PowerShell Reference Guide

SKYLINES

ACADEMY

Thank you for downloading the Skylines Academy PowerShell Reference Guide.

We're very excited to be part of your education journey! We at Skylines Academy hope this guide helps you reach success – whether it's studying for exams, or on a daily basis as an Azure administrator.

Interested in learning more? We encourage you to check out our additional courses and study groups at the following:

Training Courses: www.skylinesacademy.com

Skylines Academy

Follow on Twitter

Skylines Academy Videos

Azure Study Group

Table of Contents

POWERSHELL REFERENCE GUIDE

Introduction

Welcome to the PowerShell Reference Guide. This guide will provide you with a reference on key PowerShell commands often used by Azure administrators. The PowerShell commands are also required to pass the Azure Administrator certification exams from Microsoft.

This guide uses the recently released Azure "Az" module which replaces the AzureRM modules previously used by Microsoft. This module is intended to be more robust as it is built on .NET Standard. Microsoft currently plans to focus on building out and supporting the "Az" Module as the primary PowerShell module for interacting with Azure, a shift from the previous "AzureRM" Module. Information for supporting existing PowerShell scripts using the "AzureRM" modules is discussed below.

This guide is made up of several PowerShell commands which focus on Azure Administration and are also a core part of Microsoft Azure Administrator and Azure Arc

Note: While we make every effort to test the commands and point out any concerns when deleting objects, be sure to test these out yourself. Before running any of these commands in production, we recommend you test them out in a separate Azure test account so that you are sure you know what they are doing. Some commands are destructive in nature (e.g., removing resource groups, tags, etc.) and you need to ensure you fully understand the commands that you execute.

This guide is divided into the following sections:

- PowerShell Basics:
  - Cloud Shell
  - Downloading PowerShell and Installing Azure Az Modules for PowerShell
- Accounts and Subscriptions
- Resource Groups
- Governance
- Storage
- Virtual Machines
- Networking
- Azure Active Directory

If you spot any errors in this guide, please submit them via the Contact Us page on the Skylines Academy web site.

Thank you,

Skylines Academy Team

PowerShell Basics

At the most basic level, Azure PowerShell is designed for administrating your Azure environment. It is built upon Microsoft-extended PowerShell for Windows to include Azure modules, and underwent many iterations over time. Initially, Windows PowerShell was released and worked primarily on Windows Systems. Many server administrators would use Windows PowerShell for administration of their windows servers, Active Directory, Microsoft Exchange, etc. It was initially built on the .NET framework and you could only execute PowerShell locally from a Windows machine. With the latest releases of PowerShell Core, Microsoft has moved to .NET Core 2.x as its runtime, and now supports running on Windows, macOS, as well as Linux platforms.

If you don't already have PowerShell installed locally on your computer, then you can download the latest version for your operating system from the following links:

- Installing PowerShell Core on Windows
- Installing PowerShell Core on Linux
- Installing PowerShell Core on macOS
- Installing PowerShell Core on ARM

Azure PowerShell works with PowerShell 5.1 or higher on Windows, or PowerShell Core 6.x and later on all platforms. If you aren't sure if you have PowerShell, or are on macOS or Linux, install the latest version of PowerShell Core.

To check your PowerShell version, run the command:

```
$PSVersionTable.PSVersion
```

To run Azure PowerShell in PowerShell 5.1 on Windows:

1. Update to Windows PowerShell 5.1 if needed. If you're on Windows 10, you already have PowerShell 5.1 installed.
2. Install .NET Framework 4.7.2 or later.

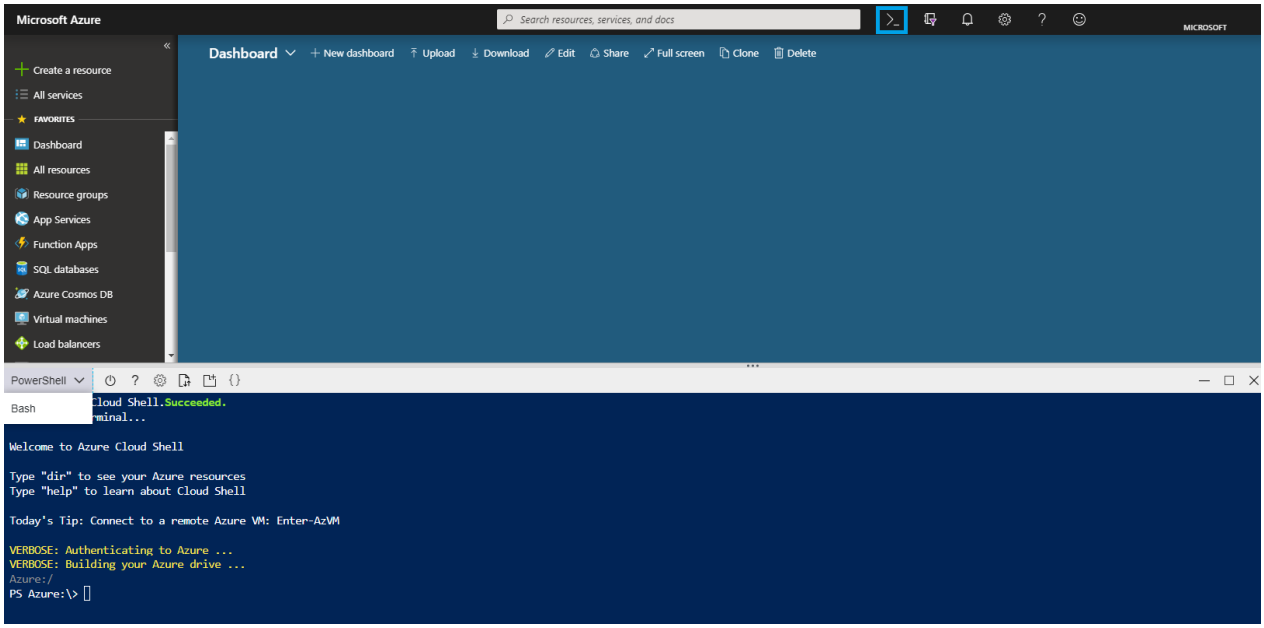
There are no additional requirements for Azure PowerShell when using PowerShell Core.

Tip: Always make sure to keep PowerShell up to date.

Cloud Shell

Before you install Azure PowerShell modules so that you can run them locally, it is worth also noting that you can now execute PowerShell from the Azure Cloud Shell. Azure Cloud Shell is a ***browser-based*** shell for managing Azure resources and provides two primary mechanisms for interacting with your Azure environment, either Bash or PowerShell. What this means is that you can execute your PowerShell commands directly from inside the Azure portal by opening up the Cloud Shell.

You can also access the shell directly by going to https://shell.Azure.com



Once you open the shell from the portal, you can now execute commands directly with instant authentication since you have already signed into the Azure Portal.

It's also important to know that while the Cloud Shell is temporary in nature, if you wish to store any scripts you create, you can do so by mounting the ***"cloud drive"*** share. You will notice the very first time you open up Cloud Shell, it will prompt you to create a resource group, storage account, and Azure Files share. This only needs to happen the first time you open up Cloud Shell and will then be automatically attached to every subsequent session you open up.

Some key concepts you should also be aware of are:

- Cloud Shell runs on a temporary host provided on a **per-session**, **per-user basis**
- Cloud Shell times out after **20 minutes** without interactive activity
- Cloud Shell **requires an Azure file share** to be mounted
- Cloud Shell uses the **same Azure file share** for both **Bash** and **PowerShell**
- Cloud Shell is assigned **one machine per user account**
- Cloud Shell persists **$HOME using a 5-GB image** held in your file share

Local Install

With all the benefits that Cloud Shell provides, you will often want to have a series of scripts that you can execute locally. Perhaps you want to administer multiple environments or complete tasks across multiple PowerShell modules. Whatever your reason, with PowerShell installed locally, you now need to install the "AZ" Module so that you can administer your Azure environment.

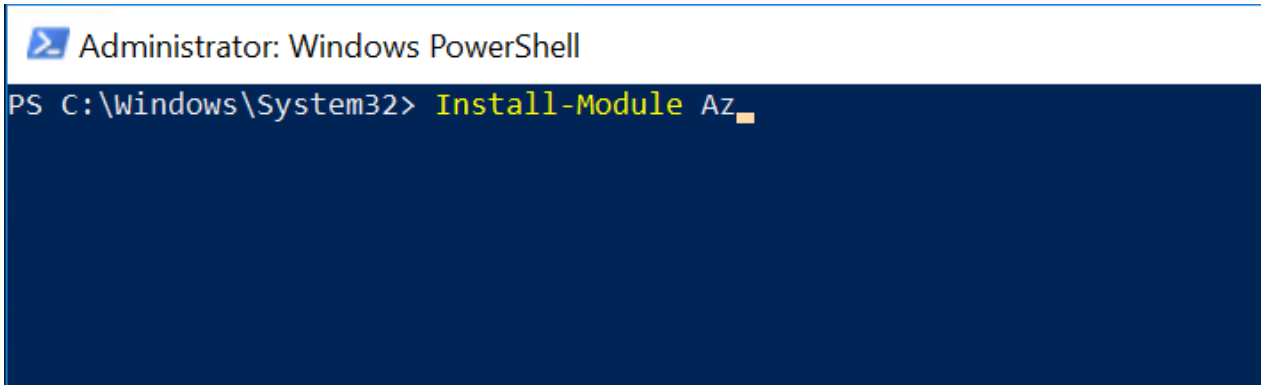
Install AZ Module for Existing AzureRM Module

If you already have AzureRM Modules installed on your computer, you'll want to uninstall the existing AzureRM Modules before installing the new Az Modules, as the modules cannot function side-by-side. You will however have the option of enabling the AzureRM alias to continue using the syntax you're comfortable with and ensure that existing PowerShell scripts continue to function properly.

Installing AZ Module (Windows Example)
Installing Azure PowerShell from the PowerShell Gallery requires elevated privileges.

Run the following command from an elevated PowerShell session (in Windows, Search for PowerShell → Right Click → Run as Administrator)

```
Install-Module Az
```



You can also be more specific in your install and choose to install for a single user, or all users. If installing for a single user, you can use the following commands:

```
Install-Module -Name Az -AllowClobber -Scope CurrentUser
```

If you want to install for all users on a system, this requires administrator privileges. From an elevated PowerShell session either run as administrator or with the sudo command on macOS or Linux:

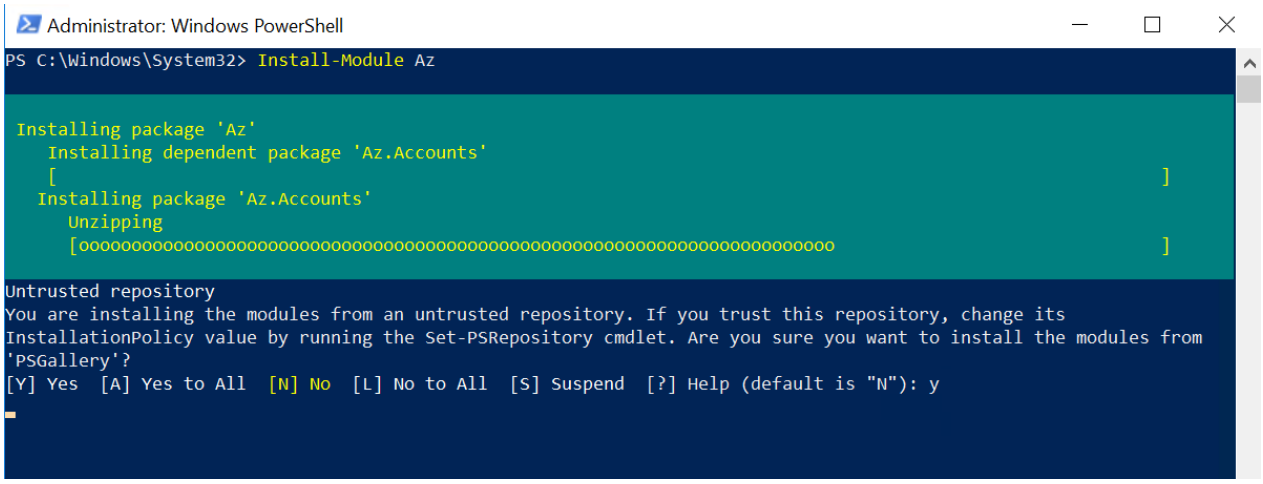
```
Install-Module -Name Az -AllowClobber -Scope AllUsers
```

When to use "AllowClobber"
You may have noticed the -AllowClobber switch in the commands above. The reason for this is to ensure that PowerShell can override any commands in existing modules. Essentially, when the Az module gets installed, it may detect existing commands already available from other modules. The clobber detection built into PowerShell will throw an error and the -AllowClobber switch allows us to override the default behavior.

Trusting the Repository

By default, the PowerShell gallery is not configured as a Trusted repository for PowerShellGet. You will see the following prompts if you have not trusted the gallery during a previous module install.

Enter **Yes to all**.



Untrusted Repository

Make sure to choose 'Yes' when prompted to install modules from the untrusted repositories. You can make these repos trusted by using the Set-PSRepository cmdlet and changing the installation policy if you desire given that the source is PSGallery.

"Are you sure you want to install the modules from 'PSGallery'?"

[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "N"): Y

Answer **'Yes'** or **'Yes to All'** to continue with the installation.

Note

If you have a version older than 2.8.5.201 of NuGet, you are prompted to download and install the latest version of NuGet+.

The Azure Az module is a rollup module for the Azure Resource Manager cmdlets. When you install the Azure Az module, any Azure PowerShell module not previously installed is downloaded and from the PowerShell Gallery.+

We are ready!

With our PowerShell Client and Azure PowerShell module setup, we should be good to go. Start by verifying you can login to your Azure account.

- Connect-AzAccount

Upon entering this command, you will be presented with a popup window to complete your login process and any MFA requirements.

Accounts and Subscriptions

Before you perform any tasks within Azure, it is important to ensure you know how to connect and disconnect from your Azure account. You will not be able to run subsequent commands if you are not connected to an Azure Account.

Azure Accounts

Task	Command	Additional Explanation
Connect to Azure Account	Connect-AzAccount	Connects to Azure with your Azure account to allow use of Azure Resource Manager Cmdlets. Note: Upon entering this command, you will be redirected to https://microsoft.com/devicelogin and presented with a popup window to complete your login process and any MFA requirements.
Disconnect from Azure Account	Disconnect-AzAccount	Terminates the session, disconnecting you from your Azure account
List Azure Tenants	Get-AzTenant	Lists all tenants which the users is authorized for
Get Specific Tenant	Get-AzTenant -TenantId xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx	Lists out a specific tenant
Enable AzureRM Alias	Enable-AzureRmAlias	Enables the use of the AzureRM alias if you are using scripts that contain AzureRM commands
Disable AzureRM Alias	Disable-AzureRmAlias	Disables the use of the AzureRM alias if you are using scripts that contain AzureRM commands

Uninstall AzureRM Modules	Uninstall-AzureRM	Remove all the AzureRM modules if you had them previously installed

Subscriptions

Once you are connected to your account, you will often want to work on or in the context of an Azure subscription where you will be placing resources. These commands show you how to retrieve a list of all your subscriptions, look at specific subscriptions in a tenant, and then select the subscription you want to work on.

List all subscriptions in all tenants the account can access	Get-AzSubscription	
Get subscriptions in a specific tenant	Get-AzSubscription -TenantId "xxxx-xxxx-xxxxxxxx"	
Choose subscription	Select-AzSubscription –SubscriptionID “SubscriptonID”	Note: Use Get-AzSubscription command above to identity the subscriptionID if you do not know it.
Create New Subscription from Enrollment account	New-AzSubscription -Name "My Subscription" -EnrollmentAccountObjectId ((Get-AzEnrollmentAccount)[0].ObjectId) -OfferType MS-XXX-0000	Advanced command used to create subscriptions off a specific enrollment account which has permissions to do so. Typically used in larger enterprises.

Resource Groups

Now that we can login to our Account and select subscriptions, we are ready to work with resources. First let's look at the commands to list out resources in our environment. These are also fantastic commands to practice with as the risk of doing anything unwanted is extremely low since you are just pulling information about your environment.

Retrieving Resource Groups

Task	Command	Additional Explanation
List All Resource Groups	Get-AzResourceGroup	Gets the resource group and additional details which can also be stored for use by additional commands.
Retrieve Specific Resource Group	Get-AzResourceGroup -Name "SkylinesRG"	Used to find a specific resource group based on name.
List All Resource Groups based on string	Get-AzResourceGroup | Where ResourceGroupName -like Skylines*	Used to find a specific resource group based on a string, but does not need to have an exact match.
Resource Group by Location	Get-AzResourceGroup | Sort Location,ResourceGroupName |	Find resource groups based on Azure region.
Resource Group by Location (Formatted as Table)	Get-AzResourceGroup | Sort Location,ResourceGroupName | Format-Table -GroupBy Location ResourceGroupName,ProvisioningState,Tags	As above with extra formatting added on.

Resource Group Provisioning & Management

Now we know how to look at resource groups, let's look at how we can create new resource groups.

Task	Command	Additional Explanation
Create a new Resource Group	New-AzResourceGroup -Name 'SkylinesRG' -Location 'northcentral'	Creates a new resource group in North Central called "Skylines RG"
Delete a Resource Group	Remove-AzResourceGroup -Name "SL-RGToDelete"	Removes a resource group and all resources contained inside it

Note: Remember even though we have to define a regional location for a resource group, this is purely to store the meta data. The resources inside the resource group can be deployed in other regions. The resource group is a management construct allowing us to enforce RBAC rights and also group resources by application lifecycle.

Resource Group Tags

As you know, the resource group itself is a management construct allowing us to enforce RBAC rights and store meta data about the resources within the group. One of the key pieces of Meta Data are tags.

They allow us to perform the following commands:

Task	Command	Additional Explanation
Display Tags associated with a specific **resource group** name	(Get-AzResourceGroup -Name "SkylinesRG").Tags	Lists out all the tags for a specific resource group.
To get all Azure **resource groups** with a specific tag	(Get-AzResourceGroup -Tag @{Owner="Skylines Academy"}).Name	Helps you locate all the resource groups with a specific tag assigned.
To get specific **resources** with a specific tag	(Get-AzResource -TagName Dept -TagValue Finance).Name	

Adding Tags

Task	Command	Additional Explanation
Add Tags to an existing resource group that has no tags	Set-AzResourceGroup -Name examplegroup -Tag @{ Dept="IT"; Environment="Test" }	Used when your resource group has NO tags currently assigned. Be careful as not to override existing tags.
Adding tags to an existing resource group that has tags 1. Get Tags 2. Append 3. Update/Apply Tags	$tags = (Get-AzResourceGroup -Name examplegroup).Tags $tags += @{Status="Approved"} Set-AzResourceGroup	Used when your resource group already has **some** existing tags.
Add tags to a specific resource without tags	$r = Get-AzResource -ResourceName examplevnet -ResourceGroupName examplegroup Set-AzResource -Tag @{ Dept="IT"; Environment="Test" } -ResourceId $r.ResourceId -Force	Tag specific resources that have NO tags currently assigned. Be careful as not to override existing tags.
Apply all tags from an existing resource group to the resources beneath. (**Note: this overrides all existing tags on the resources inside the RG)**	$groups = Get-AzResourceGroup foreach ($group in $groups) { Get-AzResource -ResourceGroupNameEquals $groups.ResourceGroupName | ForEach-Object {Set-AzResource -ResourceId $_.ResourceId -Tag $groups.Tags -Force } }	Takes the resource group tags at the parent RG and assigns them to resources inside the RG.

Apply all tags from a resource group to its resources, but retain tags on resources that are not duplicates.	$groups = Get-AzResourceGroup
	
	foreach ($g in $groups)
	{
	if ($g.Tags -ne $null) {
	$resources = Get-AzResource -ResourceGroupName $g.ResourceGroupName
	}
	
	foreach ($r in $resources)
		As above, but checks for duplicates, retaining existing tags which might be present.

```
   {
      echo "Resource :" $r
      $resourcetags = (Get-
AzResource -ResourceId
$r.ResourceId).Tags

      if ($resourcetags -ne $null)
      {
         foreach ($key in
$g.Tags.Keys)
         {
            if
($resourcetags.ContainsKey($key))
            {
               echo "Removing key : "
$key

$resourcetags.Remove($key)
            }
         }
      }
      $resourcetags += $g.Tags
      echo "Adding tags: "
$resourcetags
      Set-AzResource -Tag
$resourcetags -ResourceId
$r.ResourceId -Force
   }
}
```

Remove all tags (Caution)

You may want to remove tags. If you need to, you can use this command, but do so with caution. This command will erase all the tags from your resource groups and it is not recoverable.

Removes all tags by passing an empty hash	Set-AzResourceGroup -Tag @{} -Name exampleresourcegroup	Removes all tags. Use with caution!

Resources within RGs

What we are often concerned with are specific resources inside our Resource Group. In the Cloud world, it is not uncommon for resources to grow quickly and get out of control. These initial commands will help you find what you have inside your RG, as well as locate resources of specific types.

The following two commands are also very safe to use as they are focused on listing out resources in your environment.

Task	**Command**	**Additional Explanation**
Find resources in a resource **group** with a specific name	Get-AzResource -ResourceGroupName "SkylinesRG"	Find resources in a resource group.

Moving Resources from One Resource Group to Another

Moving resources between resource groups is another common task that you might be expected to perform. It is certainly possible from the Portal but can get tedious when you have lots of moves to complete.

Note: There are also some restrictions with Azure when moving resources between groups and you may get an error when moving certain types.

Task	**Command**	**Additional Explanation**
Step 1: Retrieve existing Resource	$Resource = Get-AzResource -ResourceType "Microsoft.ClassicCompute/storageAccounts" -ResourceName "SkylinesStorageAccount"	# Retrieves a storage account called “SkylinesStorageAccount”

Step 2: Move the Resource to the New Group	Move-AzResource -ResourceId $Resource.ResourceId -DestinationResourceGroupName "SL-NewRG" ($resourcetags.ContainsKey($key)) { $resourcetags.Remove($key) } } $resourcetags += $g.Tags Set-AzResource -Tag $resourcetags -ResourceId $r.ResourceId -Force } } }	# Moves the resource from Step 1 into the destination resource group "SL-NewRG"

Governance

Governance has become critical for users of the cloud. As mentioned before, resources can get out of control very quickly and having a complete governance strategy for your environment is essential.

Thankfully, Microsoft provides a number of free mechanisms to help enforce policies in your environment. In addition, you should also check out the Microsoft Virtual Datacenter Guide. This guide includes a complete approach for managing your Enterprise environment and is a must for large organizations.

Either way, you will need to know about Azure policies and how they can be used to help maintain your environment. Administering them is a daily job of any Azure Administrator as you deal with the demands from application teams with varying requirements.

Azure Policies: View Policies and Assignments

These commands allow you to look at your existing policies and assignments:

Task	Command	Additional Explanation
See all policy definitions in your subscription	Get-AzPolicyDefinition	Find all of your existing policy definitions. You can then assign these to resource groups or subscriptions.
Retrieve assignments for a specific resource group	$rg = Get-AzResourceGroup -Name "ExampleGroup" (Get-AzPolicyAssignment -Name accessTierAssignment -Scope $rg.ResourceId	Look up all your existing assignments for a specific resourcegroup. Using -Name allows you to narrow down to a specific policy assignment.

Creating Policies

This is a two-step process. First, you need to create your policies in JSON syntax, and then create a definition from them. In step 2 below, you will see two options for referencing the JSON policy, either via GitHub repository, or via a local file on your desktop. I encourage you to test this out and think about how you will go about managing your policy templates long term in your organization.

Task	Command	Additional Explanation
Step 1: Create JSON Policy	Create the policy in JSON	See JSON policies - https://docs.microsoft.com/en-us/azure/governance/policy/tutorials/create-custom-policy-definition

Step 2: Create Policy Definition (Local Reference)	Pass the file using Powershell. Example: $definition = New-AzPolicyDefinition ` -Name denyCoolTiering ` -Description "Deny cool access tiering for storage" ` -Policy "c:\policies\coolAccessTier.json"	Pass your JSON file using local file
Step 2: Create Policy Definition (Code Repo Reference)	$definition = New-AzPolicyDefinition ` -Name denyRegions ` -DisplayName "Deny specific regions" ` -Policy 'https://githublocation.com/azurepolicy.rules.js on'	Pass your JSON file using GitHub

Assign Policies

Now it's time to apply our policy. This involves ASSIGNING the policy to a resource group or subscription. In this example, we first retrieve our resource group and store it as a variable, then reference that variable when creating the policy assignment.

Task	**Command**	**Additional Explanation**
Assign Azure Policy	$rg = Get-AzResourceGroup -Name "ExampleGroup" New-AzPolicyAssignment -Name denyRegions - Scope $rg.ResourceId -PolicyDefinition $definition	Creates a new policy assignment to a resource group you specified by name. The policy assigned is indicated in the policy definition you would have previously created.

Resource Locks

Task	**Command**	**Additional Explanation**
Create a new resource lock	New-AzResourceLock -LockLevel ReadOnly - LockNotes "Notes about the lock" -LockName "SL- WebSiteLock" -ResourceName "SL-WebSite" ResourceType "microsoft.web/sites"	Creates a new resource lock on a specific resource. In this example, it creates a new ReadOnly resource lock on a website resource.

Retrieve a resource lock	Get-AzResourceLock -LockName "SL-WebSiteLock" -ResourceName "SL-WebSite" -ResourceType "microsoft.web/sites" -ResourceGroupName "SL-RGWebSite"	Look up a specific resource lock.

Storage

Retrieving Storage Accounts

Lists all storage accounts in the current subscription	Get-AzStorageAccount	Find all of your storage accounts – you will probably have a lot in a large environment.

Create Storage Account

Task	**Command**	**Additional Explanation**
Create Storage Account Requires the resource group name, storage account name, valid Azure location, and type (SkuName).	New-AzStorageAccount -ResourceGroupName “slstoragerg” -Name “slstorage1” -Location “eastus”-SkuName “Standard_LRS”	Creates a new storage account in a resource group. You specific the region and storage account SKU to decide on the type of account.
SKU Options	• Standard_LRS. Locally-redundant storage. • Standard_ZRS. Zone-redundant storage. • Standard_GRS. Geo-redundant storage. • Standard_RAGRS. Read access geo-redundant storage. • Premium_LRS. Premium locally-redundant storage.	
Optional Key Parameters	-Kind The kind parameter will allow you to specify the type of Storage Account. • Storage - General purpose Storage account that supports storage of Blobs, Tables, Queues, Files and Disks. • StorageV2 - General Purpose Version 2 (GPv2) Storage account that supports Blobs, Tables, Queues,	

	Files, and Disks, with advanced features like data tiering. • BlobStorage -Blob Storage account which supports storage of Blobs only. The default value is Storage. -Access Tier If you specify BlobStorage as the "Kind" then you must also include an access tier: • Hot • Cold	
Create a storage container in a storage account (using storage account name)	New-AzStorageContainer -ResourceGroupName "slstoragerg" -AccountName "slstorageaccount" - ContainerName "slContainer"	
Create a storage container in a storage account (using the storage account object)	1. Get the storage account and store it as a variable ➢ $storageaccount = Get-AzStorageAccount –ResourceGroupName "slstoragerg" -AccountName "slstorageaccount" 2. Make sure you have the right one ➢ $storageaccount This will show you the storage account object you stored in the variable $storageaccount 3. Create the container in the storage account object ➢ New-AzStorageContainer -StorageAccount $accountObject -ContainerName "slContainer" -	

Remove Accounts and Containers

Task	Command	Additional Explanation
Delete a storage account	Remove-AzStorageAccount -ResourceGroup $resourceGroup -AccountName $mystorageaccountname	Deletes your storage account
Delete a storage container using storage account name and container name	Remove-AzStorageContainer -ResourceGroupName "slstoragerg" -AccountName "slstorageaccount" - ContainerName "slcontainer"	
Delete a storage container using the storage account object	Remove-AzStorageContainer -StorageAccount $storageaccount -ContainerName "slcontainer" Note: Make sure to storage the storage account as a variable first, using: $storageaccount = Get-AzStorageAccount -ResourceGroupName "slstoragerg" -AccountName "slstorageaccount"	

Deploy and Manage Virtual Machines

Get Information About VMs

Task	Command	Additional Explanation
List all VMs in current subscription	Get-AzVM	Get all of your Azure virtual machines.
List VMs in a resource group (See Resource Groups section above)	Get -AzVM -ResourceGroupName $slResourceGroup	List all of your VMs inside a specific resource group.
Get a specific virtual machine	Get-AzVM -ResourceGroupName "slresourcegroup" -Name "myVM"	Find a specific VM by its name inside of a resource group.

Create a VM – Simplified

We put this command here as it is a quick way to create a VM, but you are far better off using VM configurations to create your VMs with more specific parameters applied. Try out both of them and you will see the difference.

Task	Command	Additional Explanation
Create a simple VM	New-AzVM -Name "vmname"	Typing in this simple command will create a VM and populate names for all the associated objects based on the VM name specified.

Create a VM Configuration Before Creating the Virtual Machine

Use the following tasks to create a new VM configuration before creating your Virtual Machine based on that config.

Task	Command	Additional Explanation
Create a VM configuration	$vmconfig = New-AzVMConfig -VMName "systemname" -VMSize "Standard_D1_v2"	Start with this step to deploy a new VM. This step creates the config.
Add configuration settings	$vmconfig = Set-AzVMOperatingSystem -VM $vmconfig -Windows -ComputerName "systemname" -Credential $cred -ProvisionVMAgent EnableAutoUpdate	This adds the operating system settings to the configuration.
Add a network interface	$vmconfig = Add-AzVMNetworkInterface -VM $vmconfig -Id $nic.Id	Adds the network interface.
Specify a platform image	$vmconfig = Set-AzVMSourceImage -VM $vmconfig -PublisherName "publisher_name" -Offer "publisher_offer" -Skus "product_sku" -Version "latest"	Chooses your OS image to use and associated SKUs.

Create a VM	New-AzVM -ResourceGroupName "slresourcegroup" -Location "eastus" -VM $vmconfigconfig All resources are created in the resource group. Before you run this command, run New-AzVMConfig, Set-AzVMOperatingSystem, Set- AzVMSourceImage, Add-AzVMNetworkInterface, and Set-AzVMOSDisk.	Finally, we create the virtual machine.

VM Operations

Task	**Command**	**Additional Explanation**
Start a VM	Start-AzVM -ResourceGroupName "slresourcegroup" -Name "vmname"	Power On
Stop a VM	Stop-AzVM -ResourceGroupName "slresourcegroup" -Name "vmname"	Power Off
Restart a running VM	Restart-AzVM -ResourceGroupName "slresourcegroup" -Name "vmname"	Soft Restart
Delete a VM	Remove-AzVM -ResourceGroupName "slresourcegroup" -Name "vmname"	Destroys the VM

Networking

Get/List Networking

Task	Command	Additional Explanation
List virtual networks	Get-AzVirtualNetwork -ResourceGroupName “slresourcegroup”	Lists all the virtual networks in the resource group.
Get information about a virtual network	Get-AzVirtualNetwork -Name "myVNet" -ResourceGroupName “slresourcegroup”	Retrieves details of a specific VNET by name.
List subnets in a virtual network	Get-AzVirtualNetwork -Name "myVNet" -ResourceGroupName “slresourcegroup” | Select Subnets	Filters down to the subnets inside of the VNET
Get information about a subnet	Get-AzVirtualNetworkSubnetConfig -Name "mySubnet1" VirtualNetwork $vnet	Gets information about the subnet in the specified virtual network. The $vnet value represents the object returned by Get-AzVirtualNetwork you used previously.
Get all IP addresses from a resource group	Get-AzPublicIpAddress -ResourceGroupName “slresourcegroup”	
Get all load balancers from a resource group	Get-AzLoadBalancer -ResourceGroupName “slresourcegroup”	
Get all network interfaces from a resource group	Get-AzNetworkInterface -ResourceGroupName “slresourcegroup”	
Get information about a network interface	Get-AzNetworkInterface -Name "slNIC" -ResourceGroupName “slresourcegroup”	
Get the IP configuration of a network interface	Get-AzNetworkInterfaceIPConfig -Name "slNICIP" -NetworkInterface $nic	

Create Network Resources

Task	Command	Additional Explanation
Create subnet configurations	$subnet1 = New-AzVirtualNetworkSubnetConfig -Name "slSubnet1" -AddressPrefix XX.X.X.X/XX $subnet2 = New-AzVirtualNetworkSubnetConfig -Name "slSubnet2" AddressPrefix XX.X.X.X/XX	
Create a virtual network	$vnet = New-AzVirtualNetwork -Name "myVNet" -ResourceGroupName "slresourcegroup" -Location $location -AddressPrefix XX.X.X.X/XX -Subnet $slsubnet1, $slsubnet2	Note: Make sure to create the subnets first as per the previous command above.
Test for a unique domain name	Test-AzDnsAvailability -DomainNameLabel "myDNS" -Location $location	You can specify a DNS domain name for a public IP resource, which creates a mapping for domainname.location.cloudapp.azure.com to the public IP address in the Azure-managed DNS servers. The name can contain only letters, numbers, and hyphens. The first and last character must be a letter or number and the domain name must be unique within its Azure location. If **True** is returned, your proposed name is globally unique.
Create a public IP address	$pip = New-AzPublicIpAddress -Name "myPublicIp" -ResourceGroupName "slresourcegroup" -DomainNameLabel "myDNS" -Location $location AllocationMethod Dynamic	The public IP address uses the domain name that you previously tested and is used by the frontend configuration of the load balancer.

Create a frontend IP configuration	$frontendIP = New-AzLoadBalancerFrontendIpConfig -Name "myFrontendIP" PublicIpAddress $pip	The frontend configuration includes the public IP address that you previously created for incoming network traffic.
Create a backend address pool	$beAddressPool = New-AzLoadBalancerBackendAddressPoolConfig -Name "myBackendAddressPool"	Provides internal addresses for the backend of the load balancer that are accessed through a network interface.
Create a probe	$healthProbe = New-AzLoadBalancerProbeConfig -Name "myProbe" RequestPath 'HealthProbe.aspx' -Protocol http -Port 80 -IntervalInSeconds 15 ProbeCount 2	Contains health probes used to check availability of virtual machines instances in the backend address pool.
Create a load balancing rule	$lbRule = New-AzLoadBalancerRuleConfig -Name HTTP -FrontendIpConfiguration $frontendIP -BackendAddressPool $beAddressPool -Probe $healthProbe -Protocol Tcp -FrontendPort 80 -BackendPort 80	Contains rules that assign a public port on the load balancer to a port in the backend address pool.

Create an inbound NAT rule	$inboundNATRule = New-AzLoadBalancerInboundNatRuleConfig -Name "myInboundRule1" -FrontendIpConfiguration $frontendIP -Protocol TCP -FrontendPort 3441 -BackendPort 3389	Contains rules mapping a public port on the load balancer to a port for a specific virtual machine in the backend address pool.

Create a load balancer	$loadBalancer = New-AzLoadBalancer -ResourceGroupName “slresourcegroup” -Name "myLoadBalancer" -Location $location -FrontendIpConfiguration $frontendIP InboundNatRule $inboundNATRule -LoadBalancingRule $lbRule -BackendAddressPool $beAddressPool -Probe $healthProbe	
Create a network interface	$nic1= New-AzNetworkInterface -ResourceGroupName “slresourcegroup” Name "myNIC" -Location $location -PrivateIpAddress XX.X.X.X -Subnet $subnet2 -LoadBalancerBackendAddressPool $loadBalancer.BackendAddressPools[0] - LoadBalancerInboundNatRule $loadBalancer.InboundNatRules[0]	Create a network interface using the public IP address and virtual network subnet that you previously created.

Remove Network Resources

Task	**Command**	**Additional Explanation**
Delete a virtual network	Remove-AzVirtualNetwork -Name "myVNet" -ResourceGroupName “slresourcegroup”	Removes the specified virtual network from the resource group.
Delete a network interface	Remove-AzNetworkInterface -Name "myNIC" -ResourceGroupName “slresourcegroup”	Removes the specified network interface from the resource group.
Delete a load balancer	Remove-AzLoadBalancer -Name "myLoadBalancer" -ResourceGroupName “slresourcegroup”	Removes the specified load balancer from the resource group.

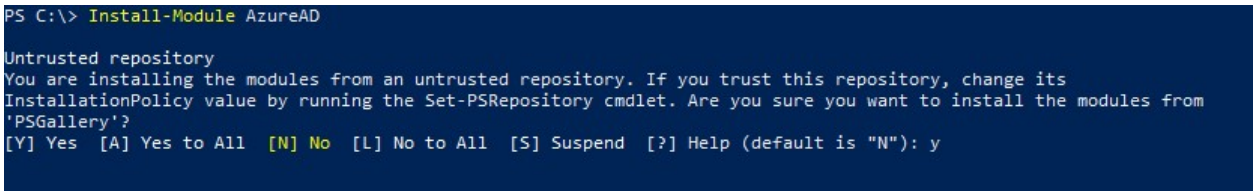
Delete a public IP address	Remove-AzPublicIpAddress-Name "myIPAddress" -ResourceGroupName "slresourcegroup"	Removes the specified public IP address from the resource group.

Azure Active Directory Commands

Install Azure AD Module

In order to use the Azure AD commands, you first need to install the Azure AD module. Use the following procedure to get it installed:

1. Open PowerShell
2. Type "Install-Module AzureAD"
3. Press Y to accept the untrusted repository (PSGallery).



Connect to Azure AD

Task	Command	Additional Explanation
Connect to Azure Active Directory	Connect-AzureAD	Note: You will be prompted to enter your credentials and any additional authentication steps required.
Disconnect from Azure Active Directory	Disconnect-AzureAD	

User and Service Principal Management

Task	Command	Additional Explanation
Get all users	Get-AzureADUser	
Get specific user	Get-AzureADUser -ObjectId "user@skylinesexam.com"	
Remove User	Remove-AzureADUser -ObjectId "user@skylinesexam.com"	

New User Creation	1. Create Password Profile $PasswordProfile = New-Object -TypeName Microsoft.Open.AzureAD.Model.PasswordProfile 2. Set Password $PasswordProfile.Password = "Password" 3. Create User New-AzureADUser -DisplayName "New User" -PasswordProfile $PasswordProfile -UserPrincipalName "user@contoso.com" -AccountEnabled $true -MailNickName "Newuser"	This is a three-step process that requires first creating a password profile, setting the password, and then passing these into the New-AzureADUser command
Service Principal Creation	First you need to create your application registration in AzureAD then you retrieve it with this command. Get-AzADApplication -DisplayNameStartWith slappregistration Once you have the application ID for the App registration, you can use it to create the SPN (Service Principal) New-AzADServicePrincipal -ApplicationId 11111111-1111-1111-1111-11111111111 -Password $securePassword	
Assign Role	New-AzRoleAssignment -ResourceGroupName “slresourcegroup” -ObjectId 11111111-1111-1111-1111-11111111111 -RoleDefinitionName Reader	This will be scoped to the resource group name you type in with the role definition assigned to the SPN In other words, the SPN is allowed to do X at the RG named Y.
View Current Role Assignment	Get-AzRoleAssignment -ResourceGroupName “slresourcegroup” -ObjectId 11111111-1111-1111-1111-11111111111	