you downloaded. Possibly confusingly, you downloaded the Az module, which actually downloads the 46 separate modules you can see in the preceding screenshots.

In step 9, you viewed the current Azure public cloud locations. Each of these 30 locations provide a range of services, although not all services are necessarily in every location. In step 10, you viewed the current Azure environments. Each of these four environments are totally separate and unrelated cloud offerings. The recipes in this chapter should run unchanged on each of the four Azure cloud environments, but no testing has been undertaken in this regard. You should also be aware that Azure is a very fast changing set of products. By the time you read this book, Azure may well have expanded.

See also

To keep up to date about new Azure product updates as well as roadmap announcements, see the Azure updates page at [https://azure.microsoft.com/updates/](https://azure.microsoft.com/en-gb/updates/). The recipes in this chapter make use of Azure data centers in Europe. Depending on where you live, you may find that these data centers are further away from locations nearer to you. To discover network latency between you and the Azure locations, see <https://www.AzureSpeed.com>.

# Creating core 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 having a brief look around. In this recipe, you create certain key Azure assets, including a resource group, a storage account, and tags.

With Azure, all Azure resources are created within a resource group. A resource group is a grouping of Azure resources. Any storage you create within Azure resides in a storage account, a fundamental building block within Azure.

All storage you use with any Azure feature always exists within a storage account. You create a storage account within one of the Azure regions you saw in the Using PowerShell with Azure recipe. When you create your storage account, you also specify the level of resiliency and durability that's provided. There are several levels of replication provided within Azure, which provide for multiple copies of the data that are replicated automatically in both the local Azure data center but also in other data centers. The extra resilience, which does come at a price, provides greater levels 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 kind of data (as you see in the Exploring your 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

This recipe requires you to have an Azure account and that you have your system configured with the Az module, which was done in the Using PowerShell with Azure recipe.

## How to do it...

1. Set values for key variables:

$Locname = 'uksouth' # location name

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

$SAName = 'packt42sa' # Storage account name

1. Log in to your Azure account:

$CredAZ = Get-Credential

Login-AzAccount -Credential $CredAZ

1. Create a resource group and tag it:

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

$RGTag += @{Author='Thomas Lee'}

$RGHT = @{

Name = $RgName

Location = $Locname

Tag = $RGTag

}

$RG = New-AzResourceGroup @RGHT

$RG

1. View the Resource Group details:

Get-AzResourceGroup -Name $RGName |

Format-List -Property \*

1. Test to see if a Storage Account name is available:

Get-AzStorageAccountNameAvailability $SAName

1. Create a new Storage Account within our newly created resource group:

$SAHT = @{

Name = $SAName

SkuName = 'Standard\_LRS'

ResourceGroupName = $RgName

Tag = $RGTag

Location = $Locname

}

New-AzStorageAccount @SAHT

1. Get an overview of the Storage Account in this Resource Group:

$SA = Get-AzStorageAccount -ResourceGroupName $RgName

$SA |

Format-List -Property \*

1. Get primary endpoints for the Storage Account:

$SA.PrimaryEndpoints

1. Review SKU for this Storage Account:

$SA.Sku

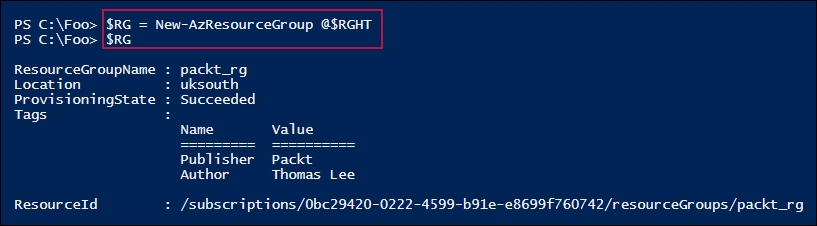
1. View the value of the C property of your Storage Account:

$SA.Context

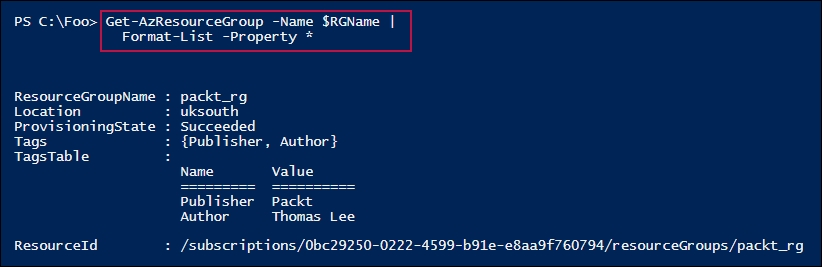
## How it works…

In step 1, you set the value of a number of key variables that are used in this recipe. In step 2, you logged in to your Azure account. Neither of these steps produce output.

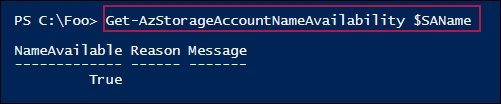
In step 3, you created a new Azure Resource Group and gave it some nice tags. This step produces output like this:



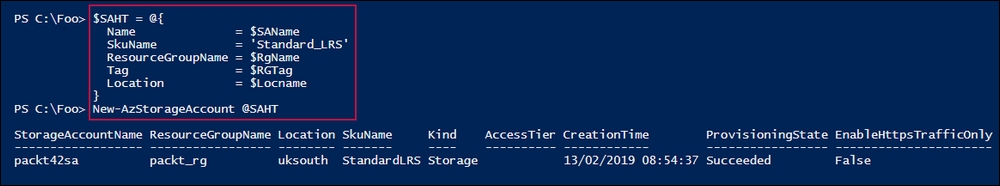
In step 4, you got the Resource Group and viewed its properties, as follows:



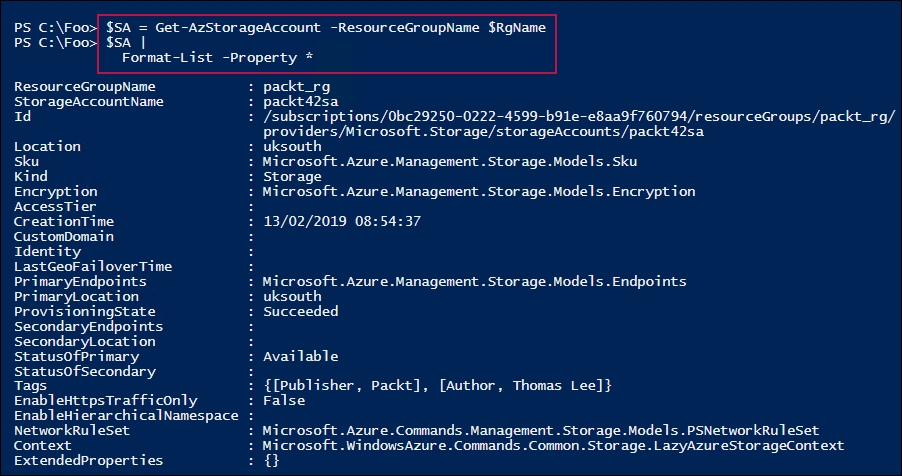
To create a Storage Account, you needed to check that the name was available, as shown in step 5:



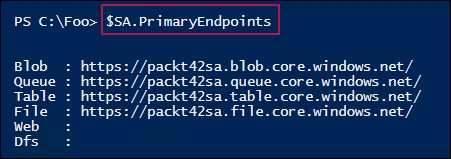
In step 6, you created a new Storage Account:



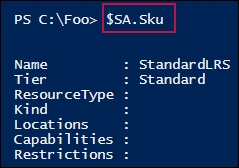
In step 7, you used the Get-AzStorageAccount cmdlet to view details of the Storage Account, which looks like this:



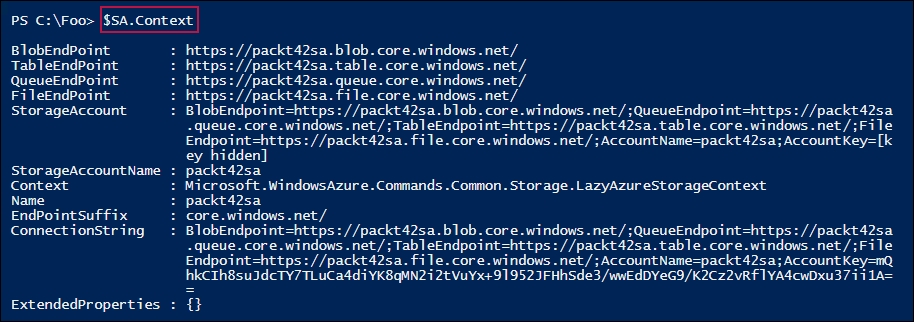
In step 8, you examined the key endpoints for your newly created Storage Account, which looks like this:



In step 9, you viewed the SKU for your Storage Account, like this:



In step 10, you viewed the details of the Storage Account's Context property, which looks like this:



## There's more...

With step 1, you created some variables and gave them the appropriate values. When you go to replicate these recipes, you may find that some of the names that are used here are already in use by other Azure customers. To run this and other recipes in this chapter, you may need to adjust these values. For example, the Storage Account name that was used in the previous edition of this book has been taken by someone so it cannot be used!

In step 2, you used Login-AzAccount to log in to Azure. In earlier versions of the Azure cmdlets, this cmdlet was somewhat restricted and did not support Microsoft accounts (for example, accounts that end in @MSN.Com). That shortcoming is eliminated in the Az cmdlets.

In step 6, you created a new Storage Account, but only after you determined the storage account name was valid. With Azure, the Storage Name (an important part of the URIs that PowerShell uses to connect to Azure) must be globally unique. Thus, no two Azure customers in the world can use the same storage account names. To ensure that your account is valid, in step 5, you used the cmdlet Get-AzStorageAccountNameAvailability to test that the name was indeed allowed. If the name was not valid, you would have to choose another storage account name and amend the recipe accordingly.

## See also

Resource groups are a fundamental part of the Azure Resource Management API. For a good introduction to the Azure Resource Management API, see <https://docs.microsoft.com/azure/storage/common/storage-introduction>.

Storage in Azure is both complex and somewhat different to on-premise storage solutions. Take a look at <https://docs.microsoft.com/azure/storage/common/storage-introduction> for an introduction to Azure Storage.

# Exploring your storage account

Many Azure resources 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. This could include MP4 movies, ISO images, VHD drives, JPG files, and so on. Individual blobs reside within blob containers, which are 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. This contrasts with an SQL table, which holds highly normalized data. A 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's used to implement scalableapplications. 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 applicationcomponents for independent scaling and to provide greater resiliency. For more details on Azure queues, see <https://azure.microsoft.com/services/storage/queues/>.

The Azure file feature provides simple cross-platform file storage that you can access using SMB. This enables 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 2.1 and 3.0, which makes it simple and easy for you 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 disks are designed for low latency and high throughput. You can provision both traditional spinning disks as well as SSD disks that provide better I/O performance for I/O intensive applications. For more details on Azure disk storage, see <https://azure.microsoft.com/services/storage/disks/>.

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, you name your storage account based on a global naming scheme which is based on HTTPS URLs. The Azure REST API relies on URLs to manage the Azure resources in your resource groups. All storage accounts are named by specifying the storage account, data type, container name, and filename. The format for a blob is as follows:

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

## Getting ready

Run this recipe on CL1, which you previously configured via the Using PowerShell with Azure recipe.

## How to do it...

1. Define key variables:

$Locname = 'uksouth' # location name

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

$SAName = 'packt42sa' # Storage account name

$CName = 'packtcontainer'

$CName2 = 'packtcontainer2'

1. Log in to your Azure account:

$CredAz = Get-Credential

Login-AzAccount -Credential $CredAz

1. Ensure that the Resource Group and the Storage Account have been created:

$RGHT = @{

Name = $RgName

ErrorAction = 'SilentlyContinue'

}

$RG = Get-AzResourceGroup @RGHT

if (-not $RG) {

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

$RGTag += @{Author='Thomas Lee'}

$RGHT2 = @{

Name = $RgName

Location = $Locname

Tag = $RGTag

}

$RG = New-AzureRmResourceGroup @RGHT2

"RG $RgName created"

}

$SAHT = @{

Name = $SAName

ResourceGroupName = $RgName

ErrorAction = 'SilentlyContinue'

}

$SA = Get-AzStorageAccount @SAHT

if (-not $SA) {

$SATag = [Ordered] @{Publisher = 'Packt'}

$SATag += @{Author = 'Thomas Lee'}

$SAHT = @{

Name = $SAName

ResourceGroupName = $RgName

Location = $Locname

Tag = $SATag

SkuName = 'Standard\_LRS'

}

$SA = New-AzStorageAccount @SAHT

"SA $SAName created"

}

1. Get and display the Storage Account key:

$SAKHT = @{

Name = $SAName

ResourceGroupName = $RgName

}

$Sak = Get-AzStorageAccountKey @SAKHT

$Sak

1. Extract the first key's password:

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

1. Get and view the Storage Account Context, which encapsulates credentials for the storage account:

$SCHT = @{

StorageAccountName = $SAName

StorageAccountKey = $Key

}

$SACon = New-AzStorageContext @SCHT

$SACon

1. Create two blob containers:

$CHT = @{

Context = $SACon

Permission = 'Blob'

}

New-AzStorageContainer -Name $CName @CHT

New-AzStorageContainer -Name $CName2 @CHT

1. View the blob container:

Get-AzStorageContainer -Context $SACon |

Select-Object -ExpandProperty CloudBlobContainer

1. Create a very small blob in Azure:

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

$BHT = @{

Context = $SACon

File = '.\azurefile.txt'

Container = $CName

}

$Blob = Set-AzStorageBlobContent @BHT

$Blob

1. Construct and display the blob name:

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

$BlobUrl

1. View the URL via IE:

$IE = New-Object -ComObject InterNetExplorer.Application

$IE.Navigate2($BlobUrl)

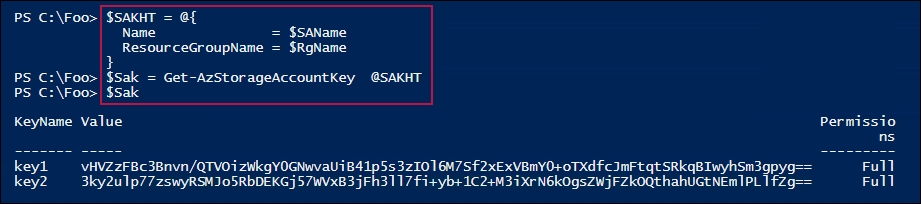
$IE.Visible = $true

## How it works...

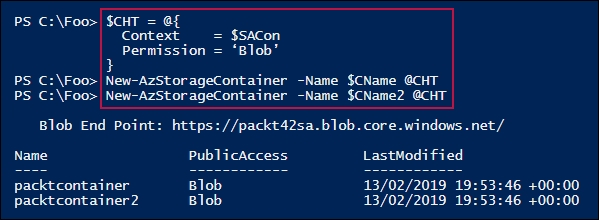
In step 1, you defined the key variables that are used in the recipe and, in step 2, you logged in to your Azure account. These two steps produce no output.

In step 3, you ensured that the Resource Group and Storage Account were created and created them if not. Assuming you completed the previous recipe, Creating core Azure resources, this step produces no output. A message indicating that the resource group and/or storage account were not found and were created is issued if the resources were not found.

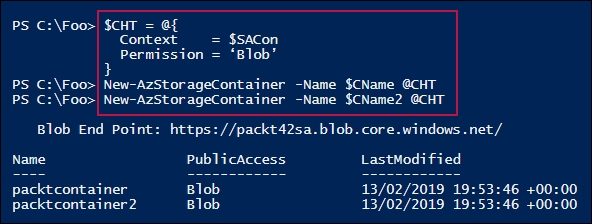
In step 4, you gathered and displayed the storage account key, which looks like this:



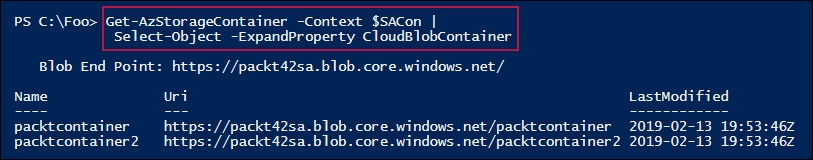
In step 5, you extracted the password from the first storage account key, which produces no output. Using that password, in step 6, you created and displayed the Storage Account context, which looks like this:



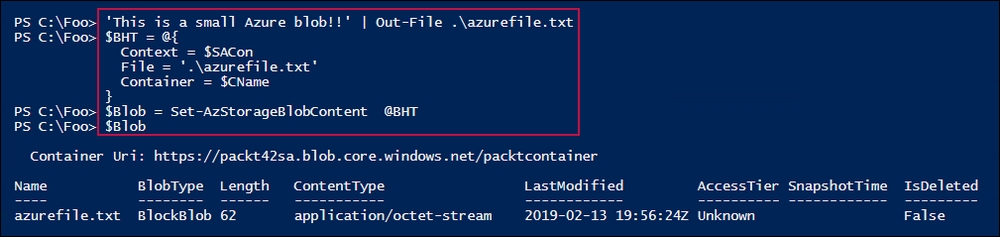
In step 7, you used the Storage Account Context (which encapsulates the credentials) to create two Azure blob containers, as follows:



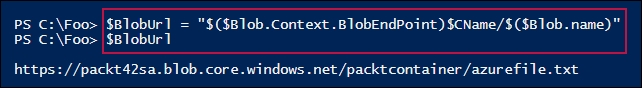
In step 8, you viewed the two new blob containers, as follows:



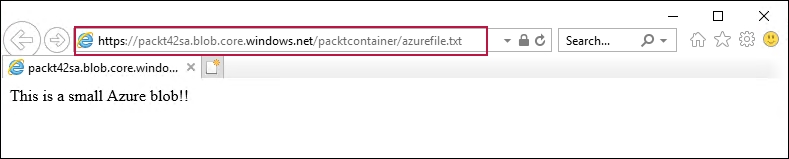
In step 9, you created a new Azure blob by using the Set-AzStorageBlobContent cmdlet. The output looks like this:



In step 10, you generated a URL for this blob, as follows:



In step 11, you viewed the blob's content using Internet Explorer, as follows:



## There's more...

In step 4, you retrieved the Storage Account keys for your Storage Account. Each key's value property is, in effect, a password for your Azure storage account. Having two keys enables you to regularly regenerate and rotate your key values. In step 5, you got this value for the first key.

In step 6, you got the storage account's storage context. This object encapsulates the details of the storage account, including the storage account key you created in the prior step.

In step 7 and step 8, you created two blob containers and displayed their URLs. Containers are a single-level, folder-like object that contains your blobs. In step 9, you created a simple blob and, as you can see from the output, this is a block blob, with the content just comprising an octet stream.

# 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 9](file:///C:\Users\siddh\OneDrive\Desktop\PEN\September%2022%20-%20Parvathy%20-%20Chapter%201-12%20STY\9781789808537\OEBPS\ch09.html), Managing Network Shares. The key difference is how you create them and the credentials 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 use a user ID/password credential, 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 provides you with two keys (imaginatively named key1 and key2). The values of both keys 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. Some time later, you repeat this—changing the application to use key1's value and then regenerate key2. This provides you with an immediate key update where you need it. Armed with the value of either key, you can easily create SMB shares that are directly addressed across the internet.

An Azure SMB share differs from Azure blobs with respect to how you access them. You access a blob via HTTP, whereas you access an Azure file share via the standard SMB networking commands that you used in, for example, [Chapter 5](file:///C:\Users\siddh\OneDrive\Desktop\PEN\September%2022%20-%20Parvathy%20-%20Chapter%201-12%20STY\9781789808537\OEBPS\ch05.html), Managing Shared Data.

Blobs and files also differ in that with blobs, you only have a single level of a folder (the container). With Azure files, 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 code.

In this recipe, you use the resource group and storage account we created earlier (in the Create Core 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 the CL1 host that you set up in the Using PowerShell with Azure recipe.

## How to do it...

1. Define the 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. Log in to your Azure Account and ensure that the RG and SA have been created:

$CredAZ = Get-Credential

$Account = Login-AzAccount -Credential $CredAZ

1. Get a Storage account, Storage 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. Add credentials to the local store:

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

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

1. Create an Azure Files file share:

New-AzStorageShare -Name share2 -Context $SACon

1. Test that the share is reachable:

$TNCHT = @{

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

Port = 445

}

Test-NetConnection @TNCHT

1. Mount the share as Z:

$Mount = 'Z:'

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

$SMHT = @{

LocalPath = $Mount

RemotePath = $Rshare

UserName = $SAName

Password = $Key

}

New-SmbMapping @SMHT

1. View the share in Azure:

Get-AzStorageShare -Context $SACon |

Format-List -Property \*

1. View the local SMB mapping:

Get-SmbMapping

1. Now, use the new share. Create a folder and a file in the share:

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

'Azure and PowerShell Rock!!!' |

Out-File -FilePath Z:\Foo\recipe.txt

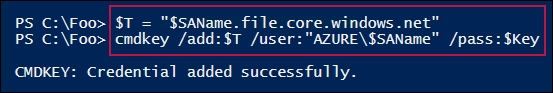
1. Get the content from the file:

Get-Content -Path Z:\Foo\recipe.txt

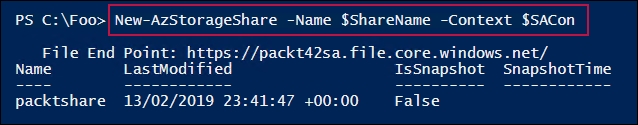
## How it works...

In step 1, you set the values for the variables that are to be used in this recipe. In step 2, you logged in to your Azure account. In step 3, you retrieved the Storage Account key and Storage Context. These three steps produce no output.

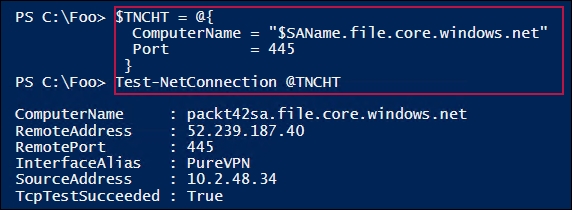
In step 4, you added the Azure Files credential details into the local store using cmdkey,exe, which looks like this:



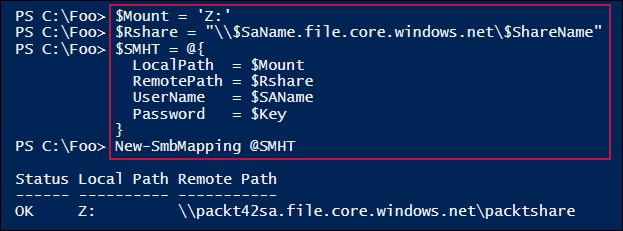
In step 5, you created a new Azure Files file share by using the New-AzStorageShare cmdlet:



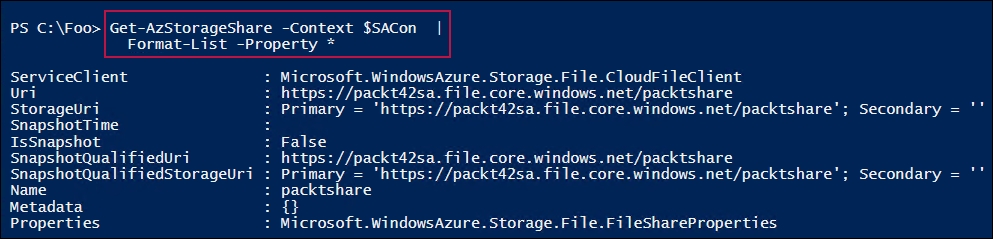
In step 6, you checked the connection over the SMB port (445) to the Azure storage server hosting the new share. The output looks like this:



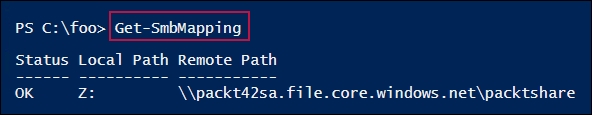
Next, in step 7, you created an SMB Mapping, mapping the Z: drive to your new share, which looks like this:



In step 8, you viewed the share in Azure, which looks like this:



In step 9, you viewed the local SMB Mappings, which looks like this:



In step 10, you created a folder in the new share and created a small file in the share. This step produces no output. In step 11, you used Get-Content to retrieve the contents of the file, which looks like this:

How it works...

## There's more...

In step 1, you created variables to hold the names of the Azure objects you were to use in this recipe. The $Locname variable holds the name of the Azure region in which you create your storage account, which you may wish to change to a more local Azure region.

In step 3, you created a storage context object. The context object encapsulates the credentials for your storage account.

You used cmdkey.exe in step 4 to save credentials for Azure's storage account. You used cmdkey to store the username and password that Windows should use to access the Azure share. For more details on cmdkey, see <https://technet.microsoft.com/library/cc754243(v=ws.11).aspx>. You can use the cmdkey utility to list all the stored credentials (cmdkey /list).

# Creating an Azure website

Azure provides a number of ways in which you can create rich web and mobile applications in the cloud. You could set up your own 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 PASS offering).

A simpler way is to create an Azure Web App. Azure Web Apps enabled 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 use any database software that's appropriate to your needs. An Azure Web App can be simple static HTML sites, or rich multi-tier applications that run on both web and mobile platforms. You have a lot of choices.

In this recipe, you create a simple single-page static website. You upload the page via FTP. The PSFTP third-party module makes the upload simple.

## Getting ready

You run this on CL1, which you set up for Azure in the Using PowerShell with Azure recipe. This recipe uses the C:\Foo folder, which should have already been created on CL1.

## How to do it...

1. Define the variables:

$Locname = 'uksouth' # location name

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

$SAName = 'packt42sa' # storage account name

$AppSrvName = 'packt42'

$AppName = 'packt42website'

1. Log in to your Azure Account:

$CredAZ = Get-Credential

$Sccount = Login-AzAccount -Credential $CredAz

1. Ensure that the Resource Group has been created:

$RGHT1 = @{

Name = $RgName

ErrorAction = 'Silentlycontinue'

}

$RG = Get-AzResourceGroup @RGHT1

if (-not $RG) {

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

$RGTag += @{Author='Thomas Lee'}

$RGHT2 = @{

Name = $RgName

Location = $Locname

Tag = $RGTag

}

$RG = New-AzResourceGroup @RGHT2

Write-Host "RG $RgName created"

}

1. Ensure that the Storage Account has been created:

$SAHT = @{

Name = $SAName

ResourceGroupName = $RgName

ErrorAction = 'SilentlyContinue'

}

$SA = Get-AzStorageAccount @SAHT

if (-not $SA) {

$SATag = [Ordered] @{Publisher='Packt'}

$SATag += @{Author='Thomas Lee'}

$SAHT = @{

Name = $SAName

ResourceGroupName = $RgName

Location = $Locname

Tag = $SATag

SkuName = 'Standard\_LRS'

}

$SA = New-AStorageAccount @SAHT

"SA $SAName created"

}

1. Create an application service plan:

$SPHT = @{

ResourceGroupName = $RgName

Name = $AppSrvName

Location = $Locname

Tier = 'Free'

}

New-AzAppServicePlan @SPHT | Out-Null

1. View the application service plan:

$PHT = @{

ResourceGroupName = $RGname

Name = $AppSrvName

}

Get-AzAppServicePlan @PHT

1. Create the new Azure Web App using the application service plan:

$WAHT = @{

ResourceGroupName = $RgName

Name = $AppName

AppServicePlan = $AppSrvName

Location = $Locname

}

New-AzWebApp @WAHT | Out-Null

1. View the application details:

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

$WebApp |

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

1. Now, view the website:

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

$IE = New-Object -ComObject InterNetExplorer.Application

$IE.Navigate2($SiteUrl)

$IE.Visible = $true

1. Install the PSFTP module:

Install-module PSFTP -Force | Out-Null

Import-Module PSFTP

1. Get the publishing profile XML and extract the FTP upload details:

$APHT = @{

ResourceGroupName = $RgName

Name = $AppName

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

}

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

$x.publishData.publishProfile[1]

1. Extract the credentials and site details:

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

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

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

1. Connect to the FTP site:

$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

$Session = Get-FTPConnection -Session $FTPSN

1. Create a web page and upload 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. Now, look at the site using the default browser (Chrome):

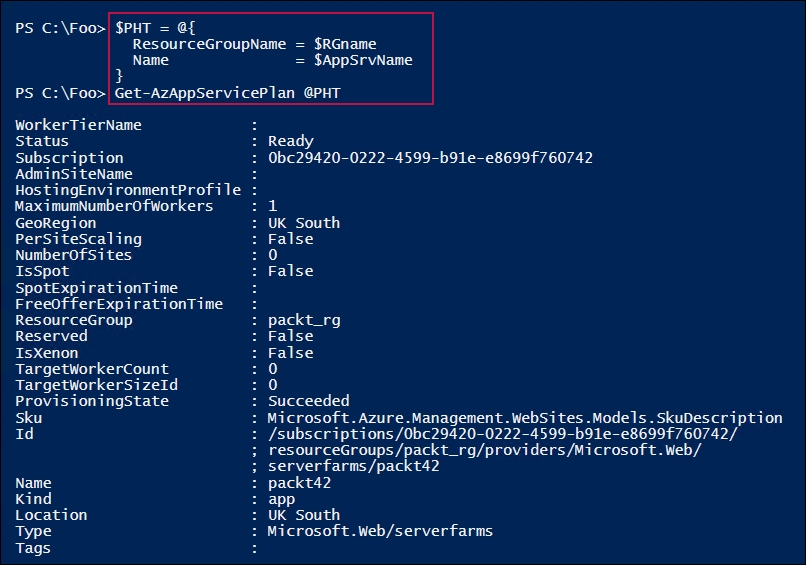
$SiteUrl = "https://$($WebApp.DefaultHostName)"

Start-Process -FilePath $SiteUrl

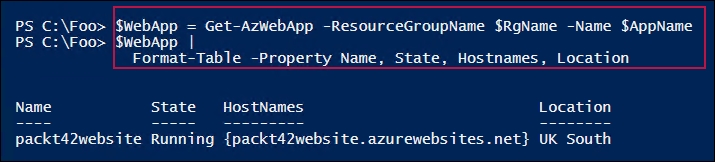
## How it works...

In step 1, you set the variables you were going to use in this recipe. In step 2, you logged in to Azure. In step 3, you ensured that the Resource Group was created and in step 4, you made sure that the Storage Account was created. In step 5, you created a new Azure application service plan. These five steps produce no output.

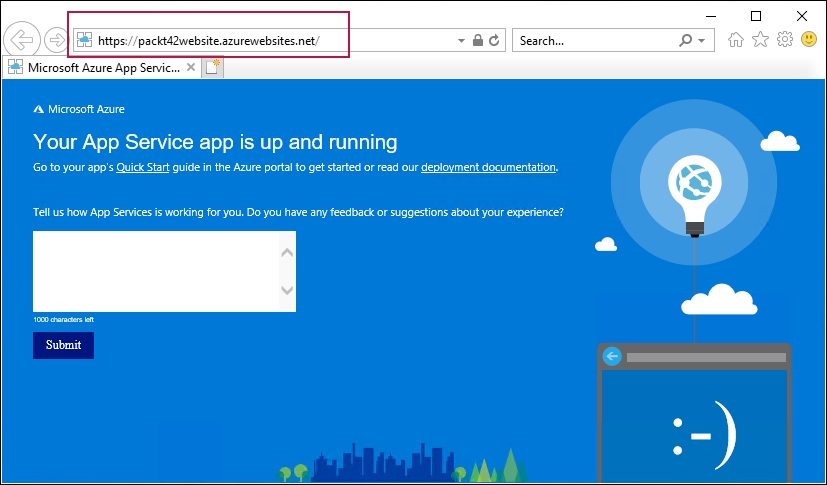
In step 6, you viewed the Azure Application Service Plan, which looks like this:



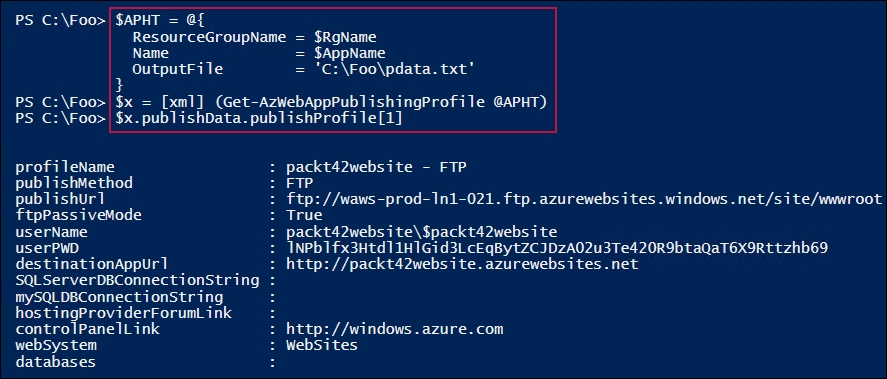
In step 7, you created a new Azure Web App using the Application Service Plan you created in the previous step. This step produces no output. In step 8, you viewed some of the properties of the Azure Web App. The output of this step looks like this:



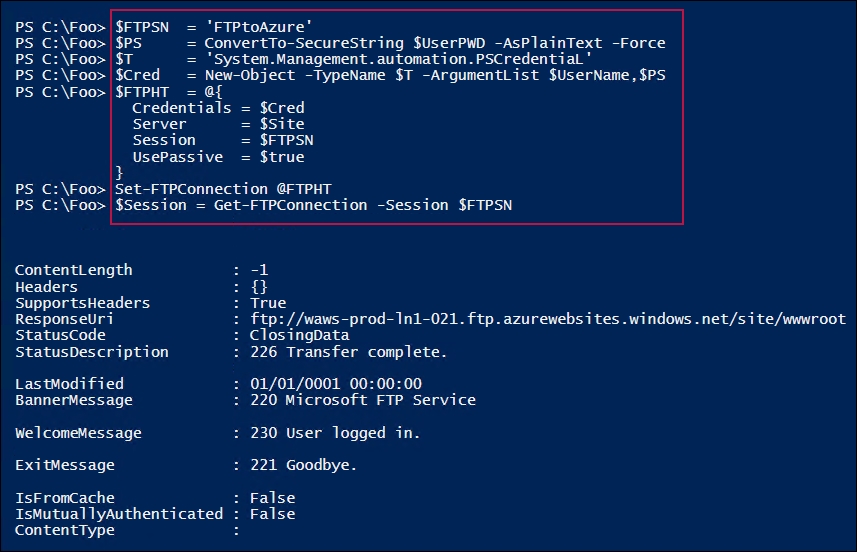
In step 9, you looked at the new website using IE, which looks like this:



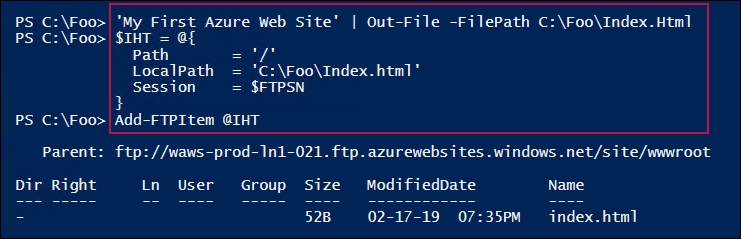
In step 10, you downloaded and installed the third-party PSFTP module, which helps to automate FTP operations. This step produces no output. In step 11, you got the Web App publishing profile, which contains the FTP credentials you need so that you can upload content to the Web App. The output of this step looks like this:



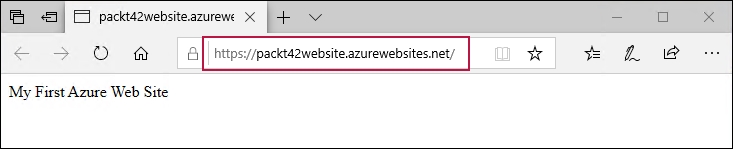
In step 12, you extracted the FTP site's credentials from the Publishing Profile, creating no output. In step 13, you used cmdlets from the PSFTP module to open an FTP session with the Azure web app. The output looks like this:



In step 14, you created a very simple web page and uploaded it to Azure. The output looks like this:



In the final step, step 15, you viewed the site in your browser, as follows:



## There's more...

In step 5, you created an application service plan. This plan basically tells Azure where to place your website and in which service tier. All Azure web apps run inside an Azure-managed VM. This recipe uses the Free Web App service tier, which is a site in a shared VM. This should provide more than adequate performance for testing and development. If you need better performance, you can scale up in terms of the processors available and scale out in terms of the number of VM instances running in your web app. For an overview of Azure App Service plans, see <https://docs.microsoft.com/azure/app-service/overview>.

## See also

This recipe uses the PSFTP third-party module. For more information about the module, see <https://gallery.technet.microsoft.com/scriptcenter/PowerShell-FTP-Client-db6fe0cb>. That web page is a little out of date and the latest version of the PSFTP module that was used in this recipe is later than shown.

# Creating an Azure VM

Azure provides a range of on-demand computing resources, one of which is virtual machines (VM). An Azure VM is a good solution where you need more control over the computing environment than you might be able to 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 2019 (or Windows 10) and Azure VMs, but they are minor. The AZ cmdlets you use to manage Azure VMs are a little different in style to Hyper-V cmdlets, which may mean a bit of a learning curve.

## Getting ready

You run this recipe on CL1, which you configured (in the Using PowerShell with Azure recipe) to work with Azure. Also, you should have already created an Azure Resource Group and an Azure Storage Account, but this recipe checks for these and creates the resources if needed.

## How to do it...

1. Define the key variables:

$Locname = 'uksouth' # Azure 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 VBM

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

$User = 'AzureAdmin' # User Name

$UserPS = 'JerryRocks42!' # User Password

$VMName = 'Packt42VM' # VM Name

1. Log in to your Azure account:

$CredAZ = Get-Credential

Login-AzAccount -Credential $CredAZ

1. Ensure that the Resource Group has been created:

$RG = Get-AzResourceGroup -Name $RgName -ErrorAction SilentlyContinue

if (-not $rg) {

$RGTag = @{Publisher = 'Packt'}

$RGTag += @{Author = 'Thomas Lee'}

$RGHT1 = @{

Name = $RgName

Location = $Locname

Tag = $RGTag

}

$RG = New-AzResourceGroup @RGHT1

Write-Host "RG $RgName created"

}

1. Ensure that the Storage Account has been created:

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

if (-not $SA) {

$SATag = [Ordered] @{Publisher = 'Packt'}

$SATag += @{Author = 'Thomas Lee'}

$SAHT - @{

Name = $SAName

ResourceGroupName = $RgName

Location = $Locname

Tag = $SATag

$SkuName = 'Standard\_LRS'

}

$SA = New-AzStorageAccount @SAHT

Write-Host "SA $SAName created"

}

1. Create the VM credentials:

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

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

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

1. Create a VM:

$VMHT = @{

ResourceGroupName = $RgName

Location = $Locname

Name = $VMName

VirtualNetworkName = $VNName

SubnetName = $CloudSN

SecurityGroupName = $NSGName

PublicIpAddressName = $IPName

OpenPorts = $Ports

Credential = $VMCred

}

New-AzVm @VMHT

1. Get and view the VM's external IP address:

$VMIP = Get-AzPublicIpAddress -ResourceGroupName $RGname

$VMIP = $VMIP.IpAddress

"VM Public IP Address: [$VMIP]"

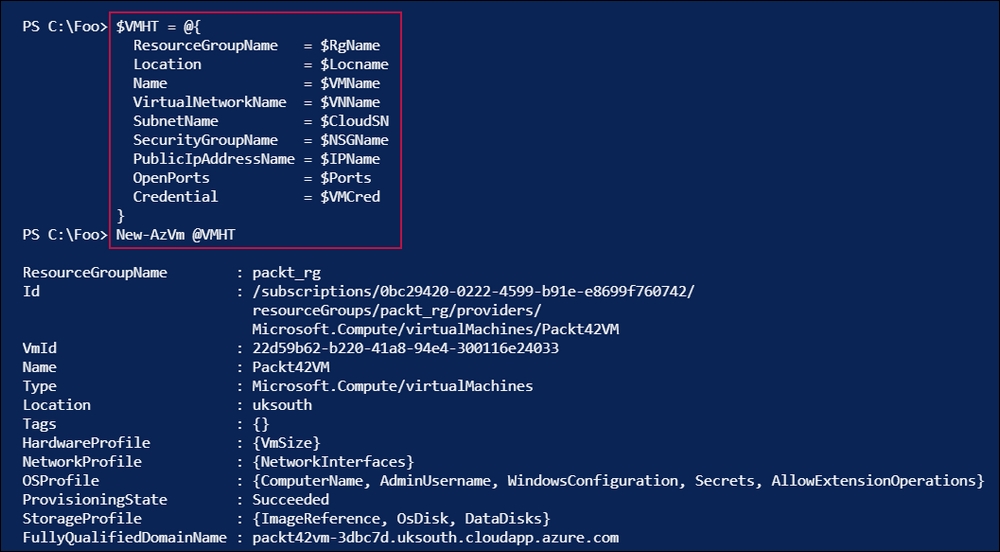
1. Connect to and view the VM:

mstsc /v:"$VMIP"

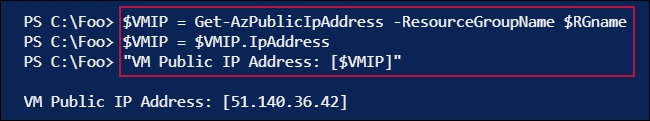
## How it works...

In step 1, you defined the variables to be used for this recipe. In step 2, you logged in to your Azure account. In step 3, you ensured that the resource group was created and, in step 4, you ensured that the Azure Storage Account for this recipe was created. In step 5, you created a PowerShell credential object that encapsulates the credentials for the VM. These steps produce no output.

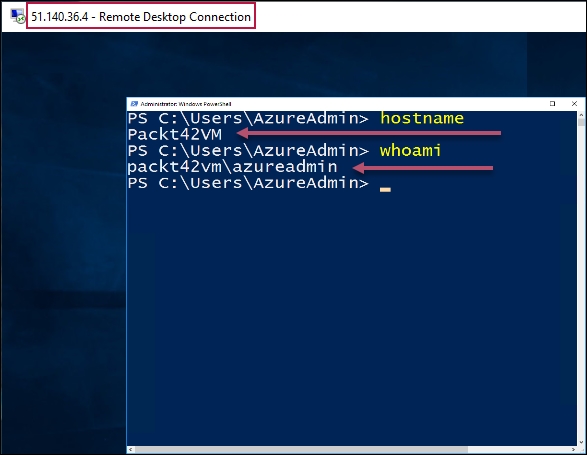
In step 6, you used the New-AzVm cmdlet to create your new VM. This produces the following output:



In step 7, you retrieved the new Azure VM's public IP address and displayed it, as follows:



In the final step, step 8, you used the mstsc.exe application to open up an RDS window into the VM, as follows:



## There's more...

In step 6, you used the New-AzVm cmdlet to create a new Azure VM. This method of creating the VM takes a number of defaults, including the version of the OS to install in the VM. This recipe takes a very simple approach to the creation of the VM—there is a lot more that you can do that is beyond the scope of this book.

## See also

There is a lot of richness in terms of the VM and the networks you can create to access it that this recipe has not covered. For more details on Azure VMs, refer to the online documentation at <https://docs.microsoft.com/azure/virtual-machines/windows/>.