Contents

Release notes

Breaking changes for Az 1.0.0

Overview of Azure PowerShell

5/6/2019 • 2 minutes to read • Edit Online

Azure PowerShell provides a set of cmdlets that use the Azure Resource Manager model for managing your Azure resources. Azure PowerShell uses .NET Standard, making it available for Windows, macOS, and Linux. Azure PowerShell is also available on Azure Cloud Shell.

About the new Az module

This documentation describes the new Az module for Azure PowerShell. This new module is written from the ground up in .NET Standard. Using .NET Standard allows Azure PowerShell to run under PowerShell 5 on Windows or PowerShell 6 on any platform. The Az module is now the intended way to interact with Azure through PowerShell. AzureRM will continue to get bug fixes, but no longer receive new features.

Learn the full details about the new module, including how commands have been renamed and the maintenance plans for AzureRM, in the Introducing the Azure PowerShell Az module. If you want to get started with using the new module right away, see Migrate from AzureRM to Az.

The AzureRM documentation is also available.

> **IMPORTANT**
>
> While the Azure documentation is being updated to reflect the new module cmdlet names, articles may still use the AzureRM commands. After installing the Az module, it's recommended that you enable the AzureRM cmdlet aliases with `Enable-AzureRmAlias`. See the Migrate from AzureRM to Az article for more details.

Run or install

You can install Azure PowerShell on PowerShell 5.1 or higher on Windows, PowerShell 6 on any platform, or run in Azure Cloud Shell.

- To run in your browser with Azure Cloud Shell, see Quickstart for PowerShell in Azure Cloud Shell.
- To install Azure PowerShell on your system, see Install Azure PowerShell.

For information about the latest Azure PowerShell release, see the release notes.

Get Started

Read the Get Started with Azure PowerShell article to learn the Azure PowerShell basics. If you're not familiar with PowerShell, an introduction might be helpful:

- Install PowerShell
- Scripting with PowerShell
- PowerShell Basics: (Part 1) Getting Started with PowerShell
- Microsoft Virtual Academy's Getting Started with PowerShell Jumpstart

The following samples can help you with some common uses of Azure:

- Linux Virtual Machines
- Windows Virtual Machines
- Web Apps

- SQL Databases

Build your skills with Microsoft Learn

- Automate Azure tasks using scripts with PowerShell
- More interactive learning...

Other Azure PowerShell modules

- Azure Active Directory
- Azure Service Fabric
- Azure ElasticDB

Introducing the new Azure PowerShell Az module

5/16/2019 • 3 minutes to read • Edit Online

Starting in December 2018, the Azure PowerShell Az module is in general release and now the intended PowerShell module for interacting with Azure. Az offers shorter commands, improved stability, and cross-platform support. Az also offers feature parity and an easy migration path from AzureRM.

With the Az module, Azure PowerShell is now compatible with PowerShell 5.1 on Windows and PowerShell Core 6.x and later on all supported platforms - including Windows, macOS, and Linux.

Az is a new module, so the version has been reset to 1.0.0.

Why a new module?

Major updates can be inconvenient, so it's important that we let you know why the decision was made to introduce a new set of modules, with new cmdlets, for interacting with Azure from PowerShell.

The biggest and most important change is that PowerShell has been a cross-platform product since the introduction of PowerShell Core 6.x, based on the .NET Standard library. We're committed to bringing Azure support to all platforms, which means that the Azure PowerShell modules needed to be updated to use .NET Standard and be compatible with PowerShell Core. Rather than taking the existing AzureRM module and introduce complex changes to add this support, the Az module was created.

Creating a new module also gave our engineers the opportunity to make the design and naming of cmdlets and modules consistent. All modules now start with the `Az.` prefix and cmdlets all use the *Verb*-`Az`*Noun* form. Previously, cmdlet names were not only longer, there were inconsistencies in cmdlet names.

The number of modules was also reduced: Some modules which worked with the same services have been rolled together, and magement plane and data plane cmdlets are now contained all within single modules for their services. For those of you who manually manage dependencies and imports, this makes things much simpler.

By making these important changes that required building a Azure PowerShell module, the team has committed to making it easier than ever, and on more platforms than previously possible, to use Azure with PowerShell cmdlets.

Upgrade to Az

To keep up with the latest Azure features in PowerShell, you should migrate to the Az module as soon as possible. If you're not ready to install the Az module as a replacement for AzureRM, you have a couple of options available to experiment with Az:

- Use a `PowerShell` environment with Azure Cloud Shell. Azure Cloud Shell is a browser-based shell environment which comes with the Az module installed and `Enable-AzureRM` compatibility aliases enabled.
- Keep the AzureRM module installed with PowerShell 5.1 for Windows, but install the Az module for PowerShell Core 6.x or later. PowerShell 5.1 for Windows and PowerShell Core use separate collections of modules. Follow the instructions to install PowerShell Core and then install the Az module from a PowerShell Core terminal.

To upgrade from an existing AzureRM install:

1. Uninstall the Azure PowerShell AzureRM module
2. Install the Azure PowerShell Az module
3. **OPTIONAL**: Enable compatibility mode to add aliases for AzureRM cmdlets with Enable-AzureRMAlias while you become familiar with the new command set.

Migrate existing scripts to Az

The Az module has a compatibility mode to help you use existing scripts while you work on updates to the new syntax. Use the Enable-AzureRmAlias cmdlet to enable the AzureRM compatibility mode. This cmdlet defines AzureRM cmdlet names as aliases for the new Az cmdlet names.

> **IMPORTANT**
> Even though the cmdlet names are aliased, there may still be new (or renamed) parameters for the Az cmdlets that will be exposed. Don't expect enabling aliases to take care of the full migration for you! After enabling them, test your scripts to make sure they don't need any immediate updates.

The new cmdlet names have been designed to be easy to learn. Instead of using `AzureRm` or `Azure` in cmdlet names, use `Az`. For example, the old command `New-AzureRMVm` has become `New-AzVm`.

For a full description of the migration process, see Migrate from AzureRM to Az.

Continued support for AzureRM

The existing AzureRM module will no longer receive new cmdlets or features. However, AzureRM is still officially maintained and will get bug fixes up through at least December 2020.

The Az module is production-ready, in GA, and does not incur any additional costs on Azure. We recommend that you switch to the Az module as soon as possible to keep up with the latest Azure features.

Install the Azure PowerShell module

5/15/2019 • 3 minutes to read • Edit Online

This article tells you how to install the Azure PowerShell modules using PowerShellGet. These instructions work on Windows, macOS, and Linux platforms. For the Az module, currently no other installation methods are supported.

Requirements

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 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.

Install the Azure PowerShell module

> **IMPORTANT**
> You can have both the AzureRM and Az modules installed at the same time. If you have both modules installed, **don't enable aliases**. Enabling aliases will cause conflicts between AzureRM cmdlets and Az command aliases, and could cause unexpected behavior. It's recommended that before installing the Az module, you uninstall AzureRM. You can always uninstall AzureRM or enable aliases at any time. To learn about the AzureRM command aliases, see Migrate from AzureRM to Az. For uninstall instructions, see Uninstall the AzureRM module.

To install modules at a global scope, you need elevated privileges to install modules from the PowerShell Gallery. To install Azure PowerShell, run the following command in an elevated session ("Run as Administrator" on Windows, or with superuser privileges on macOS or Linux):

```
Install-Module -Name Az -AllowClobber
```

If you don't have access to administrator privileges, you can install for the current user by adding the `-Scope` argument.

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

By default, the PowerShell gallery isn't configured as a trusted repository for PowerShellGet. The first time you use the PSGallery you see the following prompt:

```
Untrusted repository

You are installing the modules from an untrusted repository. If you trust this repository, change
its InstallationPolicy value by running the Set-PSRepository cmdlet.

Are you sure you want to install the modules from 'PSGallery'?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "N"):
```

Answer `Yes` or `Yes to All` to continue with the installation.

The Az module is a rollup module for the Azure PowerShell cmdlets. Installing it downloads all of the available Azure Resource Manager modules, and makes their cmdlets available for use.

Sign in

To start working with Azure PowerShell, sign in with your Azure credentials.

```
# Connect to Azure with a browser sign in token
Connect-AzAccount
```

> **NOTE**
> If you've disabled module autoloading, you need to manually import the module with `Import-Module Az`. Because of the way the module is structured, this can take a few seconds.

You'll need to repeat these steps for every new PowerShell session you start. To learn how to persist your Azure sign-in across PowerShell sessions, see Persist user credentials across PowerShell sessions.

Update the Azure PowerShell module

You can update your Azure PowerShell installation by running Update-Module. This command does **not** uninstall older versions.

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

To learn how to remove old versions of Azure PowerShell from your system, see Uninstall the Azure PowerShell module.

Use multiple versions of Azure PowerShell

It's possible to install more than one version of Azure PowerShell. To check if you have multiple versions of Azure PowerShell installed, use the following command:

```
Get-InstalledModule -Name Az -AllVersions | select Name,Version
```

To remove a version of Azure PowerShell, see Uninstall the Azure PowerShell module.

You can install or load a specific version of the `Az` module by using the `-RequiredVersion` argument:

```
# Install Az version 0.7.0
Install-Module -Name Az -RequiredVersion 0.7.0
# Load Az version 0.7.0
Import-Module -Name Az -RequiredVersion 0.7.0
```

If you have more than one version of the module installed, module autoload and `Import-Module` load the latest version by default.

Provide feedback

If you find a bug in Azure Powershell, file an issue on GitHub. To provide feedback from the command line, use the Send-Feedback cmdlet.

Next Steps

To learn more about the Azure PowerShell modules and their features, see Get Started with Azure PowerShell. If you're familiar with Azure PowerShell and need to migrate from AzureRM, see Migrate from AzureRM to Az.

Uninstall the Azure PowerShell module

5/16/2019 • 3 minutes to read • Edit Online

This article tells you how to uninstall an older version of Azure PowerShell, or completely remove it from your system. If you've decided to completely uninstall the Azure PowerShell, give us some feedback through the Send-Feedback cmdlet. If you encountered a bug, we'd appreciate it if you file a GitHub issue so that it can be fixed.

Uninstall the Az module

To uninstall the Az modules, use the Uninstall-Module cmdlet. However, `Uninstall-Module` only uninstalls one module. To remove Azure PowerShell completely, you must uninstall each module individually. Uninstallation can be complicated if you have more than one version of Azure PowerShell installed.

To check which versions of Azure PowerShell you currently have installed, run the following command:

```
Get-InstalledModule -Name Az -AllVersions
```

```
Version              Name                           Repository           Description
-------              ----                           ----------           -----------
0.7.0                Az                             PSGallery            Azure Resource Manager Module
1.0.0                Az                             PSGallery            Azure Resource Manager Module
```

The following script queries the PowerShell Gallery to get a list of dependent submodules. Then, the script uninstalls the correct version of each submodule. You will need to have administrator access to run this script in a scope other than `Process` or `CurrentUser`.

```
function Uninstall-AllModules {
  param(
    [Parameter(Mandatory=$true)]
    [string]$TargetModule,

    [Parameter(Mandatory=$true)]
    [string]$Version,

    [switch]$Force,

    [switch]$WhatIf
  )

  $AllModules = @()

  'Creating list of dependencies...'
  $target = Find-Module $TargetModule -RequiredVersion $version
  $target.Dependencies | ForEach-Object {
    if ($_.requiredVersion) {
      $AllModules += New-Object -TypeName psobject -Property @{name=$_.name; version=$_.requiredVersion}
    }
    else { # Assume minimum version
      # Minimum version actually reports the installed dependency
      # which is used, not the actual "minimum dependency." Check to
      # see if the requested version was installed as a dependency earlier.
      $candidate = Get-InstalledModule $_.name -RequiredVersion $version
      if ($candidate) {
        $AllModules += New-Object -TypeName psobject -Property @{name=$_.name; version=$version}
      }
      else {
        Write-Warning ("Could not find uninstall candidate for {0}:{1} - module may require manual uninstall"
-f $_.name,$version)
      }
    }
  }
  $AllModules += New-Object -TypeName psobject -Property @{name=$TargetModule; version=$Version}

  foreach ($module in $AllModules) {
    Write-Host ('Uninstalling {0} version {1}...' -f $module.name,$module.version)
    try {
      Uninstall-Module -Name $module.name -RequiredVersion $module.version -Force:$Force -ErrorAction Stop -
WhatIf:$WhatIf
    } catch {
      Write-Host ("`t" + $_.Exception.Message)
    }
  }
}
```

To use this function, copy and paste the code into your PowerShell session. The following example shows how to run the function to remove an older version of Azure PowerShell.

```
Uninstall-AllModules -TargetModule Az -Version 0.7.0 -Force
```

As the script runs, it will display the name and version of each submodule that is being uninstalled. To run the script to only see what would be deleted, without removing it, use the `-WhatIf` option.

```
Creating list of dependencies...
Uninstalling Az.Profile version 0.7.0
Uninstalling Az.Aks version 0.7.0
Uninstalling Az.AnalysisServices version 0.7.0
...
```

Run this command for every version of Azure PowerShell that you want to uninstall. For convenience, the following script will uninstall all versions of Az **except** for the latest.

```
$versions = (Get-InstalledModule Az -AllVersions | Select-Object Version)
$versions[1..($versions.Length-1)]  | foreach { Uninstall-AllModules -TargetModule Az -Version ($_.Version) -
Force }
```

Uninstall the AzureRM module

If you have the Az module installed on your system and would like to uninstall AzureRM, there are two options that don't require running the `Uninstall-AllModules` script above. Which method you follow depends on how you installed the AzureRM module. If you're not sure of your original install method, follow the steps for uninstalling an MSI first.

Uninstall Azure PowerShell MSI

If you installed the Azure PowerShell AzureRM modules using the MSI package, you must uninstall through the Windows system rather than PowerShell.

PLATFORM	INSTRUCTIONS
Windows 10	Start > Settings > Apps
Windows 7 Windows 8	Start > Control Panel > Programs > Uninstall a program

Once on this screen you should see **Azure PowerShell** in the program listing. This is the app to uninstall. If you don't see this program listed, then you installed through PowerShellGet, and should follow the next set of instructions.

Uninstall from PowerShell

If you installed AzureRM with PowerShellGet, then you can remove the modules with the Uninstall-AzureRM command, available as part of the `Az.Accounts` module. This removes *all* AzureRM modules from your machine, but requires administrator privileges.

```
Uninstall-AzureRm
```

If you can't successfully run the `Uninstall-AzureRM` command, use the `Uninstall-AllModules` script provided in this article, with the following invocation:

```
$versions = (Get-InstalledModule AzureRM -AllVersions | Select-Object Version)
$versions | foreach { Uninstall-AllModules -TargetModule AzureRM -Version ($_.Version) -Force }
```

Get started with Azure PowerShell

5/6/2019 • 3 minutes to read • Edit Online

Azure PowerShell is designed for managing and administering Azure resources from the command line. Use Azure PowerShell when you want to build automated tools that use the Azure Resource Manager model. Try it out in your browser with Azure Cloud Shell, or install on your local machine.

This article helps you get started with Azure PowerShell and teaches the core concepts behind it.

Install or run in Azure Cloud Shell

The easiest way to get started with Azure PowerShell is by trying it out in an Azure Cloud Shell environment. To get up and running with Cloud Shell, see Quickstart for PowerShell in Azure Cloud Shell. Cloud Shell runs PowerShell 6 on a Linux container, so Windows-specific functionality isn't available.

When you're ready to install Azure PowerShell on your local machine, follow the instructions in Install the Azure PowerShell module.

Sign in to Azure

Sign in interactively with the `Connect-AzAccount` cmdlet. Skip this step if you use Cloud Shell: Your Azure Cloud Shell session is already authenticated for the environment, subscription, and tenant that launched the Cloud Shell session.

```
Connect-AzAccount
```

If you're in a non-US region, use the `-Environment` parameter to sign in. Get the name of the environment for your region by using the Get-AzEnvironment cmdlet. For example, to sign in to Azure China 21Vianet:

```
Connect-AzAccount -Environment AzureChinaCloud
```

You'll get a token to use on https://microsoft.com/devicelogin. Open this page in your browser and enter the token, then sign in with your Azure account credentials and authorize Azure PowerShell.

After signing in, you'll see information indicating which of your Azure subscriptions is active. If you have multiple Azure subscriptions in your account and want to select a different one, get your available subscriptions with Get-AzSubscription and use the Set-AzContext cmdlet with your subscription ID. For more information about managing your Azure subscriptions in Azure PowerShell, see Use multiple Azure subscriptions.

Once signed in, use the Azure PowerShell cmdlets to access and manage resources in your subscription. To learn more about the sign-in process and authentication methods, see Sign in with Azure PowerShell.

Find commands

Azure PowerShell cmdlets follow a standard naming convention for PowerShell, `VERB-NOUN`. The verb describes the action (examples include `Create`, `Get`, `Set`, `Delete`) and the noun describes the resource type (examples include `AzVM`, `AzKeyVaultCertificate`, `AzFirewall`, `AzVirtualNetworkGateway`). Nouns in Azure PowerShell always start with the prefix `Az`. For the full list of standard verbs, see Approved verbs for PowerShell Commands.

Knowing the nouns, verbs, and the Azure PowerShell modules available help you find commands with the Get-

Command cmdlet. For example, to find all VM-related commands that use the `Get` verb:

```
Get-Command -Verb Get -Noun AzVM* -Module Az.Compute
```

To help you find common commands, this table lists the resource type, corresponding Azure PowerShell module, and noun prefix to use with `Get-Command`:

RESOURCE TYPE	AZURE POWERSHELL MODULE	NOUN PREFIX
Resource group	Az.Resources	`AzResourceGroup`
Virtual machines	Az.Compute	`AzVM`
Storage accounts	Az.Storage	`AzStorageAccount`
Key Vault	Az.KeyVault	`AzKeyVault`
Web applications	Az.Websites	`AzWebApp`
SQL databases	Az.Sql	`AzSqlDatabase`

For a full list of the modules in Azure PowerShell, see the Azure PowerShell modules list hosted on GitHub.

Learn Azure PowerShell basics with quickstarts and tutorials

To get started with Azure PowerShell, try an in-depth tutorial for setting up virtual machines and learning how to query them.

Create virtual machines with Azure PowerShell

There are also Azure PowerShell quickstarts for other popular Azure services:

- Create a storage account
- Transfer objects to/from Azure Blob storage
- Create and retrieve secrets from Azure Key Vault
- Create an Azure SQL database and firewall
- Run a container in Azure Container Instances
- Create a Virtual Machine Scale Set (VMSS)
- Create a standard load balancer

Next steps

- Sign in with Azure PowerShell
- Manage Azure subscriptions with Azure PowerShell
- Create service principals with Azure PowerShell
- Get help from the community:
  - Azure forum on MSDN
  - Stack Overflow

Sign in with Azure PowerShell

5/6/2019 • 3 minutes to read • Edit Online

Azure PowerShell supports several authentication methods. The easiest way to get started is with Azure Cloud Shell, which automatically logs you in. With a local install, you can sign in interactively through your browser. When writing scripts for automation, the recommended approach is to use a service principal with the necessary permissions. When you restrict sign-in permissions as much as possible for your use case, you help keep your Azure resources secure.

After signing in, commands are run against your default subscription. To change your active subscription for a session, use the Set-AzContext cmdlet. To change the default subscription used when logging in with Azure PowerShell, use Set-AzDefault.

> **IMPORTANT**
> Your credentials are shared among multiple PowerShell sessions as long as you remain signed in. For more information, see the article on Persistent Credentials.

Sign in interactively

To sign in interactively, use the Connect-AzAccount cmdlet.

```
Connect-AzAccount
```

When run, this cmdlet will present a token string. To sign in, copy this string and paste it into https://microsoft.com/devicelogin in a browser. Your PowerShell session will be authenticated to connect to Azure.

> **IMPORTANT**
> Username/password credential authorization has been removed in Azure PowerShell due to changes in Active Directory authorization implementations and security concerns. If you use credential authorization for automation purposes, instead create a service principal.

Sign in with a service principal

Service principals are non-interactive Azure accounts. Like other user accounts, their permissions are managed with Azure Active Directory. By granting a service principal only the permissions it needs, your automation scripts stay secure.

To learn how to create a service principal for use with Azure PowerShell, see Create an Azure service principal with Azure PowerShell.

To sign in with a service principal, use the `-ServicePrincipal` argument with the `Connect-AzAccount` cmdlet. You'll also need the service principal's application ID, sign-in credentials, and the tenant ID associate with the service principal. How you sign in with a service principal will depend on whether it's configured for password-based or certificate-based authentication.

Password-based authentication

To get the service principal's credentials as the appropriate object, use the Get-Credential cmdlet. This cmdlet will

present a prompt for a username and password. Use the service principal ID for the username.

```
$pscredential = Get-Credential
Connect-AzAccount -ServicePrincipal -Credential $pscredential -TenantId $tenantId
```

For automation scenarios, you need to create credentials from a user name and secure string:

```
$passwd = ConvertTo-SecureString <use a secure password here> -AsPlainText -Force
$pscredential = New-Object System.Management.Automation.PSCredential('service principal name/id', $passwd)
Connect-AzAccount -ServicePrincipal -Credential $pscredential -TenantId $tenantId
```

Make sure that you use good password storage practices when automating service principal connections.

Certificate-based authentication

Certificate-based authentication requires that Azure PowerShell can retrieve information from a local certificate store based on a certificate thumbprint.

```
Connect-AzAccount -ServicePrincipal -TenantId $tenantId -CertificateThumbprint <thumbprint>
```

In PowerShell 5, the certificate store can be managed and inspected with the PKI module. For PowerShell 6, the process is more complicated. The following scripts show you how to import an existing certificate into the certificate store accessible by PowerShell.

Import a certificate in PowerShell 5

```
# Import a PFX
$credentials = Get-Credential -Message "Provide PFX private key password"
Import-PfxCertificate -FilePath <path to certificate> -Password $credentials.Password -CertStoreLocation
cert:\CurrentUser\My
```

Import a certificate in PowerShell 6

```
# Import a PFX
$storeName = [System.Security.Cryptography.X509Certificates.StoreName]::My
$storeLocation = [System.Security.Cryptography.X509Certificates.StoreLocation]::CurrentUser
$store = [System.Security.Cryptography.X509Certificates.X509Store]::new($storeName, $storeLocation)
$certPath = <path to certificate>
$credentials = Get-Credential -Message "Provide PFX private key password"
$flag = [System.Security.Cryptography.X509Certificates.X509KeyStorageFlags]::Exportable
$certificate = [System.Security.Cryptography.X509Certificates.X509Certificate2]::new($certPath,
$credentials.Password, $flag)
$store.Open([System.Security.Cryptography.X509Certificates.OpenFlags]::ReadWrite)
$store.Add($Certificate)
$store.Close()
```

Sign in using a managed identity

Managed identities are a feature of Azure Active Directory. Managed identities are service principals assigned to resources that run in Azure. You can use a managed identity service principal for sign-in, and acquire an app-only access token to access other resources. Managed identities are only available on resources running in an Azure cloud.

To learn more about managed identities for Azure resources, see How to use managed identities for Azure resources on an Azure VM to acquire an access token.

Sign in with a non-default tenant or as a Cloud Solution Provider (CSP)

If your account is associated with more than one tenant, sign-in requires the use of the `-TenantId` parameter when connecting. This parameter will work with any other sign-in method. When logging in, this parameter value can either be the Azure object ID of the tenant (Tenant ID) or the fully qualified domain name of the tenant.

If you're a Cloud Solution Provider (CSP), the `-TenantId` value **must** be a tenant ID.

```
Connect-AzAccount -TenantId 'xxxx-xxxx-xxxx-xxxx'
```

Sign in to another Cloud

Azure cloud services offer environments compliant with regional data-handling laws. For accounts in a regional cloud, set the environment when you sign in with the `-Environment` argument. For example, if your account is in the China cloud:

```
Connect-AzAccount -Environment AzureChinaCloud
```

The following command gets a list of available environments:

```
Get-AzEnvironment | Select-Object Name
```

Create an Azure service principal with Azure PowerShell

5/6/2019 • 5 minutes to read • Edit Online

Automated tools that use Azure services should always have restricted permissions. Instead of having applications sign in as a fully privileged user, Azure offers service principals.

An Azure service principal is an identity created for use with applications, hosted services, and automated tools to access Azure resources. This access is restricted by the roles assigned to the service principal, giving you control over which resources can be accessed and at which level. For security reasons, it's always recommended to use service principals with automated tools rather than allowing them to log in with a user identity.

This article shows you the steps for creating, getting information about, and resetting a service principal with Azure PowerShell.

Create a service principal

Create a service principal with the New-AzADServicePrincipal cmdlet. When creating a service principal, you choose the type of sign-in authentication it uses.

> **NOTE**
>
> If your account doesn't have permission to create a service principal, `New-AzADServicePrincipal` will return an error message containing "Insufficient privileges to complete the operation." Contact your Azure Active Directory admin to create a service principal.

There are two types of authentication available for service principals: Password-based authentication, and certificate-based authentication.

Password-based authentication

Without any other authentication parameters, password-based authentication is used and a random password created for you. If you want password-based authentication, this method is recommended.

```
$sp = New-AzADServicePrincipal -DisplayName ServicePrincipalName
```

The returned object contains the `Secret` member, which is a `SecureString` containing the generated password. Make sure that you store this value somewhere secure to authenticate with the service principal. Its value **won't** be displayed in the console output. If you lose the password, reset the service principal credentials.

For user-supplied passwords, the `-PasswordCredential` argument takes `Microsoft.Azure.Commands.ActiveDirectory.PSADPasswordCredential` objects. These objects must have a valid `StartDate` and `EndDate`, and take a plaintext `Password`. When creating a password, make sure you follow the Azure Active Directory password rules and restrictions. Don't use a weak password or reuse a password.

```
Import-Module Az.Resources # Imports the PSADPasswordCredential object
$credentials = New-Object Microsoft.Azure.Commands.ActiveDirectory.PSADPasswordCredential -Property @{
StartDate=Get-Date; EndDate=Get-Date -Year 2024; Password=<Choose a strong password>}
$sp = New-AzAdServicePrincipal -DisplayName ServicePrincipalName -PasswordCredential $credentials
```

The object returned from `New-AzADServicePrincipal` contains the `Id` and `DisplayName` members, either of which can be used for sign in with the service principal.

> **IMPORTANT**
>
> Signing in with a service principal requires the tenant ID which the service principal was created under. To get the active tenant when the service principal was created, run the following command **immediately after** service principal creation:
>
> ```
> (Get-AzContext).Tenant.Id
> ```

Certificate-based authentication

Service principals using certificate-based authentication are created with the `-CertValue` parameter. This parameter takes a base64-encoded ASCII string of the public certificate. This is represented by a PEM file, or a text-encoded CRT or CER. Binary encodings of the public certificate aren't supported. These instructions assume that you already have a certificate available.

```
$cert = <public certificate as base64-encoded string>
$sp = New-AzADServicePrincipal -DisplayName ServicePrincipalName -CertValue $cert
```

You can also use the `-KeyCredential` parameter, which takes `PSADKeyCredential` objects. These objects must have a valid `StartDate`, `EndDate`, and have the `CertValue` member set to a base64-encoded ASCII string of the public certificate.

```
$cert = <public certificate as base64-encoded string>
$credentials = New-Object Microsoft.Azure.Commands.ActiveDirectory.PSADKeyCredential -Property @{
StartDate=Get-Date; EndDate=Get-Date -Year 2024; KeyId=New-Guid; CertValue=$cert}
$sp = New-AzADServicePrincipal -DisplayName ServicePrincipalName -KeyCredential $credentials
```

The object returned from `New-AzADServicePrincipal` contains the `Id` and `DisplayName` members, either of which can be used for sign in with the service principal. Clients which sign in with the service principal also need access to the certificate's private key.

> **IMPORTANT**
>
> Signing in with a service principal requires the tenant ID which the service principal was created under. To get the active tenant when the service principal was created, run the following command **immediately after** service principal creation:
>
> ```
> (Get-AzContext).Tenant.Id
> ```

Get an existing service principal

A list of service principals for the currently active tenant can be retrieved with Get-AzADServicePrincipal. By default this command returns **all** service principals in a tenant, so for large organizations, it may take a long time to return results. Instead, using one of the optional server-side filtering arguments is recommended:

- `-DisplayNameBeginsWith` requests service principals that have a *prefix* that match the provided value. The display name of a service principal is the value set with `-DisplayName` during creation.
- `-DisplayName` requests an *exact match* of a service principal name.

Manage service principal roles

Azure PowerShell has the following cmdlets to manage role assignments:

- Get-AzRoleAssignment
- New-AzRoleAssignment
- Remove-AzRoleAssignment

The default role for a service principal is **Contributor**. This role has full permissions to read and write to an Azure account. The **Reader** role is more restrictive, with read-only access. For more information on Role-Based Access Control (RBAC) and roles, see RBAC: Built-in roles.

This example adds the **Reader** role and removes the **Contributor** one:

```
New-AzRoleAssignment -ApplicationId <service principal application ID> -RoleDefinitionName "Reader"
Remove-AzRoleAssignment -ApplicationId <service principal application ID> -RoleDefinitionName "Contributor"
```

IMPORTANT

Role assignment cmdlets don't take the service principal object ID. They take the associated application ID, which is generated at creation time. To get the application ID for a service principal, use `Get-AzADServicePrincipal`.

NOTE

If your account doesn't have permission to assign a role, you see an error message that your account "does not have authorization to perform action 'Microsoft.Authorization/roleAssignments/write'." Contact your Azure Active Directory admin to manage roles.

Adding a role *doesn't* restrict previously assigned permissions. When restricting a service principal's permissions, the **Contributor** role should be removed.

The changes can be verified by listing the assigned roles:

```
Get-AzRoleAssignment -ServicePrincipalName ServicePrincipalName
```

Sign in using a service principal

Test the new service principal's credentials and permissions by signing in. To sign in with a service principal, you need the `applicationId` value associated with it, and the tenant it was created under.

To sign in with a service principal using a password:

```
# Use the application ID as the username, and the secret as password
$credentials = Get-Credential
Connect-AzAccount -ServicePrincipal -Credential $credentials -Tenant <tenant ID>
```

Certificate-based authentication requires that Azure PowerShell can retrieve information from a local certificate store based on a certificate thumbprint.

```
Connect-AzAccount -ServicePrincipal -TenantId $tenantId -CertificateThumbprint <thumbprint>
```

For instructions on importing a certificate into a credential store accessible by PowerShell, see Sign in with Azure PowerShell

Reset credentials

If you forget the credentials for a service principal, use New-AzADSpCredential to add a new credential. This cmdlet takes the same credential arguments and types as `New-AzADServicePrincipal`. Without any credential arguments, a new `PasswordCredential` with a random password is created.

> **IMPORTANT**
>
> Before assigning any new credentials, you may want to remove existing credentials to prevent sign in with them. To do so, use the Remove-AzADSpCredential cmdlet:
>
> ```
> Remove-AzADSpCredential -DisplayName ServicePrincipalName
> ```

```
$newCredential = New-AzADSpCredential -ServicePrincipalName ServicePrincipalName
```

Persist user credentials across PowerShell sessions

5/17/2019 • 5 minutes to read • Edit Online

Azure PowerShell offers a feature called **Azure Context Autosave**, which gives the following features:

- Retention of sign-in information for reuse in new PowerShell sessions.
- Easier use of background tasks for executing long-running cmdlets.
- Switch between accounts, subscriptions, and environments without a separate sign-in.
- Execution of tasks using different credentials and subscriptions, simultaneously, from the same PowerShell session.

Azure contexts defined

An *Azure context* is a set of information that defines the target of Azure PowerShell cmdlets. The context consists of five parts:

- An *Account* - the UserName or Service Principal used to authenticate communications with Azure
- A *Subscription* - The Azure Subscription with the Resources being acted upon.
- A *Tenant* - The Azure Active Directory tenant that contains your subscription. Tenants are more important to ServicePrincipal authentication.
- An *Environment* - The particular Azure Cloud being targeted, usually the Azure global Cloud. However, the environment setting allows you to target National, Government, and on-premises (Azure Stack) clouds as well.
- *Credentials* - The information used by Azure to verify your identity and confirm your authorization to access resources in Azure

With the latest version of Azure PowerShell, Azure Contexts can automatically be saved whenever opening a new PowerShell session.

Automatically save the context for the next sign-in

Azure PowerShell retains your context information automatically between sessions. To set PowerShell to forget your context and credentials, use `Disable-AzContextAutoSave`. With context saving disabled, you'll need to sign in to Azure every time you open a PowerShell session.

To allow Azure PowerShell to remember your context after the PowerShell session is closed, use `Enable-AzContextAutosave`. Context and credential information are automatically saved in a special hidden folder in your user directory (`$env:USERPROFILE\.Azure` on Windows, and `$HOME/.Azure` on other platforms). Each new PowerShell session targets the context used in your last session.

The cmdlets that allow you to manage Azure contexts also allow you fine grained control. If you want changes to apply only to the current PowerShell session (`Process` scope) or every PowerShell session (`CurrentUser` scope). These options are discussed in more detail in Using Context Scopes.

Running Azure PowerShell cmdlets as background jobs

The **Azure Context Autosave** feature also allows you to share you context with PowerShell background jobs. PowerShell allows you to start and monitor long-executing tasks as background jobs without having to wait for the tasks to complete. You can share credentials with background jobs in two different ways:

- Passing the context as an argument

Most AzureRM cmdlets allow you to pass the context as a parameter to the cmdlet. You can pass a context to a background job as shown in the following example:

```
PS C:\> $job = Start-Job { param ($ctx) New-AzVm -AzureRmContext $ctx [... Additional parameters ...]}
-ArgumentList (Get-AzContext)
```

- Using the default context with Autosave enabled

  If you have enabled **Context Autosave**, background jobs automatically use the default saved context.

```
PS C:\> $job = Start-Job { New-AzVm [... Additional parameters ...]}
```

When you need to know the outcome of the background task, use `Get-Job` to check the job status and `Wait-Job` to wait for the Job to complete. Use `Receive-Job` to capture or display the output of the background job. For more information, see about_Jobs.

Creating, selecting, renaming, and removing contexts

To create a context, you must be signed in to Azure. The `Connect-AzAccount` cmdlet (or its alias, `Login-AzAccount`) sets the default context used by Azure PowerShell cmdlets, and allows you to access any tenants or subscriptions allowed by your credentials.

To add a new context after sign-in, use `Set-AzContext` (or its alias, `Select-AzSubscription`).

```
PS C:\> Set-AzContext -Subscription "Contoso Subscription 1" -Name "Contoso1"
```

The previous example adds a new context targeting 'Contoso Subscription 1' using your current credentials. The new context is named 'Contoso1'. If you don't provide a name for the context, a default name, using the account ID and subscription ID is used.

To rename an existing context, use the `Rename-AzContext` cmdlet. For example:

```
PS C:\> Rename-AzContext '[user1@contoso.org; 123456-7890-1234-564321]` 'Contoso2'
```

This example renames the context with automatic name `[user1@contoso.org; 123456-7890-1234-564321]` to the simple name 'Contoso2'. Cmdlets that manage contexts also use tab completion, allowing you to quickly select the context.

Finally, to remove a context, use the `Remove-AzContext` cmdlet. For example:

```
PS C:\> Remove-AzContext Contoso2
```

Forgets the context that was named 'Contoso2'. You can recreate this context using `Set-AzContext`

Removing credentials

You can remove all credentials and associated contexts for a user or service principal using `Disconnect-AzAccount` (also known as `Logout-AzAccount`). When executed without parameters, the `Disconnect-AzAccount` cmdlet removes all credentials and contexts associated with the User or Service Principal in the current context. You may pass in a Username, Service Principal Name, or context to target a particular principal.

```
Disconnect-AzAccount user1@contoso.org
```

Using context scopes

Occasionally, you may want to select, change, or remove a context in a PowerShell session without impacting other sessions. To change the default behavior of context cmdlets, use the `Scope` parameter. The `Process` scope overrides the default behavior by making it apply only for the current session. Conversely `CurrentUser` scope changes the context in all sessions, instead of just the current session.

As an example, to change the default context in the current PowerShell session without impacting other windows, or the context used the next time a session is opened, use:

```
PS C:\> Select-AzContext Contoso1 -Scope Process
```

How the context autosave setting is remembered

The context AutoSave setting is saved to the user Azure PowerShell directory (`$env:USERPROFILE\.Azure` on Windows, and `$HOME/.Azure` on other platforms). Some kinds of computer accounts may not have access to this directory. For such scenarios, you can use the environment variable

```
$env:AzureRmContextAutoSave="true" | "false"
```

When set to 'true', the context is automatically saved. If set to 'false', the context isn't saved.

Context management cmdlets

- Enable-AzContextAutosave - Allow saving the context between powershell sessions. Any changes alter the global context.
- Disable-AzContextAutosave - Turn off autosaving the context. Each new PowerShell session is required to sign in again.
- Select-AzContext - Select a context as the default. All cmdlets use the credentials in this context for authentication.
- Disconnect-AzAccount - Remove all credentials and contexts associated with an account.
- Remove-AzContext - Remove a named context.
- Rename-AzContext - Rename an existing context.
- Add-AzAccount - Allow scoping of the sign-in to the process or the current user. Allow naming the default context after authentication.
- Import-AzContext - Allow scoping of the sign-in to the process or the current user.
- Set-AzContext - Allow selection of existing named contexts, and scope changes to the process or current user.

Migrate from AzureRM to Azure PowerShell Az

5/16/2019 • 3 minutes to read • Edit Online

The Az module has feature parity with AzureRM, but uses shorter and more consistent cmdlet names. Scripts written for the AzureRM cmdlets won't automatically work with the new module. To make the transition easier, Az offers tools to allow you to run your existing scripts using AzureRM. No migration to a new command set is ever convenient, but this article will help you get started on transitioning to the new module.

To see the full list of breaking changes between AzureRM and Az, see the Migration guide for Az 1.0.0

Check for installed versions of AzureRM

The Az module can be installed side-by-side with the AzureRM module, but this is not recommended. Before taking any migration steps, check which versions of AzureRM are installed on your system. Doing so allows you to make sure scripts are already running on the latest release, and let you know if you can enable command aliases without uninstalling AzureRM.

To check which version(s) of AzureRM you have installed, run the command:

```
Get-InstalledModule -Name AzureRM -AllVersions
```

Ensure your existing scripts work with the latest AzureRM release

This is the most important step! Run your existing scripts, and make sure that they work with the *latest* release of AzureRM (**6.13.1**). If your scripts don't work, make sure to read the AzureRM 5.x to 6.x migration guide.

Uninstall AzureRM

The Az module is not guaranteed to be compatible with any existing AzureRM installs in PowerShell 5.1 for Windows. Before you install the Az module, uninstall AzureRM.

> **IMPORTANT**
>
> If you're not ready to remove the AzureRM module from your system, you can install the Az module for PowerShell Core 6.x or later instead. PowerShell Core and PowerShell 5.1 for Windows use different module libraries, so there will be no conflicts. You can still enable aliases in PowerShell Core.

Install the Azure PowerShell Az module

The first step is to install the Az module on your platform. When you install Az, it's recommended that you uninstall AzureRM. In the following steps, you'll learn how to keep running your existing scripts and enable compatibility for old cmdlet names.

To install the Azure PowerShell Az module, follow the instructions in Install the Az module.

> **NOTE**
>
> At this point, you might want to run the Uninstall-AzureRM cmdlet provided in the Az module, just to make sure that all versions of AzureRM have been uninstalled and won't cause conflicts.

Enable AzureRM compatibility aliases

With AzureRM uninstalled and your scripts working with the latest AzureRM version, the next step is to enable the compatibility mode for the Az module. Compatibility is enabled with the command:

```
Enable-AzureRmAlias -Scope CurrentUser
```

Aliases enable the ability to use old cmdlet names with the Az module installed. These aliases are written to the user profile for the selected scope. If no user profile exists, one is created.

> **WARNING**
> You can use a different `-Scope` for this command, but it's not recommended. Aliases are written to the user profile for the selected scope, so keep enabling them to as limited a scope as possible. Enabling aliases system-wide could also cause issues for other users which have AzureRM installed in their local scope.

Once the alias mode is enabled, run your scripts again to confirm that they still function as expected.

Change module imports and cmdlet names

In general, the module names have been changed so that `AzureRM` and `Azure` become `Az`, and the same for cmdlets. For example, the `AzureRM.Compute` module has been renamed to `Az.Compute`. `New-AzureRMVM` has become `New-AzVM`, and `Get-AzureStorageBlob` is now `Get-AzStorageBlob`.

There are exceptions to this naming change that you should be aware of. Some modules were renamed or merged into existing modules without this affecting the suffix of their cmdlets, other than changing `AzureRM` or `Azure` to `Az`. Otherwise, the full cmdlet suffix was changed to reflect the new module name.

AZURERM MODULE	AZ MODULE	CMDLET SUFFIX CHANGED?
AzureRM.Profile	Az.Accounts	Yes
AzureRM.Insights	Az.Monitor	Yes
AzureRM.DataFactories	Az.DataFactory	Yes
AzureRM.DataFactoryV2	Az.DataFactory	Yes
AzureRM.RecoveryServices.Backup	Az.RecoveryServices	No
AzureRM.RecoveryServices.SiteRecovery	Az.RecoveryServices	No
AzureRM.Tags	Az.Resources	No
AzureRM.MachineLearningCompute	Az.MachineLearning	No
AzureRM.UsageAggregates	Az.Billing	No
AzureRM.Consumption	Az.Billing	No

Summary

By following these steps, you can update all of your existing scripts to use the new module. If you have any questions or problems with these steps that made your migration difficult, please comment on this article so that we can improve the instructions.

Query output of Azure PowerShell

5/6/2019 • 2 minutes to read • Edit Online

The results of each Azure PowerShell cmdlet are an Azure PowerShell object. Even cmdlets that aren't explicitly `Get-` operations might return a value that can be inspected, to give information about a resource that was created or modified. While most cmdlets return a single object, some return an array that should be iterated through.

In almost all cases, you query output from Azure PowerShell with the Select-Object cmdlet, often abbreviated to `select`. Output can be filtered with Where-Object, or its alias `where`.

Select simple properties

In the default table format, Azure PowerShell cmdlets don't display all of their available properties. You can get the full properties by using the Format-List cmdlet, or by piping output to `Select-Object *`:

```
Get-AzVM -Name TestVM -ResourceGroupName TestGroup | Select-Object *
```

```
ResourceGroupName        : TESTGROUP
Id                       : /subscriptions/711d8ed1-b888-4c52-8ab9-
66f07b87eb6b/resourceGroups/TESTGROUP/providers/Micro
                           soft.Compute/virtualMachines/TestVM
VmId                     : 711d8ed1-b888-4c52-8ab9-66f07b87eb6b
Name                     : TestVM
Type                     : Microsoft.Compute/virtualMachines
Location                 : westus2
LicenseType              :
Tags                     : {}
AvailabilitySetReference :
DiagnosticsProfile       :
Extensions               : {}
HardwareProfile          : Microsoft.Azure.Management.Compute.Models.HardwareProfile
InstanceView             :
NetworkProfile           : Microsoft.Azure.Management.Compute.Models.NetworkProfile
OSProfile                : Microsoft.Azure.Management.Compute.Models.OSProfile
Plan                     :
ProvisioningState        : Succeeded
StorageProfile           : Microsoft.Azure.Management.Compute.Models.StorageProfile
DisplayHint              : Compact
Identity                 :
Zones                    : {}
FullyQualifiedDomainName :
AdditionalCapabilities   :
RequestId                : 711d8ed1-b888-4c52-8ab9-66f07b87eb6b
StatusCode               : OK
```

Once you know the names of the properties that you're interested in, you can use those property names with `Select-Object` to get them directly:

```
Get-AzVM -Name TestVM -ResourceGroupName TestGroup | Select-Object Name,VmId,ProvisioningState
```

```
Name   VmId                                 ProvisioningState
----   ----                                 -----------------
TestVM 711d8ed1-b888-4c52-8ab9-66f07b87eb6b Succeeded
```

Output from using `Select-Object` is always formatted to display the requested information. To learn about using formatting as part of querying cmdlet results, see Format Azure PowerShell cmdlet output.

Select nested properties

Some properties in Azure PowerShell cmdlet output use nested objects, like the `StorageProfile` property of `Get-AzVM` output. To get a value from a nested property, provide a display name and the full path to the value you want to inspect as part of a dictionary argument to `Select-Object`:

```
Get-AzVM -ResourceGroupName TestGroup | `
    Select-Object Name,@{Name="OSType"; Expression={$_.StorageProfile.OSDisk.OSType}}
```

```
Name     OSType
----     ------
TestVM    Linux
TestVM2   Linux
WinVM   Windows
```

Each dictionary argument selects one property from the object. The property to extract must be part of an expression.

Filter results

The `Where-Object` cmdlet allows you to filter the result based on any property value, including nested properties. The next example shows how to use `Where-Object` to find the Linux VMs in a resource group.

```
Get-AzVM -ResourceGroupName TestGroup | `
    Where-Object {$_.StorageProfile.OSDisk.OSType -eq "Linux"}
```

```
ResourceGroupName    Name Location          VmSize OsType        NIC ProvisioningState Zone
-----------------    ---- --------          ------ ------        --- ----------------- ----
TestGroup           TestVM  westus2 Standard_D2s_v3  Linux  testvm299         Succeeded
TestGroup          TestVM2  westus2 Standard_D2s_v3  Linux testvm2669         Succeeded
```

You can pipe the results of `Select-Object` and `Where-Object` to each other. For performance purposes, it's always recommended to put the `Where-Object` operation before `Select-Object`:

```
Get-AzVM -ResourceGroupName TestGroup | `
    Where-Object {$_.StorageProfile.OsDisk.OsType -eq "Linux"} | `
    Select-Object Name,VmID,ProvisioningState
```

```
Name    VmId                                 ProvisioningState
----    ----                                 -----------------
TestVM  711d8ed1-b888-4c52-8ab9-66f07b87eb6  Succeeded
TestVM2 cbcee769-dd78-45e3-a14d-2ad11c647d0  Succeeded
```

Format Azure PowerShell cmdlet output

5/6/2019 • 3 minutes to read • Edit Online

By default each Azure PowerShell cmdlet formats output to be easy to read. PowerShell allows you to convert or format cmdlet output by piping to one of the following cmdlets:

FORMATTING	CONVERSION
Format-Custom	ConvertTo-Csv
Format-List	ConvertTo-Html
Format-Table	ConvertTo-Json
Format-Wide	ConvertTo-Xml

Formatting is used for display in a PowerShell terminal, and conversion is used for generating data to be consumed by other scripts or programs.

Table output format

By default, Azure PowerShell cmdlets output in the table format. This format doesn't display all information of the requested resource:

```
Get-AzVM
```

```
ResourceGroupName           Name Location          VmSize  OsType              NIC ProvisioningState Zone
-----------------           ---- --------          ------  ------              --- ----------------- ----
QueryExample      ExampleLinuxVM  westus2        Basic_A0   Linux examplelinuxvm916         Succeeded
QueryExample         RHELExample  westus2  Standard_D2_v3   Linux    rhelexample469         Succeeded
QueryExample        WinExampleVM  westus2 Standard_DS1_v2 Windows   winexamplevm268         Succeeded
```

The amount of data displayed by `Format-Table` can be affected by the width of your PowerShell session window. To restrict the output to specific properties and order them, property names can be provided as arguments to `Format-Table`:

```
Get-AzVM -ResourceGroupName QueryExample | Format-Table Name,ResourceGroupName,Location
```

```
Name           ResourceGroupName Location
----           ----------------- --------
ExampleLinuxVM QueryExample      westus2
RHELExample    QueryExample      westus2
WinExampleVM   QueryExample      westus2
```

List output format

List output format produces two columns, property names followed by the value. For complex objects, the type of the object is displayed instead.

```
Get-AzVM | Format-List
```

The following output has some fields removed.

```
ResourceGroupName        : QueryExample
Id                       :
/subscriptions/.../resourceGroups/QueryExample/providers/Microsoft.Compute/virtualMachines/ExampleLinuxVM
VmId                     : ...
Name                     : ExampleLinuxVM
Type                     : Microsoft.Compute/virtualMachines
Location                 : westus2
...
HardwareProfile          : Microsoft.Azure.Management.Compute.Models.HardwareProfile
InstanceView             :
NetworkProfile           : Microsoft.Azure.Management.Compute.Models.NetworkProfile
OSProfile                : Microsoft.Azure.Management.Compute.Models.OSProfile
...
StatusCode               : OK

ResourceGroupName        : QueryExample
Id                       :
/subscriptions/.../resourceGroups/QueryExample/providers/Microsoft.Compute/virtualMachines/RHELExample
VmId                     : ...
Name                     : RHELExample
Type                     : Microsoft.Compute/virtualMachines
Location                 : westus2
...
```

Like `Format-Table`, property names can be provided to order and restrict the output:

```
Get-AzVM | Format-List ResourceGroupName,Name,Location
```

```
ResourceGroupName : QueryExample
Name              : ExampleLinuxVM
Location          : westus2

ResourceGroupName : QueryExample
Name              : RHELExample
Location          : westus2

ResourceGroupName : QueryExample
Name              : WinExampleVM
Location          : westus2
```

Wide output format

Wide output format produces only one property name per query. Which property is displayed can be controlled by giving a property as an argument.

```
Get-AzVM | Format-Wide
```

```
ExampleLinuxVM                              RHELExample
WinExampleVM
```

```
Get-AzVM | Format-Wide ResourceGroupName
```

```
QueryExample                                QueryExample
QueryExample
```

Custom output format

The `Custom-Format` output type is meant for formatting custom objects. Without any arguments, it behaves like `Format-List` but displays the property names of custom classes.

```
Get-AzVM | Format-Custom
```

The following output has some fields removed.

```
ResourceGroupName : QueryExample
Id                :
/subscriptions/.../resourceGroups/QueryExample/providers/Microsoft.Compute/virtualMachines/ExampleLinuxVM
VmId              : ...
Name              : ExampleLinuxVM
Type              : Microsoft.Compute/virtualMachines
Location          : westus2
Tags              : {}
HardwareProfile   : {VmSize}
NetworkProfile    : {NetworkInterfaces}
OSProfile         : {ComputerName, AdminUsername, LinuxConfiguration, Secrets,
AllowExtensionOperations}
ProvisioningState : Succeeded
StorageProfile    : {ImageReference, OsDisk, DataDisks}
...
```

Giving property names as arguments to `Custom-Format` displays the property/value pairs for custom objects set as values:

```
Get-AzVM | Format-Custom Name,ResourceGroupName,Location,OSProfile
```

The following output has some fields removed.

```
class PSVirtualMachineList
{
  Name = ExampleLinuxVM
  ResourceGroupName = QueryExample
  Location = westus2
  OSProfile =
    class OSProfile
    {
      ComputerName = ExampleLinuxVM
      AdminUsername = ...
      AdminPassword =
      CustomData =
      WindowsConfiguration =
      LinuxConfiguration =
        class LinuxConfiguration
        {
          DisablePasswordAuthentication = False
          Ssh =
          ProvisionVMAgent = True
        }
      Secrets =
        [
        ]

      AllowExtensionOperations = True
    }
}

...

class PSVirtualMachineList
{
  Name = WinExampleVM
  ResourceGroupName = QueryExample
  Location = westus2
  OSProfile =
    class OSProfile
    {
      ComputerName = WinExampleVM
      AdminUsername = ...
      AdminPassword =
      CustomData =
      WindowsConfiguration =
        class WindowsConfiguration
        {
          ProvisionVMAgent = True
          EnableAutomaticUpdates = True
          TimeZone =
          AdditionalUnattendContent =
          WinRM =
        }
      LinuxConfiguration =
      Secrets =
        [
        ]

      AllowExtensionOperations = True
    }
}
```

Conversion to other data formats

The `ConvertTo-*` family of cmdlets allows for converting the results of Azure PowerShell cmdlets to machine-readable formats. To get only some properties from the Azure PowerShell results, use the `Select-Object` command in a pipe before performing the conversion. The following examples demonstrate the different kinds of

output that each conversion produces.

Conversion to CSV

```
Get-AzVM | ConvertTo-CSV
```

```
#TYPE Microsoft.Azure.Commands.Compute.Models.PSVirtualMachineList
"ResourceGroupName","Id","VmId","Name","Type","Location","LicenseType","Tags","AvailabilitySetReference","Diag
nosticsProfile","Extensions","HardwareProfile","InstanceView","NetworkProfile","OSProfile","Plan","Provisionin
gState","StorageProfile","DisplayHint","Identity","Zones","FullyQualifiedDomainName","AdditionalCapabilities",
"RequestId","StatusCode"
"QUERYEXAMPLE","/subscriptions/.../resourceGroups/QUERYEXAMPLE/providers/Microsoft.Compute/virtualMachines/Exa
mpleLinuxVM","...","ExampleLinuxVM","Microsoft.Compute/virtualMachines","westus2",,"System.Collections.Generic
.Dictionary`2[System.String,System.String]",,,"System.Collections.Generic.List`1[Microsoft.Azure.Management.Co
mpute.Models.VirtualMachineExtension]","Microsoft.Azure.Management.Compute.Models.HardwareProfile",,"Microsoft
.Azure.Management.Compute.Models.NetworkProfile","Microsoft.Azure.Management.Compute.Models.OSProfile",,"Succe
eded","Microsoft.Azure.Management.Compute.Models.StorageProfile","Compact",,"System.Collections.Generic.List`1
[System.String]",,,"...","OK"
"QUERYEXAMPLE","/subscriptions/.../resourceGroups/QUERYEXAMPLE/providers/Microsoft.Compute/virtualMachines/RHE
LExample","...","RHELExample","Microsoft.Compute/virtualMachines","westus2",,"System.Collections.Generic.Dicti
onary`2[System.String,System.String]",,,"System.Collections.Generic.List`1[Microsoft.Azure.Management.Compute.
Models.VirtualMachineExtension]","Microsoft.Azure.Management.Compute.Models.HardwareProfile",,"Microsoft.Azure
.Management.Compute.Models.NetworkProfile","Microsoft.Azure.Management.Compute.Models.OSProfile",,"Succeeded",
"Microsoft.Azure.Management.Compute.Models.StorageProfile","Compact",,"System.Collections.Generic.List`1[Syste
m.String]",,,"...","OK"
"QUERYEXAMPLE","/subscriptions/.../resourceGroups/QUERYEXAMPLE/providers/Microsoft.Compute/virtualMachines/Win
ExampleVM","...","WinExampleVM","Microsoft.Compute/virtualMachines","westus2",,"System.Collections.Generic.Dic
tionary`2[System.String,System.String]",,,"System.Collections.Generic.List`1[Microsoft.Azure.Management.Comput
e.Models.VirtualMachineExtension]","Microsoft.Azure.Management.Compute.Models.HardwareProfile",,"Microsoft.Azu
re.Management.Compute.Models.NetworkProfile","Microsoft.Azure.Management.Compute.Models.OSProfile",,"Succeeded
","Microsoft.Azure.Management.Compute.Models.StorageProfile","Compact",,"System.Collections.Generic.List`1[Sys
tem.String]",,,"...","OK"
```

Conversion to JSON

JSON output doesn't expand all properties by default. To change the depth of properties expanded, use the `-Depth` argument. By default, the expansion depth is `2`.

```
Get-AzVM|ConvertTo-JSON
```

The following output has some fields removed.

```
[
    {
        "ResourceGroupName":  "QUERYEXAMPLE",
        "Id":
"/subscriptions/.../resourceGroups/QUERYEXAMPLE/providers/Microsoft.Compute/virtualMachines/ExampleLinuxVM",
        "VmId":  "...",
        "Name":  "ExampleLinuxVM",
        "Type":  "Microsoft.Compute/virtualMachines",
        "Location":  "westus2",
        ...
        "OSProfile":  {
                          "ComputerName":  "ExampleLinuxVM",
                          "AdminUsername":  "...",
                          "AdminPassword":  null,
                          "CustomData":  null,
                          "WindowsConfiguration":  null,
                          "LinuxConfiguration":
"Microsoft.Azure.Management.Compute.Models.LinuxConfiguration",
                          "Secrets":  "",
                          "AllowExtensionOperations":  true
                      },
        "Plan":  null,
        "ProvisioningState":  "Succeeded",
        "StorageProfile":  {
                               "ImageReference":  "Microsoft.Azure.Management.Compute.Models.ImageReference",
                               "OsDisk":  "Microsoft.Azure.Management.Compute.Models.OSDisk",
                               "DataDisks":  ""
                           },
        "DisplayHint":  0,
        "Identity":  null,
        "Zones":  [

                  ],
        "FullyQualifiedDomainName":  null,
        "AdditionalCapabilities":  null,
        "RequestId":  "...",
        "StatusCode":  200
    },
    ...
]
```

Conversion to XML

The `ConvertTo-XML` cmdlet converts the Azure PowerShell response object into a pure XML object, which can be handled like any other XML object within PowerShell.

```
Get-AzVM | ConvertTo-XML
```

```
xml                          Objects
---                          -------
version="1.0" encoding="utf-8" Objects
```

Conversion to HTML

Converting an object to HTML produces output that will be rendered as an HTML table. Rendering of the HTML will depend on your browser behavior for rendering tables which contain no width information. No custom class objects are expanded.

```
Get-AzVM | ConvertTo-HTML
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>HTML TABLE</title>
</head><body>
<table>
<colgroup><col/><col/><col/><col/><col/><col/><col/><col/><col/><col/><col/><col/><col/><col/><col/><col/>
<col/><col/><col/><col/><col/><col/><col/><col/><col/></colgroup>
<tr><th>ResourceGroupName</th><th>Id</th><th>VmId</th><th>Name</th><th>Type</th><th>Location</th>
<th>LicenseType</th><th>Tags</th><th>AvailabilitySetReference</th><th>DiagnosticsProfile</th>
<th>Extensions</th><th>HardwareProfile</th><th>InstanceView</th><th>NetworkProfile</th><th>OSProfile</th>
<th>Plan</th><th>ProvisioningState</th><th>StorageProfile</th><th>DisplayHint</th><th>Identity</th>
<th>Zones</th><th>FullyQualifiedDomainName</th><th>AdditionalCapabilities</th><th>RequestId</th>
<th>StatusCode</th></tr>
<tr><td>QUERYEXAMPLE</td>
<td>/subscriptions/.../resourceGroups/QUERYEXAMPLE/providers/Microsoft.Compute/virtualMachines/ExampleLinuxVM<
/td><td>...</td><td>ExampleLinuxVM</td><td>Microsoft.Compute/virtualMachines</td><td>westus2</td><td></td>
<td>System.Collections.Generic.Dictionary`2[System.String,System.String]</td><td></td><td></td>
<td>System.Collections.Generic.List`1[Microsoft.Azure.Management.Compute.Models.VirtualMachineExtension]</td>
<td>Microsoft.Azure.Management.Compute.Models.HardwareProfile</td><td></td>
<td>Microsoft.Azure.Management.Compute.Models.NetworkProfile</td>
<td>Microsoft.Azure.Management.Compute.Models.OSProfile</td><td></td><td>Succeeded</td>
<td>Microsoft.Azure.Management.Compute.Models.StorageProfile</td><td>Compact</td><td></td>
<td>System.Collections.Generic.List`1[System.String]</td><td></td><td></td><td></td><td>...</td><td>OK</td></tr>
<tr><td>QUERYEXAMPLE</td>
<td>/subscriptions/.../resourceGroups/QUERYEXAMPLE/providers/Microsoft.Compute/virtualMachines/RHELExample</td
><td>...</td><td>RHELExample</td><td>Microsoft.Compute/virtualMachines</td><td>westus2</td><td></td>
<td>System.Collections.Generic.Dictionary`2[System.String,System.String]</td><td></td><td></td>
<td>System.Collections.Generic.List`1[Microsoft.Azure.Management.Compute.Models.VirtualMachineExtension]</td>
<td>Microsoft.Azure.Management.Compute.Models.HardwareProfile</td><td></td>
<td>Microsoft.Azure.Management.Compute.Models.NetworkProfile</td>
<td>Microsoft.Azure.Management.Compute.Models.OSProfile</td><td></td><td>Succeeded</td>
<td>Microsoft.Azure.Management.Compute.Models.StorageProfile</td><td>Compact</td><td></td>
<td>System.Collections.Generic.List`1[System.String]</td><td></td><td></td><td></td><td>...</td><td>OK</td></tr>
<tr><td>QUERYEXAMPLE</td>
<td>/subscriptions/.../resourceGroups/QUERYEXAMPLE/providers/Microsoft.Compute/virtualMachines/WinExampleVM</t
d><td>...</td><td>WinExampleVM</td><td>Microsoft.Compute/virtualMachines</td><td>westus2</td><td></td>
<td>System.Collections.Generic.Dictionary`2[System.String,System.String]</td><td></td><td></td>
<td>System.Collections.Generic.List`1[Microsoft.Azure.Management.Compute.Models.VirtualMachineExtension]</td>
<td>Microsoft.Azure.Management.Compute.Models.HardwareProfile</td><td></td>
<td>Microsoft.Azure.Management.Compute.Models.NetworkProfile</td>
<td>Microsoft.Azure.Management.Compute.Models.OSProfile</td><td></td><td>Succeeded</td>
<td>Microsoft.Azure.Management.Compute.Models.StorageProfile</td><td>Compact</td><td></td>
<td>System.Collections.Generic.List`1[System.String]</td><td></td><td></td><td></td><td>...</td><td>OK</td></tr>
</table>
</body></html>
```

Use multiple Azure subscriptions

5/6/2019 • 2 minutes to read • Edit Online

Most Azure users will only ever have a single subscription. However, if you are part of more than one organization or your organization has divided up access to certain resources across groupings, you may have multiple subscriptions within Azure. The CLI supports selecting a subscription both globally and per command.

Tenants, users, and subscriptions

You might have some confusion over the difference between tenants, users, and subscriptions within Azure. A *tenant* is the Azure Active Directory entity that encompasses a whole organization. This tenant has at least one *subscription* and *user*. A user is an individual and is associated with only one tenant, the organization that they belong to. Users are those accounts that sign in to Azure to create, manage, and use resources. A user may have access to multiple *subscriptions*, which are the agreements with Microsoft to use cloud services, including Azure. Every resource is associated with a subscription.

To learn more about the differences between tenants, users, and subscriptions, see the Azure cloud terminology dictionary. To learn how to add a new subscription to your Azure Active Directory tenant, see How to add an Azure subscription to Azure Active Directory. To learn how to sign in to a specific tenant, see Sign in with Azure PowerShell.

Change the active subscription

In Azure PowerShell, accessing the resources for a subscription requires changing the subscription associated with your current Azure session. This is done by modifying the active session context, the information about which tenant, subscription, and user cmdlets should be run against. In order to change subscriptions, an Azure PowerShell Context object first needs to be retrieved with Get-AzSubscription and then the current context changed with Set-AzContext.

The next example shows how to get a subscription in the currently active tenant, and set it as the active session:

```
$context = Get-AzSubscription -SubscriptionId ...
Set-AzContext $context
```

To learn more about Azure PowerShell contexts, including how to save them and quickly switch between them for working with multiple subscriptions easily, see Persist credentials with Azure PowerShell contexts.

Use experimental Azure PowerShell modules

5/6/2019 • 3 minutes to read • Edit Online

With the emphasis on developer tools in Azure, the Azure PowerShell team experiments with many improvements to the Azure PowerShell experience. This article describes how to opt-in to experiments with Azure PowerShell and give feedback to the development team.

Experimentation methodology

To facilitate experimentation, we're creating new Azure PowerShell modules that implement existing Azure SDK functionality in new, easier to use ways. In most cases, the cmdlets exactly mirror existing cmdlets. However, the experimental cmdlets provide a shorthand notation and smarter default values that make it easier to create and manage Azure resources.

These modules can be installed side by side with existing Azure PowerShell modules. The cmdlet names have been shortened to provide shorter names and avoid name conflicts with existing, non-experimental cmdlets.

The experimental modules use the following naming convention: `Az.*.Experiments`. This naming convention is similar to the naming of Preview modules: `Az.*.Preview`. Preview modules differ from experimental modules. Preview modules implement new functionality of Azure services that is only available as a Preview offering. Preview modules replace existing Azure PowerShell modules and use the same cmdlet and parameter names.

How to install an experimental module

Experimental modules are published to the PowerShell Gallery just like the existing Azure PowerShell modules. To see a list of experimental modules, run the following command:

```
Find-Module Az.*.Experiments
```

To install an experimental module, use the `Install-Module` cmdlet.

Documentation and support

Documentation is included in the install package and is accessed using the `Get-Help` cmdlet. No official documentation is published for experimental modules.

We encourage you to test these modules. Your feedback allows us to improve usability and respond to your needs. However, being experimental, support for these modules is limited. Questions or problem reports can be submitted by creating an issue in the GitHub repository.

Experiments and areas of improvement

These improvements were selected based on key differentiators in competing products. For example, Azure CLI has made a point of basing commands on *scenarios* rather than *API surface area*. Azure CLI uses a number of smart defaults that make "getting started" scenarios easier for end users.

Core improvements

The core improvements are regarded as "common sense", and little experimentation is needed to move forward in implementing these updates.

- Scenario-based Cmdlets - **All*- cmdlets should be designed around scenarios, not the Azure REST service.

- Shorter Names - Includes the names of cmdlets and the names of parameters. Use aliases for compatibility with "old" cmdlets. Provide *backwards compatible* parameter sets.
- Smart Defaults - Create smart defaults to fill in "required" information. For example:
  - Resource Group
  - Location
  - Dependent resources

Experimental improvements

The experimental improvements present a significant change that the team wants to validate via experimentation.

- Simple types - Create scenarios should move away from complex types and config objects. Simplify the configuration down to one or two parameters.
- "Smart Create" - All create scenarios implementing "Smart Create" would have *no* required parameters: all necessary information would be chosen by Azure PowerShell in an opinionated fashion.
- Gray Parameters - In many cases, some parameters could be "gray" or semi-optional. If the parameter isn't specified, the user is asked if they want the parameter generated for them. It also makes sense for gray parameters to infer a value based on context with the user's consent. For example, it may make sense to have the gray parameter suggest the most recently used value.
- `-Auto` Switch - The `-Auto` switch would provide an "opt-in" way for users to *default all the things* while maintaining the integrity of required parameters in the mainline path.

Feature-specific switches

For example, the "Create web app" scenario might have a `-Git` or `-AddRemote` switch that would automatically add an "azure" remote to an existing git repository.

- Settable Defaults - Users should be able to set defaults for common parameters like `-ResourceGroupName` and `-Location`.
- Size Defaults - Resource "sizes" can be confusing to users since many Resource Providers use different names (for example, "Standard_DS1_v2" or "S1"). However, most users care more about cost. So it makes sense to define "universal" sizes based on a pricing schedule. Users can choose a specific size or let Azure PowerShell choose the *best option* based on the resource the budget.
- Output Format - Azure PowerShell currently returns `PSObject`s and there's little console output. Azure PowerShell may need to display some information to the user about the "smart defaults" used.

Running cmdlets in parallel using PowerShell jobs

5/6/2019 • 3 minutes to read • Edit Online

PowerShell supports asynchronous action with PowerShell Jobs. Azure PowerShell is heavily dependent on making, and waiting for, network calls to Azure. You may often find yourself needing to make non-blocking calls. To address this need, Azure PowerShell provides first-class PSJob support.

Context Persistence and PSJobs

Since PSJobs are run as separate processes, your Azure connection must be shared with them. After signing in to your Azure account with `Connect-AzAccount`, pass the context to a job.

```
$creds = Get-Credential
$job = Start-Job { param($context,$vmadmin) New-AzVM -Name MyVm -AzContext $context -Credential $vmadmin} -
ArgumentList (Get-AzContext),$creds
```

However, if you have chosen to have your context automatically saved with `Enable-AzContextAutosave`, the context is automatically shared with any jobs you create.

```
Enable-AzContextAutosave
$creds = Get-Credential
$job = Start-Job { param($vmadmin) New-AzVM -Name MyVm -Credential $vmadmin} -ArgumentList $creds
```

Automatic Jobs with `-AsJob`

As a convenience, Azure PowerShell also provides an `-AsJob` switch on some long-running cmdlets. The `-AsJob` switch makes creating PSJobs even easier.

```
$creds = Get-Credential
$job = New-AzVM -Name MyVm -Credential $creds -AsJob
```

You can inspect the job and progress at any time with `Get-Job` and `Get-AzVM`.

```
Get-Job $job
Get-AzVM MyVm
```

```
Id Name                                    PSJobTypeName         State   HasMoreData Location  Command
-- ----                                    -------------         -----   ----------- --------  -------
1  Long Running Operation for 'New-AzVM' AzureLongRunningJob`1 Running True        localhost New-AzVM

ResourceGroupName    Name Location          VmSize  OsType     NIC ProvisioningState Zone
-----------------    ---- --------          ------  ------     --- ----------------- ----
MyVm                 MyVm   eastus Standard_DS1_v2 Windows    MyVm          Creating
```

When the job completes, get the result of the job with `Receive-Job`.

NOTE

`Receive-Job` returns the result from the cmdlet as if the `-AsJob` flag were not present. For example, the `Receive-Job` result of `Do-Action -AsJob` is of the same type as the result of `Do-Action`.

```
$vm = Receive-Job $job
$vm
```

```
ResourceGroupName        : MyVm
Id                       : /subscriptions/XXXXXXXX-XXXX-XXXX-XXXX-
XXXXXXXXXXXX/resourceGroups/MyVm/providers/Microsoft.Compute/virtualMachines/MyVm
VmId                     : dff1f79e-a8f7-4664-ab72-0ec28b9fbb5b
Name                     : MyVm
Type                     : Microsoft.Compute/virtualMachines
Location                 : eastus
Tags                     : {}
HardwareProfile          : {VmSize}
NetworkProfile           : {NetworkInterfaces}
OSProfile                : {ComputerName, AdminUsername, WindowsConfiguration, Secrets}
ProvisioningState        : Succeeded
StorageProfile           : {ImageReference, OsDisk, DataDisks}
FullyQualifiedDomainName : myvmmyvm.eastus.cloudapp.azure.com
```

Example Scenarios

Create several VMs at once:

```
$creds = Get-Credential
# Create 10 jobs.
for($k = 0; $k -lt 10; $k++) {
    New-AzVm -Name MyVm$k  -Credential $creds -AsJob
}

# Get all jobs and wait on them.
Get-Job | Wait-Job
"All jobs completed"
Get-AzVM
```

In this example, the `Wait-Job` cmdlet causes the script to pause while jobs run. The script continues executing once all of the jobs have completed. Several jobs run in parallel then the script waits for completion before continuing.

```
Id     Name            PSJobTypeName   State         HasMoreData     Location             Command
--     ----            -------------   -----         -----------     --------             -------
2      Long Running... AzureLongRun... Running       True            localhost            New-AzVM
3      Long Running... AzureLongRun... Running       True            localhost            New-AzVM
4      Long Running... AzureLongRun... Running       True            localhost            New-AzVM
5      Long Running... AzureLongRun... Running       True            localhost            New-AzVM
6      Long Running... AzureLongRun... Running       True            localhost            New-AzVM
7      Long Running... AzureLongRun... Running       True            localhost            New-AzVM
8      Long Running... AzureLongRun... Running       True            localhost            New-AzVM
9      Long Running... AzureLongRun... Running       True            localhost            New-AzVM
10     Long Running... AzureLongRun... Running       True            localhost            New-AzVM
11     Long Running... AzureLongRun... Running       True            localhost            New-AzVM
2      Long Running... AzureLongRun... Completed     True            localhost            New-AzVM
3      Long Running... AzureLongRun... Completed     True            localhost            New-AzVM
4      Long Running... AzureLongRun... Completed     True            localhost            New-AzVM
5      Long Running... AzureLongRun... Completed     True            localhost            New-AzVM
6      Long Running... AzureLongRun... Completed     True            localhost            New-AzVM
7      Long Running... AzureLongRun... Completed     True            localhost            New-AzVM
8      Long Running... AzureLongRun... Completed     True            localhost            New-AzVM
9      Long Running... AzureLongRun... Completed     True            localhost            New-AzVM
10     Long Running... AzureLongRun... Completed     True            localhost            New-AzVM
11     Long Running... AzureLongRun... Completed     True            localhost            New-AzVM
All Jobs completed.

ResourceGroupName        Name   Location          VmSize  OsType           NIC ProvisioningState Zone
-----------------        ----   --------          ------  ------           --- ----------------- ----
MYVM                     MyVm     eastus Standard_DS1_v2 Windows          MyVm         Succeeded
MYVM0                   MyVm0     eastus Standard_DS1_v2 Windows        MyVm0         Succeeded
MYVM1                   MyVm1     eastus Standard_DS1_v2 Windows        MyVm1         Succeeded
MYVM2                   MyVm2     eastus Standard_DS1_v2 Windows        MyVm2         Succeeded
MYVM3                   MyVm3     eastus Standard_DS1_v2 Windows        MyVm3         Succeeded
MYVM4                   MyVm4     eastus Standard_DS1_v2 Windows        MyVm4         Succeeded
MYVM5                   MyVm5     eastus Standard_DS1_v2 Windows        MyVm5         Succeeded
MYVM6                   MyVm6     eastus Standard_DS1_v2 Windows        MyVm6         Succeeded
MYVM7                   MyVm7     eastus Standard_DS1_v2 Windows        MyVm7         Succeeded
MYVM8                   MyVm8     eastus Standard_DS1_v2 Windows        MyVm8         Succeeded
MYVM9                   MyVm9     eastus Standard_DS1_v2 Windows        MyVm9         Succeeded
```

2.0.0 - May 2019

5/6/2019 • 21 minutes to read • Edit Online

Az.Accounts

- Update Authentication Library to fix ADFS issues with username/password auth

Az.CognitiveServices

- Only display Bing disclaimer for Bing Search Services.
- Improve error when create account failed.

Az.Compute

- Proximity placement group feature.
  - The following new cmdlets are added: New-AzProximityPlacementGroup Get-AzProximityPlacementGroup Remove-AzProximityPlacementGroup
  - The new parameter, ProximityPlacementGroupId, is added to the following cmdlets: New-AzAvailabilitySet New-AzVMConfig New-AzVmssConfig
- StorageAccountType parameter is added to New-AzGalleryImageVersion.
- TargetRegion of New-AzGalleryImageVersion can contain StorageAccountType.
- SkipShutdown switch parameter is added to Stop-AzVM and Stop-AzVmss
- Breaking changes
  - Set-AzVMBootDiagnostics is changed to Set-AzVMBootDiagnostic.
  - Export-AzLogAnalyticThrottledRequests is changed to Export-AzLogAnalyticThrottledRequests.

Az.DeploymentManager

- First Generally Available release of Azure Deployment Manager cmdlets

Az.Dns

- Automatic DNS NameServer Delegation
  - Create DNS zone cmdlet accepts parent zone name as additional optional parameter.
  - Adds NS records in the parent zone for newly created child zone.

Az.FrontDoor

- First Generally Available Release of Azure FrontDoor cmdlets
- Rename WAF cmdlets to include 'Waf'
  - `Get-AzFrontDoorFireWallPolicy --> Get-AzFrontDoorWafPolicy`
  - `New-AzFrontDoorCustomRuleObject --> New-AzFrontDoorWafCustomRuleObject`
  - `New-AzFrontDoorFireWallPolicy --> New-AzFrontDoorWafPolicy`
  - `New-AzFrontDoorManagedRuleObject --> New-AzFrontDoorWafManagedRuleObject`
  - `New-AzFrontDoorManagedRuleOverrideObject --> New-AzFrontDoorWafManagedRuleOverrideObject`
  - `New-AzFrontDoorMatchConditionObject --> New-AzFrontDoorWafMatchConditionObject`
  - `New-AzFrontDoorRuleGroupOverrideObject --> New-AzFrontDoorWafRuleGroupOverrideObject`
  - `Remove-AzFrontDoorFireWallPolicy --> Remove-AzFrontDoorWafPolicy`
  - `Update-AzFrontDoorFireWallPolicy --> Update-AzFrontDoorWafPolicy`

Az.HDInsight

- Removed two cmdlets:
  - Grant-AzHDInsightHttpServicesAccess
  - Revoke-AzHDInsightHttpServicesAccess

- Added a new cmdlet Set-AzHDInsightGatewayCredential to replace Grant-AzHDInsightHttpServicesAccess
- Update cmdlet Get-AzHDInsightJobOutput to distinguish reader role and hdinsight operator role:
  - Users with reader role need to specify 'DefaultStorageAccountKey' parameter explicitly, otherwise error occurs.
  - Users with hdinsight operator role will not be affected.

Az.Monitor

- New cmdlets for SQR API (Scheduled Query Rule)
  - New-AzScheduledQueryRuleAlertingAction
  - New-AzScheduledQueryRuleAznsActionGroup
  - New-AzScheduledQueryRuleLogMetricTrigger
  - New-AzScheduledQueryRuleSchedule
  - New-AzScheduledQueryRuleSource
  - New-AzScheduledQueryRuleTriggerCondition
  - New-AzScheduledQueryRule
  - Get-AzScheduledQueryRule
  - Set-AzScheduledQueryRule
  - Update-AzScheduledQueryRule
  - Remove-AzScheduledQueryRule
  - More information about SQR API
  - Updated Az.Monitor.md to include cmdlets for GenV2(non classic) metric-based alert rule

Az.Network

- Add support for Nat Gateway Resource
  - New cmdlets
    - New-AzNatGateway
    - Get-AzNatGateway
    - Set-AzNatGateway
    - Remove-AzNatGateway
  - Updated cmdlets - New-AzureVirtualNetworkSubnetConfigCommand - Add-AzureVirtualNetworkSubnetConfigCommand
- Updated below commands for feature: Custom routes set/remove on Brooklyn Gateway.
  - Updated New-AzVirtualNetworkGateway: Added optional parameter -CustomRoute to set the address prefixes as custom routes to set on Gateway.
  - Updated Set-AzVirtualNetworkGateway: Added optional parameter -CustomRoute to set the address prefixes as custom routes to set on Gateway.

Az.PolicyInsights

- Support for querying policy evaluation details.
  - Add '-Expand' parameter to Get-AzPolicyState. Support '-Expand PolicyEvaluationDetails'.

Az.RecoveryServices

- Support for Cross subscription Azure to Azure site recovery.
- Marking upcoming breaking changes for Azure Site Recovery.
- Fix for Azure Site Recovery recovery plan end action plan.
- Fix for Azure Site Recovery Update network mapping for Azure to Azure.
- Fix for Azure Site Recovery update protection direction for Azure to Azure for managed disk.
- Other minor fixes.

Az.Relay

- Fix typos in customer-facing messages

Az.ServiceBus

- Added new cmdlets for NetworkRuleSet of Namespace

Az.Storage

- Upgrade to Storage Client Library 10.0.1 (the namespace of all objects from this SDK change from 'Microsoft.WindowsAzure.Storage.' *to* '*Microsoft.Azure.Storage.*')
- Upgrade to Microsoft.Azure.Management.Storage 11.0.0, to support new API version 2019-04-01.
- The default Storage account Kind in Create Storage account change from 'Storage' to 'StorageV2'
  - New-AzStorageAccount
- Change the Storage account cmdlet output Sku.Name to be aligned with input SkuName by add '-', like 'StandardLRS' change to 'Standard_LRS'
  - New-AzStorageAccount
  - Get-AzStorageAccount
  - Set-AzStorageAccount

Az.Websites

- 'Kind' property will now be set for PSSite objects returned by Get-AzWebApp
- Get-AzWebApp*Metrics and Get-AzAppServicePlanMetrics marked deprecated

1.8.0 - April 2019

Highlights since the last major release

- General availability of `Az` module
- For more information about the `Az` module, please visit the following: https://aka.ms/azps-announce
- Added Location, ResourceGroup, and ResourceName completers: https://azure.microsoft.com/en-us/blog/completers-in-azure-powershell/
- Added wildcard support to Get cmdlets for Az.Compute and Az.Network
- Added interactive and username/password authentication for Windows PowerShell 5.1 only
- Added support for Python 2 runbooks in Az.Automation
- Az.LogicApp: New cmdlets for Integration Account Assemblies and Batch Configuration

Az.Accounts

- Update Uninstall-AzureRm to correctly delete modules in Mac

Az.Batch

- Updated cmdlets with plural nouns to singular, and deprecated plural names.

Az.Cdn

- Updated cmdlets with plural nouns to singular, and deprecated plural names.

Az.CognitiveServices

- Updated cmdlets with plural nouns to singular, and deprecated plural names.

Az.Compute

- Fix issue with AEM installation if resource ids of disks had lowercase resourcegroups in resource id
- Updated cmdlets with plural nouns to singular, and deprecated plural names.
- Fix documentation for wildcards

Az.DataFactory

- Add SsisProperties if NodeCount not null for managed integration runtime.

Az.DataLakeStore

- Updated cmdlets with plural nouns to singular, and deprecated plural names.

Az.EventGrid

- Updated the help text for endpoint to indicate that resources should be created before using the create/update event subscription cmdlets.

Az.EventHub

- Added new cmdlets for NetworkRuleSet of Namespace

Az.HDInsight

- Updated cmdlets with plural nouns to singular, and deprecated plural names.

Az.IotHub

- Updated cmdlets with plural nouns to singular, and deprecated plural names.

Az.KeyVault

- Updated cmdlets with plural nouns to singular, and deprecated plural names.
- Fix documentation for wildcards

Az.MachineLearning

- Updated cmdlets with plural nouns to singular, and deprecated plural names.

Az.Media

- Updated cmdlets with plural nouns to singular, and deprecated plural names.

Az.Monitor

- New cmdlets for GenV2(non classic) metric-based alert rule
  - New-AzMetricAlertRuleV2DimensionSelection
  - New-AzMetricAlertRuleV2Criteria
  - Remove-AzMetricAlertRuleV2
  - Get-AzMetricAlertRuleV2
  - Add-AzMetricAlertRuleV2
- Updated Monitor SDK to version 0.22.0-preview

Az.Network

- Updated cmdlets with plural nouns to singular, and deprecated plural names.
- Fix documentation for wildcards

Az.NotificationHubs

- Updated cmdlets with plural nouns to singular, and deprecated plural names.

Az.OperationalInsights

- Updated cmdlets with plural nouns to singular, and deprecated plural names.

Az.PowerBIEmbedded

- Updated cmdlets with plural nouns to singular, and deprecated plural names.

Az.RecoveryServices

- Updated cmdlets with plural nouns to singular, and deprecated plural names.
- Updated table format for SQL in azure VM
- Added alternate method to fetch location in AzureFileShare
- Updated ScheduleRunDays in SchedulePolicy object according to timezone

Az.RedisCache

- Updated cmdlets with plural nouns to singular, and deprecated plural names.

Az.Resources

- Fix documentation for wildcards

Az.Sql

- Replace dependency on Monitor SDK with common code

- Updated cmdlets with plural nouns to singular, and deprecated plural names.
- Enhanced process of multiple columns classification.
- Include sku properties (sku name, family, capacity) in response from Get-AzSqlServerServiceObjective and format as table by default.
- Ability to Get-AzSqlServerServiceObjective by location without needing a preexisting server in the region.
- Support for time zone parameter in Managed Instance create.
- Fix documentation for wildcards

Az.Websites

- fixes the Set-AzWebApp and Set-AzWebAppSlot to not remove the tags on execution
- Updated cmdlets with plural nouns to singular, and deprecated plural names.
- Updated the WebSites SDK.
- Removed the AdminSiteName property from PSAppServicePlan.

1.7.0 - April 2019

Highlights since the last major release

- General availability of `Az` module
- For more information about the `Az` module, please visit the following: https://aka.ms/azps-announce
- Added Location, ResourceGroup, and ResourceName completers: https://azure.microsoft.com/en-us/blog/completers-in-azure-powershell/
- Added wildcard support to Get cmdlets for Az.Compute and Az.Network
- Added interactive and username/password authentication for Windows PowerShell 5.1 only
- Added support for Python 2 runbooks in Az.Automation
- Az.LogicApp: New cmdlets for Integration Account Assemblies and Batch Configuration

Az.Accounts

- Updated Add-AzEnvironment and Set-AzEnvironment to accept parameter AzureAnalysisServicesEndpointResourceId

Az.AnalysisServices

- Using ServiceClient in dataplane cmdlets and removing the original authentication logic
- Making Add-AzureASAccount a wrapper of Connect-AzAccount to avoid a breaking change

Az.Automation

- Fixed New-AzAutomationSoftwareUpdateConfiguration cmdlet bug for Inclusions. Now parameter IncludedKbNumber and IncludedPackageNameMask should work.
- Bug fix for azure automation update management dynamic group

Az.Compute

- Add HyperVGeneration parameter to New-AzDiskConfig and New-AzSnapshotConfig
- Allow VM creation with galley image from other tenants.

Az.ContainerInstance

- Fixed issue in the -Command parameter of New-AzContainerGroup which added a trailing empty argument

Az.DataFactory

- Updated ADF .Net SDK version to 3.0.2
- Updated Set-AzDataFactoryV2 cmdlet with extra parameters for RepoConfiguration related settings.

Az.Resources

- Improve handling of providers for 'Get-AzResource' when providing '-ResourceId' or '-ResourceGroupName', '-Name' and '-ResourceType' parameters

- Improve error handling for for 'Test-AzDeployment' and 'Test-AzResourceGroupDeployment'
  - Handle errors thrown outside of deployment validation and include them in output of command instead
  - More information here: https://github.com/Azure/azure-powershell/issues/6856
- Add '-IgnoreDynamicParameters' switch parameter to set of deployment cmdlets to skip prompt in script and job scenarios
  - More information here: https://github.com/Azure/azure-powershell/issues/6856

Az.Sql

- Support Database Data Classification.

Az.Storage

- Report detail error when create Storage context with parameter -UseConnectedAccount, but without login Azure account
  - New-AzStorageContext
- Support Manage Blob Service Properties of a specified Storage account with Management plane API
  - Update-AzStorageBlobServiceProperty
  - Get-AzStorageBlobServiceProperty
  - Enable-AzStorageBlobDeleteRetentionPolicy
  - Disable-AzStorageBlobDeleteRetentionPolicy
- -AsJob support for Blob and file upload and download cmdlets
  - Get-AzStorageBlobContent
  - Set-AzStorageBlobContent
  - Get-AzStorageFileContent
  - Set-AzStorageFileContent

1.6.0 - March 2019

Highlights since the last major release

- General availability of `Az` module
- For more information about the `Az` module, please visit the following: https://aka.ms/azps-announce
- Added Location, ResourceGroup, and ResourceName completers: https://azure.microsoft.com/en-us/blog/completers-in-azure-powershell/
- Added wildcard support to Get cmdlets for Az.Compute and Az.Network
- Added interactive and username/password authentication for Windows PowerShell 5.1 only
- Added support for Python 2 runbooks in Az.Automation
- Az.LogicApp: New cmdlets for Integration Account Assemblies and Batch Configuration

Az.Automation

- Azure automation update management change to support the following new features :
  - Dynamic grouping
  - Pre-Post script
  - Reboot Setting

Az.Compute

- Fix issue with path resolution in Get-AzVmBootDiagnosticsData
- Update Compute client library to 25.0.0.

Az.KeyVault

- Added wildcard support to KeyVault cmdlets

Az.Network

- Add Threat Intelligence support for Azure Firewall

- Add Application Gateway Firewall Policy top level resource and Custom Rules

Az.RecoveryServices

- Added SnapshotRetentionInDays in Azure VM policy to support Instant RP
- Added pipe support for unregister container

Az.Resources

- Update wildcard support for Get-AzResource and Get-AzResourceGroup
- Update credentials used when making generic calls to ARM

Az.Sql

- changed Threat Detection's cmdlets param (ExcludeDetectionType) from DetectionType to string[] to make it future proof when new DetectionTypes are added and to support autocomplete as well.

Az.Storage

- Support Get/Set/Remove Management Policy on a Storage account
  - Set-AzStorageAccountManagementPolicy
  - Get-AzStorageAccountManagementPolicy
  - Remove-AzStorageAccountManagementPolicy
  - Add-AzStorageAccountManagementPolicyAction
  - New-AzStorageAccountManagementPolicyFilter
  - New-AzStorageAccountManagementPolicyRule

Az.Websites

- Fix ARM template bug that breaks cloning all slots using 'New-AzWebApp -IncludeSourceWebAppSlots'

1.5.0 - March 2019

Az.Accounts

- Add 'Register-AzModule' command to support AutoRest generated cmdlets
- Update examples for Connect-AzAccount

Az.Automation

- Fixed issue when retreiving certain monthly schedules in several Azure Automation cmdlets
- Fix Get-AzAutomationDscNode returning just top 20 nodes. Now it returns all nodes

Az.Cdn

- Added new Powershell cmdlets for Enable/Disable Custom Domain Https and deprecated the old ones

Az.Compute

- Add wildcard support to Get cmdlets

Az.DataFactory

- Updated ADF .Net SDK version to 3.0.1

Az.LogicApp

- Fix for ListWorkflows only retrieving the first page of results

Az.Network

- Add wildcard support to Network cmdlets

Az.RecoveryServices

- Added Sql server in Azure VM support
- SDK Update
- Removed VMappContainer check in Get-ProtectableItem
- Added Name and ServerName as parameters for Get-ProtectableItem

Az.Resources

- Add `-TemplateObject` parameter to deployment cmdlets
  - More information here: https://github.com/Azure/azure-powershell/issues/2933
- Fix issue when piping the result of `Get-AzResource` to `Set-AzResource`
  - More information here: https://github.com/Azure/azure-powershell/issues/8240
- Fix issue with JSON data type change when running `Set-AzResource`
  - More information here: https://github.com/Azure/azure-powershell/issues/7930

Az.Sql

- Updating AuditingEndpointsCommunicator.
  - Fixing the behavior of an edge case while creating new diagnostic settings.

Az.Storage

- Support Kind BlockBlobStorage when create Storage account - New-AzStorageAccount

1.4.0 - February 2019

Az.AnalysisServices

- Deprecated AddAzureASAccount cmdlet

Az.Automation

- Update help for Import-AzAutomationDscNodeConfiguration
- Added configuration name validation to Import-AzAutomationDscConfiguration cmdlet
- Improved error handling for Import-AzAutomationDscConfiguration cmdlet

Az.CognitiveServices

- Added CustomSubdomainName as a new optional parameter for New-AzCognitiveServicesAccount which is used to specify subdomain for the resource.

Az.Compute

- Fix issue with ID parameter sets
- Update Get-AzVMExtension to list all installed extension if Name parameter is not provided
- Add Tag and ResourceId parameters to Update-AzImage cmdlet
- Get-AzVmssVM without instance ID and with InstanceView can list VMSS VMs with instance view.

Az.DataLakeStore

- Add cmdlets for ADL deleted item enumerate and restore

Az.EventHub

- Added new boolean property SkipEmptyArchives to Skip Empty Archives in CaptureDescription class of Eventhub

Az.KeyVault

- Fix tagging on Set-AzKeyVaultSecret

Az.LogicApp

- Add in Basic sku for Integration Accounts
- Add in XSLT 2.0, XSLT 3.0 and Liquid Map Types
- New cmdlets for Integration Account Assemblies
  - Get-AzIntegrationAccountAssembly
  - New-AzIntegrationAccountAssembly
  - Remove-AzIntegrationAccountAssembly
  - Set-AzIntegrationAccountAssembly
- New cmdlets for Integration Account Batch Configuration

    - Get-AzIntegrationAccountBatchConfiguration
    - New-AzIntegrationAccountBatchConfiguration
    - Remove-AzIntegrationAccountBatchConfiguration
    - Set-AzIntegrationAccountBatchConfiguration
- Update Logic App SDK to version 4.1.0

Az.Monitor

- Update help for Get-AzMetric

Az.Network

- Update help example for Add-AzApplicationGatewayCustomError

Az.OperationalInsights

- Additional support for New and Get ApplicationInsights data source.
    - Added new 'ApplicationInsights' kind to support Get specific and Get all ApplicationInsights data sources for given workspace.
    - Added New-AzOperationalInsightsApplicationInsightsDataSource cmdlet for creating data source by given Application-Insights resource parameters: subscription Id, resourceGroupName and name.

Az.Resources

- Fix for issue https://github.com/Azure/azure-powershell/issues/8166
- Fix for issue https://github.com/Azure/azure-powershell/issues/8235
- Fix for issue https://github.com/Azure/azure-powershell/issues/6219
- Fix bug preventing repeat creation of KeyCredentials

Az.Sql

- Add support for SQL DB Hyperscale tier
- Fixed bug where restore could fail due to setting unnecessary properties in restore request

Az.Websites

- Correct example in Get-AzWebAppSlotMetrics

1.3.0 - February 2019

Az.Accounts

- Update to latest version of ClientRuntime

Az.AnalysisServices

General availability for Az.AnalysisServices module.

Az.Compute

- AEM extension: Add support for UltraSSD and P60,P70 and P80 disks
- Update help description for Set-AzVMBootDiagnostics
- Update help description and example for Update-AzImage

Az.RecoveryServices

General availability for Az.RecoveryServices module.

Az.Resources

- Fix tagging for resource groups
    - More information here: https://github.com/Azure/azure-powershell/issues/8166
- Fix issue where `Get-AzureRmRoleAssignment` doesn't respect -ErrorAction
    - More information here: https://github.com/Azure/azure-powershell/issues/8235

Az.Sql

- Add Get/Set AzSqlDatabaseBackupShortTermRetentionPolicy

- Fix issue where not being logged into Azure account would result in nullref exception when executing SQL cmdlets
- Fixed null ref exception in Get-AzSqlCapability

1.2.1 - January 2019

Az.Accounts

- Release with correct version of Authentication

Az.AnalysisServices

- Release with updated Authentication dependency

Az.RecoveryServices

- Release with updated Authentication dependency

1.2.0 - January 2019

Az.Accounts

- Add interactive and username/password authentication for Windows PowerShell 5.1 only
- Update incorrect online help URLs
- Add warning message in PS Core for Uninstall-AzureRm

Az.Aks

- Update incorrect online help URLs

Az.Automation

- Added support for Python 2 runbooks
- Update incorrect online help URLs

Az.Cdn

- Update incorrect online help URLs

Az.Compute

- Add Invoke-AzVMReimage cmdlet
- Add TempDisk parameter to Set-AzVmss
- Fix the warning message of New-AzVM

Az.ContainerRegistry

- Update incorrect online help URLs

Az.DataFactory

- Updated ADF .Net SDK version to 3.0.0

Az.DataLakeStore

- Fix issue with ADLS endpoint when using MSI
  - More information here: https://github.com/Azure/azure-powershell/issues/7462
- Update incorrect online help URLs

Az.IotHub

- Add Encoding format to Add-IotHubRoutingEndpoint cmdlet.

Az.KeyVault

- Update incorrect online help URLs

Az.Network

- Update incorrect online help URLs

Az.Resources

- Fix incorrect examples in 'New-AzADAppCredential' and 'New-AzADSpCredential' reference documentation
- Fix issue where path for '-TemplateFile' parameter was not being resolved before executing resource group deployment cmdlets
- Az.Resources: Correct documentation for New-AzureRmPolicyDefinition -Mode default value
- Az.Resources: Fix for issue https://github.com/Azure/azure-powershell/issues/7522
- Az.Resources: Fix for issue https://github.com/Azure/azure-powershell/issues/5747
- Fix formatting issue with 'PSResourceGroupDeployment' object
  - More information here: https://github.com/Azure/azure-powershell/issues/2123

Az.ServiceFabric

- Rollback when a certificate is added to VMSS model but an exception is thrown this is to fix bug: https://github.com/Azure/service-fabric-issues/issues/932
- Fix some error messages.
- Fix create cluster with default ARM template for New-AzServiceFabriCluster which was not working with migration to Az.
- Fix add cluster/application certificate to only add to VM Scale Sets that correspond to the cluster by checking cluster id in the extension.

Az.SignalR

- Update incorrect online help URLs

Az.Sql

- Update incorrect online help URLs
- Updated parameter description for LicenseType parameter with possible values
- Fix for updating managed instance identity not working when it is the only updated property
- Support for custom collation on managed instance

Az.Storage

- Update incorrect online help URLs
- Give detail error message when get/set classic Logging/Metric on Premium Storage Account, since Premium Storage Account not supoort classic Logging/Metric.
  - Get/Set-AzStorageServiceLoggingProperty
  - Get/Set-AzStorageServiceMetricsProperty

Az.TrafficManager

- Update incorrect online help URLs

Az.Websites

- Update incorrect online help URLs
- Fixes 'New-AzWebAppSSLBinding' to upload the certificate to the correct resourcegroup+location if the app is hosted on an ASE.
- Fixes 'New-AzWebAppSSLBinding' to not overwrite the tags on binding an SSL certificate to an app

1.1.0 - January 2019

Az.Accounts

- Add 'Local' Scope to Enable-AzureRmAlias

Az.Compute

- Name is now optional in ID parameter set for Restart/Start/Stop/Remove/Set-AzVM and Save-AzVMImage
- Updated the description of ID in help files
- Fix backward compatibility issue with Az.Accounts module

Az.DataLakeStore

- Update the sdk version of dataplane to 1.1.14 for SDK fixes.
  - Fix handling of negative acesstime and modificationtime for getfilestatus and liststatus, Fix async cancellation token

Az.EventGrid

- Updated to use the 2019-01-01 API version.
- Update the following cmdlets to support new scenario in 2019-01-01 API version
  - New-AzureRmEventGridSubscription: Add new optional parameters for specifying:
    - Event Time-To-Live,
    - Maximum number of delivery attempts for the events,
    - Dead letter endpoint.
  - Update-AzureRmEventGridSubscription: Add new optional parameters for specifying:
    - Event Time-To-Live,
    - Maximum number of delivery attempts for the events,
    - Dead letter endpoint.
- Add new enum values (namely, storageQueue and hybridConnection) for EndpointType option in New-AzureRmEventGridSubscription and Update-AzureRmEventGridSubscription cmdlets.
- Show warning message if creating or updating the event subscription is expected to entail manual action from user.

Az.IotHub

- Updated to the latest version of the IotHub SDK

Az.LogicApp

- Get-AzLogicApp lists all without specified Name

Az.Resources

- Fix parameter set issue when providing '-ODataQuery' and '-ResourceId' parameters for 'Get-AzResource'
  - More information here: https://github.com/Azure/azure-powershell/issues/7875
- Fix handling of the -Custom parameter in New/Set-AzPolicyDefinition
- Fix typo in New-AzDeployment documentation
- Made '-MailNickname' parameter mandatory for 'New-AzADUser'
  - More information here: https://github.com/Azure/azure-powershell/issues/8220

Az.SignalR

- Fix backward compatibility issue with Az.Accounts module

Az.Sql

- Converted the Storage management client dependency to the common SDK implementation.

Az.Storage

- Set the StorageAccountName of Storage context as the real Storage Account Name, when it's created with Sas Token, OAuth or Anonymous
  - New-AzStorageContext
- Create Sas Token of Blob Snapshot Object with '-FullUri' parameter, fix the returned Uri to be the sanpshot Uri
  - New-AzStorageBlobSASToken

Az.Websites

- Fixed a date parsing bug in 'Get-AzDeletedWebApp'
- Fix backward compatibility issue with Az.Accounts module

1.0.0 - December 2018

General

- General Availability of Az Module
- Online help for each module
- For more details and a roadmap, see the Az Announcement page
- See the Migration Guide for information on migrating from AzureRM

Az.Accounts

- Changed from Az.Profile
- Fixed table formats for profile and context types

Az.ApiManagement

- Fixes for #7002
- Minor breaking changes, see the Migration Guide for details

Az.Batch

- Added the ability to see what version of the Azure Batch Node Agent is running on each of the VMs in a pool, via the new `NodeAgentInformation` property on `PSComputeNode`.
- The `Caching` default for `PSDataDisk` is now `ReadWrite` instead of `None`.
- Minor breaking changes, see the Migration Guide for details

Az.Billing

- Combines Billing, Consumption, and UsageAggregates cmdlets, see the Migration Guide for details

Az.CognitivServices

- Add completers for SkuName and Typem available on New-AzureRmCognitiveServicesAccount operation
- Removed GetSkusWithAccountParamSetName parameter set from Get-AzCognitiveServicesAccountSkus

Az.ContainerInstance

- Added ManagedIdentity support

Az.DataLakeAnalytics

- Minor breaking changes, see the Migration Guide for details

Az.DataLakeStore

- Minor breaking changes, see the Migration Guide for details

Az.Monitor

- Renamed Az.Insights to Az.Monitor and other minor breaking changes, see the Migration Guide for details

Az.KeyVault

- Removed the deprecated 'PurgeDisabled' property from output types

Az.MachineLearning

- Included cmdlets from Az.MachineLearningCompute module

Az.Media

- Remove deprecated -Tags alias from New-AzMediaService

Az.Network

Added support for the configuring RewriteRuleSets in the Application Gateway - New cmdlets added: - Add-AzureRmApplicationGatewayRewriteRuleSet - Get-AzureRmApplicationGatewayRewriteRuleSet - New-AzureRmApplicationGatewayRewriteRuleSet - Remove-AzureRmApplicationGatewayRewriteRuleSet - Set-AzureRmApplicationGatewayRewriteRuleSet - New-AzureRmApplicationGatewayRewriteRule - New-AzureRmApplicationGatewayRewriteRuleActionSet - New-

AzureRmApplicationGatewayRewriteRuleHeaderConfiguration - Cmdlets updated with optional parameter - RewriteRuleSet - New-AzureRmApplicationGateway - New-AzureRmApplicationGatewayRequestRoutingRule - Add-AzureRmApplicationGatewayRequestRoutingRule - New-AzureRmApplicationGatewayPathRuleConfig - Add-AzureRmApplicationGatewayUrlPathMapConfig - New-AzureRmApplicationGatewayUrlPathMapConfig Added KeyVault Support to Application Gateway using Identity. - Cmdlets updated with optonal parameter - KeyVaultSecretId, -KeyVaultSecret - Add-AzApplicationGatewaySslCertificate - New-AzApplicationGatewaySslCertificate - Set-AzApplicationGatewaySslCertificate - New-AzApplicationGateway cmdlet updated with optional parameter -UserAssignedIdentity

- Minor breaking changes, see the Migration Guide for details

Az.OperationalInsights

- Minor breaking changes, see the Migration Guide for details

Az.Profile

- Changed module name to Az.Accounts

Az.RecoveryServices

- Minor breaking changes, see the Migration Guide for details

Az.Resources

- Minor breaking changes, see the Migration Guide for details

Az.ServiceFabric

- Support specfying certificate by common name and thumbprint
- Mnor breaking changes, see the Migration Guide for details

Az.SIgnalR

- General Availability for PowerShell cmdlets for SIgnalR

Az.Sql

- Added new Data_Exfiltration and Unsafe_Action detection types to Threat Detection's cmdlets
- Updated documentation examples for Sql Auditing cmdlets
- Minor breaking changes, see the Migration Guide for details

Az.Storage

- Minor breaking changes, see the Migration Guide for details

Az.Websites

- Minor breaking changes, see the Migration Guide for details

0.7.0 - December 2018

General

- Minor changes for upcoming AzureRM to Az transition

Az.Compute

- Add support for UltraSSD and Gallery Images in the simple param sets for `New-AzVm(ss)` cmdlets.

Az.DataLakeStore

- Fix the trailing slash of the domain of adls account

Az.FrontDoor

- Fixed some broken links
  - In the New-AzureRmFrontDoor and Set-AzureRmFrontDoor articles, fixed the link to the New-

AzureRmFrontDoorHealthProbeSettingObject cmdlet article.
    - In the New-AzureRmFrontDoorManagedRuleObject article, fixed the link to the New-AzureRmFrontDoorRuleGroupOverrideObject cmdlet article.

Az.RecoveryServices

- Added client side validations for Azure File Share restore operations.
- Made storageAccountName and storageAccountResourceGroupName optional for afs restore.

Az.Resources

- Fix for https://github.com/Azure/azure-powershell/issues/7679
    - Update Get-AzureRmRoleAssignment to use the subscription scope if it is provided when requesting classic administrators.

Az.Sql

- Minor changes for upcoming AzureRM to Az transition
- Fixed issue with using Get-AzureRmSqlDatabaseVulnerabilityAssessment with DotNet core
- Modified documentation of help messages related to SQL Auditing cmdlets.

Az.Storage

- Add -EnableHierarchicalNamespace to New-AzureRmStorageAccount
- Fix issue that Copy File cmdlet can't reuse source context in destination when not input -DestContext
    - Start-AzureStorageFileCopy
- Support Static Website configuration
    - Enable-AzureStorageStaticWebsite
    - Disable-AzureStorageStaticWebsite

Az.Websites

- Set-AzureRmWebApp and Set-AzureRmWebAppSlot
    - New parameter (-AzureStoragePath) added to specify Azure Storage paths to be mounted in Windows and Linux container apps. Use the output of the new cmdlet New-AzureRmWebAppAzureStoragePath as a parameter to set the Azure Storage paths.

0.6.1 - November 2018

Az.ApiManagement

- Update dependencies for type mapping issue

Az.Automation

- Swagger based Azure Automation cmdlets
- Added Update Management cmdlets
- Added Source Control cmdlets
- Added Remove-AzureRmAutomationHybridWorkerGroup cmdlet
- Fixed the DSC Register Node command

Az.Compute

- Fixed identity issue for SystemAssigned identity
- Update dependencies for type mapping issue

Az.ContainerInstance

- Update dependencies for type mapping issue

Az.MarketplaceOrdering

- update the examples description for marketplace cmdlets

Az.Network

- Added cmdlet New-AzureRmApplicationGatewayCustomError, Add-AzureRmApplicationGatewayCustomError, Get-AzureRmApplicationGatewayCustomError, Set-AzureRmApplicationGatewayCustomError, Remove-AzureRmApplicationGatewayCustomError, Add-AzureRmApplicationGatewayHttpListenerCustomError, Get-AzureRmApplicationGatewayHttpListenerCustomError, Set-AzureRmApplicationGatewayHttpListenerCustomError, Remove-AzureRmApplicationGatewayHttpListenerCustomError
- Added ICMP back to supported AzureFirewall Network Protocols
- Update cmdlet Test-AzureRmNetworkWatcherConnectivity, add validation on destination id, address and port.
- Fix issues with memory usage in VirtualNetwork map

Az.RecoveryServices.Backup

- Fix for modifying policy for a protected file share.
- Converted policy timezone to uppercase.

Az.RecoveryServices.SiteRecovery

- Corrected example in New-AzureRmRecoveryServicesAsrProtectableItem
- Update dependencies for type mapping issue

Az.Relay

- Added optional Parameter -KeyValue to New-AzureRmRelayKey cmdlet, which enables user to provide KeyValue.

Az.Resources

- Update help documentation for resource identity related parameters in `New-AzureRmPolicyAssignment` and `Set-AzureRmPolicyAssignment`
- Add an example for New-AzureRmPolicyDefinition that uses -Metadata
- Fix to allow case preservation in Tag keys in NetStandard: #7678 #7703

Az.ServiceFabric

- Add deprecation messages for upcoming breaking changes

Az.Sql

- Added new cmdlets for CRUD operations on Azure Sql Database Managed Instance and Azure Sql Managed Database
  - Get-AzureRmSqlInstance
  - New-AzureRmSqlInstance
  - Set-AzureRmSqlInstance
  - Remove-AzureRmSqlInstance
  - Get-AzureRmSqlInstanceDatabase
  - New-AzureRmSqlInstanceDatabase
  - Restore-AzureRmSqlInstanceDatabase
  - Remove-AzureRmSqlInstanceDatabase
- Enabled Extended Auditing Policy management on a server or a database.
  - New parameter (PredicateExpression) was added to enable filtering of audit logs.
  - Cmdlets were modified to use SQL clients instead of Legacy clients.
  - Set-AzureRmSqlServerAuditing.
  - Get-AzureRmSqlServerAuditing.

  - Set-AzureRmSqlDatabaseAuditing.
  - Get-AzureRmSqlDatabaseAuditing.
- Fixed issue with using Update-AzureRmSqlDatabaseVulnerabilityAssessmentSettings with storage account name parameter set

0.5.0 - November 2018

General

- Added Resource Completers to many core cmdlets - these alloow you to tab through existing resource names when invoking cmdlets interactively

Az.Profile

- Update common code to use latest version of ClientRuntime
- Rename param TenantId in cmdlet Connect-AzAccount to Tenant and add an alias for TenantId
- Updated TenantId description for Connect-AzAccount
- Fix error message for failed login when providing tenant domain
  - https://github.com/Azure/azure-powershell/issues/6936
- Fix issue with context name clashing for accounts with no subscriptions in tenant
  - https://github.com/Azure/azure-powershell/issues/7453
- Fix issue with DataLake endpoints when using MSI
  - https://github.com/Azure/azure-powershell/issues/7462
- Fix issue where 'Disconnect-AzAccount' would throw if not connected
  - https://github.com/Azure/azure-powershell/issues/7167

Az.CognitiveServices

- Add Get-AzCognitiveServicesAccountSkus operation.

Az.Compute

- Add Add-AzVmssVMDataDisk and Remove-AzVmssVMDataDisk cmdlets
- Get-AzVMImage shows AutomaticOSUpgradeProperties
- Fixed SetAzVMChefExtension -BootstrapOptions and -JsonAttribute option values are not setting in json format.

Az.DataLakeStore

- Update the DataLake package to 1.1.10.
- Add default Concurrency to multithreaded operations.

Az.Insights

- Fixed issue #7267 (Autoscale area)
  - Issues with creating a new autoscale rule not properly setting enumerated parameters (would always set them to the default value).
- Fixed issue #7513 [Insights] Set-AzDiagnosticSetting requires explicit specification of categories during creation of setting
  - Now the cmdlet does not require explicit indication of the categories to enable during creation, i.e. it works as it is documented

Az.Network

- Changed PeeringType to be a mandatory parameter for the following cmdlets:-
  - Get-AzExpressRouteCircuitRouteTable
  - Get-AzExpressRouteCircuitARPTable
  - Get-AzExpressRouteCircuitRouteTableSummary
  - Get-AzExpressRouteCrossConnectionArpTable

- Get-AzExpressRouteCrossConnectionRouteTable
- Get-AzExpressRouteCrossConnectionRouteTableSummary

Az.PolicyInsights

- Added policy remediation cmdlets

Az.Resources

- Fix for https://github.com/Azure/azure-powershell/issues/7402
  - Allow listing resources using the '-ResourceId' parameter for 'Get-AzResource'

Az.ServiceBus

- Added MigrationState read-only property to PSServiceBusMigrationConfigurationAttributes which will help to know the Migration state.

Az.ServiceFabric

- Fix add certificate to Linux Vmss.
- Fix 'Add-AzServiceFabricClusterCertificate'
  - Using correct thumbprint from new certificate (Azure/service-fabric-issues#932).
  - Display exception correctly (Azure/service-fabric-issues#1054).
- Fix 'Update-AzServiceFabricDurability' to update cluster configuration before starting Vmss CreateOrUpdate operation.

0.4.0 - October 2018

Az.Profile

- Fix issue with Get-AzSubscription in CloudShell
- Update common code to use latest version of ClientRuntime

Az.Compute

- Added new sizes to the whitelist of VM sizes for which accelerated networking will be turned on when using the simple param set for 'New-AzVm'
- Added ResourceName argument completer to all cmdlets.

Az.DataLakeStore

- Adding support for Virtual Network Rules
  - Get-AzDataLakeStoreVirtualNetworkRule: Gets or Lists Azure Data Lake Store virtual network rule.
  - Add-AzDataLakeStoreVirtualNetworkRule: Adds a virtual network rule to the specified Data Lake Store account.
  - Set-AzDataLakeStoreVirtualNetworkRule: Modifies the specified virtual network rule in the specified Data Lake Store account.
  - Remove-AzDataLakeStoreVirtualNetworkRule: Deletes an Azure Data Lake Store virtual network rule.

Az.Network

- Update cmdlet Test-AzNetworkWatcherConnectivity, pass the protocol value to backend.
- Added ResourceName argument completer to all cmdlets.

Az.Resources

- Fix isssue where Get-AzRoleDefinition throws an unintelligible exception (when the default profile has no subscription in it and no scope is specified) by adding a meaningful exception in the scenario. Also set the default param set to 'RoleDefinitionNameParameterSet'.

0.3.0 - October 2018

Azure.Storage

- Fix Copy Blob/File won't copy metadata when destination has metadata issue

    - Start-AzureStorageBlobCopy
    - Start-AzureStorageFileCopy
- Support get the Storage resource usage of a specific location, and add warning message for get global Storage resource usage is obsolete.
    - Get-AzStorageUsage

Az.CognitiveServices

- Support Get-AzCognitiveServicesAccountSkus without an existing account.

Az.Compute

- Fix Get-AzVM -ResourceGroupName to return more than 50 results if needed
- Added an example of the 'SimpleParameterSet' to New-AzVmss cmdlet help.
- Fixed a typo in the Azure Disk Encryption progress message

Az.DataFactoryV2

- Updated the ADF .Net SDK version to 2.3.0.

Az.Network

- Added NetworkProfile functionality. new cmdlets added
    - Get-AzNetworkProfile
    - New-AzNetworkProfile
    - Remove-AzNetworkProfile
    - Set-AzNetworkProfile
    - New-AzContainerNicConfig
    - New-AzContainerNicConfigIpConfig
- Added service association link on Subnet Model
- Added cmdlet New-AzVirtualNetworkTap, Get-AzVirtualNetworkTap, Set-AzVirtualNetworkTap, Remove-AzVirtualNetworkTap
- Added cmdlet Set-AzNEtworkInterfaceTapConfig, Get-AzNEtworkInterfaceTapConfig, Remove-AzNEtworkInterfaceTapConfig

Az.RedisCache

- Allow any string as Size parameter going forward. Add P5 in PSArgumentCompleter popup

Az.Resources

- Add missing -Mode parameter to Set-AzPolicyDefinition
- Fix Get-AzProviderOperation commandlet bug for operations with Origin containing User

Az.Sql

- Fixed issue where some backup cmdlets would not recognize the current azure subscription

Az.Websites

- New Cmdlet Get-AzWebAppContainerContinuousDeploymentUrl - Gets the Container Continuous Deployment Webhook URL
- New Cmdlets New-AzWebAppContainerPSSession and Enter-WebAppContainerPSSession - Initiates a PowerShell remote session into a windows container app

0.2.0 - September 2018

Initial Release

Migration Guide for Az 1.0.0

5/17/2019 • 8 minutes to read • Edit Online

This document describes the changes between the 6.x versions of AzureRM and Az version 1.0.0.

Table of Contents

General breaking changes

Cmdlet Noun Prefix Changes

In AzureRM, cmdlets used either 'AzureRM' or 'Azure' as a noun prefix. Az simplifies and normalizes cmndlet names, so that all cmdlets use 'Az' as their cmdlet noun prefix. For example:

```
Get-AzureRMVM
Get-AzureKeyVaultSecret
```

Have changed to

```
Get-AzVM
Get-AzKeyVaultSecret
```

To make the transition to these new cmdlet names simpler, Az introduces two new cmdlets, `Enable-AzureRmAlias` and `Disable-AzureRmAlias`. `Enable-AzureRmAlias` creates aliases from the older cmdlet names in AzureRM to the newer Az cmdlet names. The cmdlet allows creating aliases in the current session, or across all sessions by changing your user or machine profile.

For example, the following script in AzureRM:

```
#Requires -Modules AzureRM.Storage
Get-AzureRmStorageAccount | Get-AzureStorageContainer | Get-AzureStorageBlob
```

Could be run with minimal changes using `Enable-AzureRmAlias`:

```
#Requires -Modules Az.Storage
Enable-AzureRmAlias
Get-AzureRmStorageAccount | Get-AzureStorageContainer | Get-AzureStorageBlob
```

Running `Enable-AzureRmAlias -Scope CurrentUser` will enable the aliases for all powershell sessions you open, so that after executing this cmdlet, a script like this would not need to be changed at all:

```
Get-AzureRmStorageAccount | Get-AzureStorageContainer | Get-AzureStorageBlob
```

For complete details on the usage of the alias cmdlets, execute `Get-Help -Online Enable-AzureRmAlias` from the powershell prompt.

`Disable-AzureRmAlias` removes AzureRM cmdlet aliases created by `Enable-AzureRmAlias`. For complete details, execute `Get-Help -Online Disable-AzureRmAlias` from the powershell prompt.

Module Name Changes

- The module names have changed from `AzureRM.*` to `Az.*`, except for the following modules:

```
AzureRM.Profile                      -> Az.Accounts
Azure.AnalysisServices               -> Az.AnalysisServices
AzureRM.Consumption                  -> Az.Billing
AzureRM.UsageAggregates              -> Az.Billing
AzureRM.DataFactories                -> Az.DataFactory
AzureRM.DataFactoryV2                -> Az.DataFactory
AzureRM.MachineLearningCompute       -> Az.MachineLearning
AzureRM.Insights                     -> Az.Monitor
AzureRM.RecoveryServices.Backup      -> Az.RecoveryServices
AzureRM.RecoveryServices.SiteRecovery -> Az.RecoveryServices
AzureRM.Tags                         -> Az.Resources
Azure.Storage                        -> Az.Storage
```

The changes in module names mean that any script that uses `#Requires` or `Import-Module` to load specific modules will need to be changed to use the new module instead.

Migrating #Requires Statements

Scripts that use #Requires to declare a dependency on AzureRM modules should be updated to use the new module names

```
#Requires -Module AzureRM.Compute
```

Should be changed to

```
#Requires -Module Az.Compute
```

Migrating Import-Module Statements

Scripts that use `Import-Module` to load AzureRM modules will need to be updated to reflect the new module names.

```
Import-Module -Name AzureRM.Compute
```

Should be changed to

```
Import-Module -Name Az.Compute
```

Migrating Fully-Qualified Cmdlet Invocations

Scripts that use module-qualified cmdlet invocations, like

```
AzureRM.Compute\Get-AzureRmVM
```

Should be changed to use the new module and cmdlet names

```
Az.Compute\Get-AzVM
```

Migrating Module Manifest Dependencies

Modules that express dependencies on AzureRM modules through a module manifest (.psd1) file will need to updated the module names in their 'RequiredModules' section

```
RequiredModules = @(@{ModuleName="AzureRM.Profile"; ModuleVersion="5.8.2"})
```

Should be changed to

```
RequiredModules = @(@{ModuleName="Az.Profile"; ModuleVersion="1.0.0"})
```

Removed modules

- `AzureRM.Backup`
- `AzureRM.Compute.ManagedService`
- `AzureRM.Scheduler`

The tooling for these services are no longer actively supported. Customers are encouraged to move to alternative services as soon as it is convenient.

Windows PowerShell 5.1 and .NET 4.7.2

- Using Az with Windows PowerShell 5.1 requires the installation of .NET 4.7.2. However, using Az with PowerShell Core does not require .NET 4.7.2.

Temporary removal of User login using PSCredential

- Due to changes in the authentication flow for .NET Standard, we are temporarily removing user login via PSCredential. This capability will be re-introduced in the 1/15/2019 release for Windows PowerShell 5.1. This is discussed in detail in this issue.

Default Device Code login instead of Web Browser prompt

- Due to changes in the authentication flow for .NET Standard, we are using device login as the default login flow during interactive login. Web browser based login will be re-introduced for Windows PowerShell 5.1 as the default in the 1/15/2019 release. At that time, users will be able to choose device login using a Switch parameter.

Module breaking changes

Az.ApiManagement (previously AzureRM.ApiManagement)

- Removing the following cmdlets:
  - New-AzureRmApiManagementHostnameConfiguration
  - Set-AzureRmApiManagementHostnames
  - Update-AzureRmApiManagementDeployment
  - Import-AzureRmApiManagementHostnameCertificate
  - Use **Set-AzApiManagement** cmdlet to set these properites instead
- Following properties were removed
  - Removed property `PortalHostnameConfiguration`, `ProxyHostnameConfiguration`, `ManagementHostnameConfiguration` and `ScmHostnameConfiguration` of type `PsApiManagementHostnameConfiguration` from `PsApiManagementContext`. Instead use `PortalCustomHostnameConfiguration`, `ProxyCustomHostnameConfiguration`, `ManagementCustomHostnameConfiguration` and `ScmCustomHostnameConfiguration` of type `PsApiManagementCustomHostNameConfiguration`.
  - Removed property `StaticIPs` from PsApiManagementContext. The property has been split into `PublicIPAddresses` and `PrivateIPAddresses`.
  - Removed required property `Location` from New-AzureApiManagementVirtualNetwork cmdlet.

Az.Billing (previously AzureRM.Billing, AzureRM.Consumption, and AzureRM.UsageAggregates)

- The `InvoiceName` parameter was removed from the `Get-AzConsumptionUsageDetail` cmdlet. Scripts will need to use other identity parameters for the invoice.

Az.CognitiveServices (previously AzureRM.CognitiveServices)

- Removed `GetSkusWithAccountParamSetName` parameter set from `Get-AzCognitiveServicesAccountSkus` cmdlet. You must get Skus by Account Type and Location, instead of using ResourceGroupName and Account Name.

Az.Compute (previously AzureRM.Compute)

- `IdentityIds` are removed from `Identity` property in `PSVirtualMachine` and `PSVirtualMachineScaleSet` objects Scripts should no longer use the value of this field to make processing decisions.
- The type of `InstanceView` property of `PSVirtualMachineScaleSetVM` object is changed from `VirtualMachineInstanceView` to `VirtualMachineScaleSetVMInstanceView`
- `AutoOSUpgradePolicy` and `AutomaticOSUpgrade` properties are removed from `UpgradePolicy` property
- The type of `Sku` property in `PSSnapshotUpdate` object is changed from `DiskSku` to `SnapshotSku`
- `VmScaleSetVMParameterSet` is removed from `Add-AzVMDataDisk` cmdlet, you cna no longer add a data disk individually to a ScaleSet VM.

Az.DataFactory (previously AzureRM.DataFactories and AzureRM.DataFactoryV2)

- The `GatewayName` parameter has become mandatory in the `New-AzDataFactoryEncryptValue` cmdlet
- Removed `New-AzDataFactoryGatewayKey` cmdlet

- Removed `LinkedServiceName` parameter from `Get-AzDataFactoryV2ActivityRun` cmdlet Scripts should no longer use the value of this field to make processing decisions.

Az.DataLakeAnalytics (previously AzureRM.DataLakeAnalytics)

- Removed deprecated cmdlets: `New-AzDataLakeAnalyticsCatalogSecret`, `Remove-AzDataLakeAnalyticsCatalogSecret`, and `Set-AzDataLakeAnalyticsCatalogSecret`

Az.DataLakeStore (previously AzureRM.DataLakeStore)

- The following cmdlets have had the `Encoding` parameter changed from the type `FileSystemCmdletProviderEncoding` to `System.Text.Encoding`. This change removes the encoding values `String` and `Oem`. All the other prior encoding values remain.
  - New-AzureRmDataLakeStoreItem
  - Add-AzureRmDataLakeStoreItemContent
  - Get-AzureRmDataLakeStoreItemContent
- Removed deprecated `Tags` property alias from `New-AzDataLakeStoreAccount` and `Set-AzDataLakeStoreAccount` cmdlets

  Scripts using

  ```
  New-AzureRMDataLakeStoreAccount -Tags @{TagName="TagValue"}
  ```

  Should be changed to

  ```
  New-AzDataLakeStoreAccount -Tag @{TagName="TagValue"}
  ```

- Removed deprecated properties `Identity`, `EncryptionState`, `EncrypotionProvisioningState`, `EncryptionConfig`, `FirewallState`, `FirewallRules`, `VirtualNetworkRules`, `TrustedIdProviderState`, `TrustedIdProviders`, `DefaultGroup`, `NewTier`, `CurrentTier`, `FirewallAllowAzureIps` from `PSDataLakeStoreAccountBasic` object. Any script that uses the `PSDatalakeStoreAccount` returned from `Get-AzDataLakeStoreAccount` should not reference these properties.

Az.KeyVault (previously AzureRM.KeyVault)

- The `PurgeDisabled` property was removed from the `PSKeyVaultKeyAttributes`, `PSKeyVaultKeyIdentityItem`, and `PSKeyVaultSecretAttributes` objects Scripts should no longer reference the `PurgeDisabled` property to make processing decisions.

Az.Media (previously AzureRM.Media)

- Remove deprecated `Tags` property alias from `New-AzMediaService` cmdlet Scripts using

  ```
  New-AzureRMMediaService -Tags @{TagName="TagValue"}
  ```

  Should be changed to

  ```
  New-AzMMediaService -Tag @{TagName="TagValue"}
  ```

Az.Monitor (previously AzureRM.Insights)

- Removed plural names `Categories` and `Timegrains` parameter in favor of singular parameter names from `Set-AzDiagnosticSetting` cmdlet Scripts using

```
Set-AzureRmDiagnosticSetting -Timegrains PT1M -Categories Category1, Category2
```

Should be changed to

```
Set-AzDiagnosticSetting -Timegrain PT1M -Category Category1, Category2
```

Az.Network (previously AzureRM.Network)

- Removed deprecated `ResourceId` parameter from `Get-AzServiceEndpointPolicyDefinition` cmdlet
- Removed deprecated `EnableVmProtection` property from `PSVirtualNetwork` object
- Removed deprecated `Set-AzVirtualNetworkGatewayVpnClientConfig` cmdlet

Scripts shoudl no longer make processing decisions based on the values fo these fields.

Az.OperationalInsights (previously AzureRM.OperationalInsights)

- Default parameter set for `Get-AzOperationalInsightsDataSource` is removed, and `ByWorkspaceNameByKind` has become the default parameter set

  Scripts that listed data sources using

  ```
  Get-AzureRmOperationalInsightsDataSource
  ```

  Should be changed to specify a Kind

  ```
  Get-AzOperationalInsightsDataSource -Kind AzureActivityLog
  ```

Az.RecoveryServices (previously AzureRM.RecoveryServices, AzureRM.RecoveryServices.Backup, and AzureRM.RecoveryServices.SiteRecovery)

- Removed `Encryption` parameter from `New/Set-AzRecoveryServicesAsrPolicy` cmdlet
- `TargetStorageAccountName` parameter is now mandatory for managed disk restores in `Restore-AzRecoveryServicesBackupItem` cmdlet
- Removed `StorageAccountName` and `StorageAccountResourceGroupName` parameters in `Restore-AzRecoveryServicesBackupItem` cmdlet
- Removed `Name` parameter in `Get-AzRecoveryServicesBackupContainer` cmdlet

Az.Resources (previously AzureRM.Resources)

- Removed `Sku` parameter from `New/Set-AzPolicyAssignment` cmdlet
- Removed `Password` parameter from `New-AzADServicePrincipal` and `New-AzADSpCredential` cmdlet
  Passwords are automatically generated, scripts that provided the password:

  ```
  New-AzAdSpCredential -ObjectId 1f99cf81-0146-4f4e-beae-2007d0668476 -Password $secPassword
  ```

  Should be changed to retriedve the password from the output:

  ```
  $credential = New-AzAdSpCredential -ObjectId 1f99cf81-0146-4f4e-beae-2007d0668476
  $secPassword = $credential.Secret
  ```

Az.ServiceFabric (previously AzureRM.ServiceFabric)

- The following cmdlet return types have been changed:

- The property `SerivceTypeHealthPolicies` of type `ApplicationHealthPolicy` has been removed.
- The property `ApplicationHealthPolicies` of type `ClusterUpgradeDeltaHealthPolicy` has been removed.
- The property `OverrideUserUpgradePolicy` of type `ClusterUpgradePolicy` has been removed.
- These changes affect the following cmdlets:
  - Add-AzServiceFabricClientCertificate
  - Add-AzServiceFabricClusterCertificate
  - Add-AzServiceFabricNode
  - Add-AzServiceFabricNodeType
  - Get-AzServiceFabricCluster
  - Remove-AzServiceFabricClientCertificate
  - Remove-AzServiceFabricClusterCertificate
  - Remove-AzServiceFabricNode
  - Remove-AzServiceFabricNodeType
  - Remove-AzServiceFabricSetting
  - Set-AzServiceFabricSetting
  - Set-AzServiceFabricUpgradeType
  - Update-AzServiceFabricDurability
  - Update-AzServiceFabricReliability

Az.Sql (previously AzureRM.Sql)

- Removed `State` and `ResourceId` parameters from `Set-AzSqlDatabaseBackupLongTermRetentionPolicy` cmdlet
- Removed deprecated cmdlets: `Get/Set-AzSqlServerBackupLongTermRetentionVault`, `Get/Start/Stop-AzSqlServerUpgrade`, `Get/Set-AzSqlDatabaseAuditingPolicy`, `Get/Set-AzSqlServerAuditingPolicy`, `Remove-AzSqlDatabaseAuditing`, `Remove-AzSqlServerAuditing`
- Removed deprecated parameter `Current` from `Get-AzSqlDatabaseBackupLongTermRetentionPolicy` cmdlet
- Removed deprecated parameter `DatabaseName` from `Get-AzSqlServerServiceObjective` cmdlet
- Removed deprecated parameter `PrivilegedLogin` from `Set-AzSqlDatabaseDataMaskingPolicy` cmdlet

Az.Storage (previously Azure.Storage and AzureRM.Storage)

- To support creating an Oauth storage context with only the storage account name, the default parameter set has been changed to `OAuthParameterSet`
  - Example: `$ctx = New-AzureStorageContext -StorageAccountName $accountName`
- The `Location` parameter has become mandatory in the `Get-AzStorageUsage` cmdlet
- The Storage API methods now use the Task-based Asynchronous Pattern (TAP), instead of synchronous API calls.

1. Blob Snapshot

Before:

```
$b = Get-AzureStorageBlob -Container $containerName -Blob $blobName -Context $ctx
$b.ICloudBlob.Snapshot()
```

After:

```
$b = Get-AzureStorageBlob -Container $containerName -Blob $blobName -Context $ctx
$task = $b.ICloudBlob.SnapshotAsync()
$task.Wait()
$snapshot = $task.Result
```

2. Share Snapshot

Before:

```
$Share = Get-AzureStorageShare -Name $containerName -Context $ctx
$snapshot = $Share.Snapshot()
```

After:

```
$Share = Get-AzureStorageShare -Name $containerName -Context $ctx
$task = $Share.SnapshotAsync()
$task.Wait()
$snapshot = $task.Result
```

3. Undelete a soft delete blob

Before:

```
$b = Get-AzureStorageBlob -Container $containerName -Blob $blobName -IncludeDeleted -Context $ctx
$b.ICloudBlob.Undelete()
```

After:

```
$b = Get-AzureStorageBlob -Container $containerName -Blob $blobName -IncludeDeleted -Context $ctx
$task = $b.ICloudBlob.UndeleteAsync()
$task.Wait()
```

4. Set Blob Tier

Before:

```
$blockBlob = Get-AzureStorageBlob -Container $containerName -Blob $blockBlobName -Context $ctx
$blockBlob.ICloudBlob.SetStandardBlobTier("hot")

$pageBlob = Get-AzureStorageBlob -Container $containerName -Blob $pageBlobName -Context $ctx
$pageBlob.ICloudBlob.SetPremiumBlobTier("P4")
```

After:

```
$blockBlob = Get-AzureStorageBlob -Container $containerName -Blob $blockBlobName -Context $ctx
$task = $blockBlob.ICloudBlob.SetStandardBlobTierAsync("hot")
$task.Wait()

$pageBlob = Get-AzureStorageBlob -Container $containerName -Blob $pageBlobName -Context $ctx
$task = $pageBlob.ICloudBlob.SetPremiumBlobTierAsync("P4")
$task.Wait()
```

Az.Websites (previously AzureRM.Websites)

- Removed deprecated properties from the `PSAppServicePlan`, `PSCertificate`, `PSCloningInfo`, and `PSSite` objects