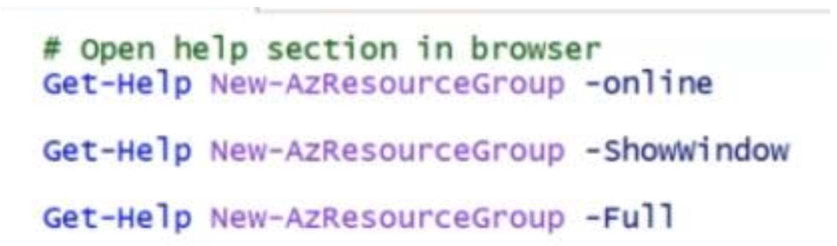
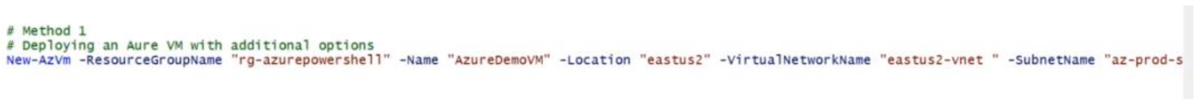
Azure-PowerShell



```
# Open help section in browser
Get-Help New-AzResourceGroup -online

Get-Help New-AzResourceGroup -ShowWindow

Get-Help New-AzResourceGroup -Full
```

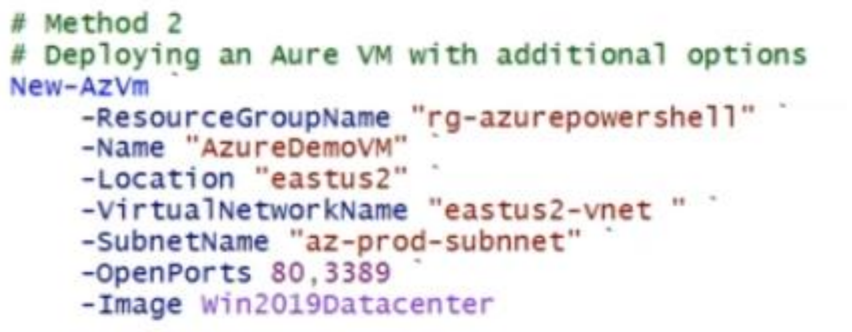
- Creating VM in 3 ways

1. Writing 1 line code

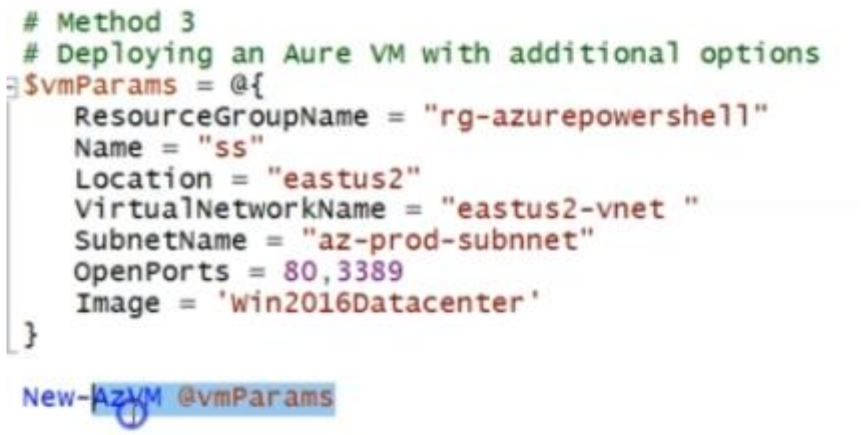


```
# Method 1
# Deploying an Aure VM with additional options
New-AzVm -ResourceGroupName "rg-azurepowershell" -Name "AzureDemoVM" -Location "eastus2" -VirtualNetworkName "eastus2-vnet " -SubnetName "az-prod-s
```

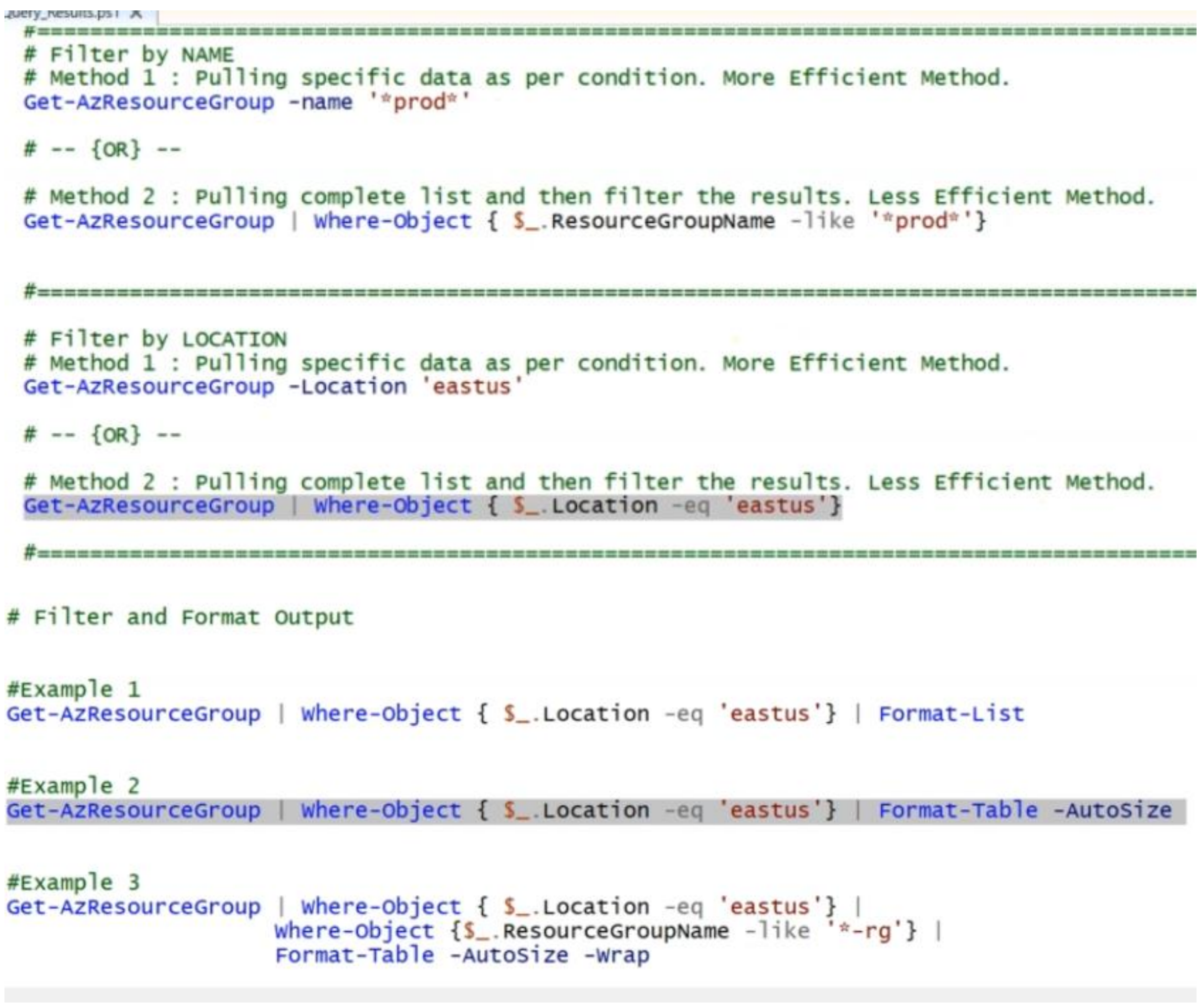
2. Breaking script in readable
3. Writing parameters



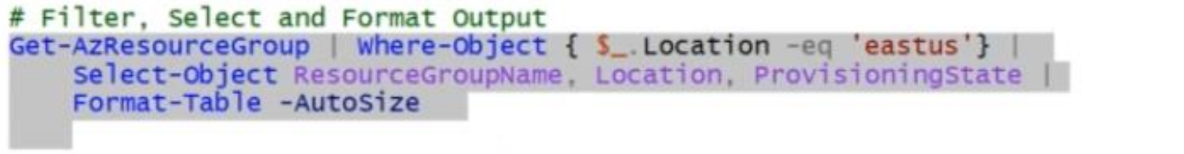
```
# Method 2
# Deploying an Aure VM with additional options
New-AzVm `
    -ResourceGroupName "rg-azurepowershell" `
    -Name "AzureDemoVM" `
    -Location "eastus2" `
    -VirtualNetworkName "eastus2-vnet " `
    -SubnetName "az-prod-subnnet" `
    -OpenPorts 80,3389 `
    -Image Win2019Datacenter
```



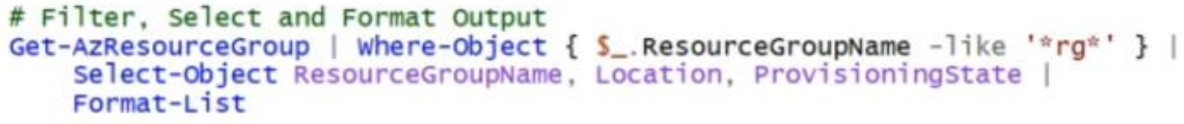
```
# Method 3
# Deploying an Aure VM with additional options
$vmParams = @{
    ResourceGroupName = "rg-azurepowershell"
    Name = "ss"
    Location = "eastus2"
    VirtualNetworkName = "eastus2-vnet "
    SubnetName = "az-prod-subnnet"
    OpenPorts = 80,3389
    Image = 'Win2016Datacenter'
}

New-AzVM @vmParams
```



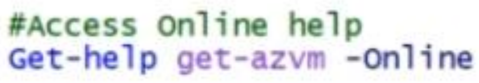
```
#==========================================================================================
# Filter by NAME
# Method 1 : Pulling specific data as per condition. More Efficient Method.
Get-AzResourceGroup -name '*prod*'

# -- {OR} --

# Method 2 : Pulling complete list and then filter the results. Less Efficient Method.
Get-AzResourceGroup | Where-Object { $_.ResourceGroupName -like '*prod*'}


#==========================================================================================

# Filter by LOCATION
# Method 1 : Pulling specific data as per condition. More Efficient Method.
Get-AzResourceGroup -Location 'eastus'

# -- {OR} --

# Method 2 : Pulling complete list and then filter the results. Less Efficient Method.
Get-AzResourceGroup | Where-Object { $_.Location -eq 'eastus'}

#==========================================================================================


# Filter and Format Output


#Example 1
Get-AzResourceGroup | Where-Object { $_.Location -eq 'eastus'} | Format-List


#Example 2
Get-AzResourceGroup | Where-Object { $_.Location -eq 'eastus'} | Format-Table -AutoSize


#Example 3
Get-AzResourceGroup | Where-Object { $_.Location -eq 'eastus'} |
                     Where-Object {$_.ResourceGroupName -like '*-rg'} |
                     Format-Table -AutoSize -Wrap
```



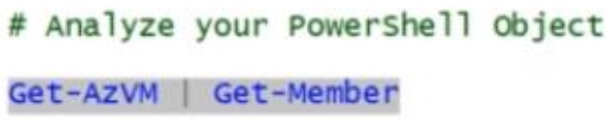
```
# Filter, Select and Format Output
Get-AzResourceGroup | Where-Object { $_.Location -eq 'eastus'} |
    Select-Object ResourceGroupName, Location, ProvisioningState |
    Format-Table -AutoSize
```



```
# Filter, Select and Format Output
Get-AzResourceGroup | Where-Object { $_.ResourceGroupName -like '*rg*' } |
    Select-Object ResourceGroupName, Location, ProvisioningState |
    Format-List
```



```
#Access Online help
Get-help get-azvm -Online
```



```
# Analyze your PowerShell Object

Get-AzVM | Get-Member
```

```
# We can access the singular properties values ( like String integer, boolean etc ).
Get-AzVM | select name, LicenseType, Location, VmId, Type, StatusCode, RequestId, ResourceGroupName
```

```
# But we CANNOT directly access the values of Collection objects or Arrays that are stored as a property.
Get-AzVM | select HardwareProfile, StorageProfile, OSProfile, BillingProfile
```

```
#ASSIGNMENT

#1.) List all the Virtual Machine names in Azure within a given resource group(say demo1_group)
#      whose name starts with  "prod" and ends with "webserver".
#     Output should be formatted in a table with only VM Name and its Resource group as columns
```

```
Get-AzVm -Name 'prod*webserver' -ResourceGroupName demo1_group |
               Select Name, ResourceGroupName |
               Format-Table -AutoSize
```

```
# 2.) List Virtual Machines within eastus2 or westus2 location.
    # Output should be in List format using properties: name, location, ResourceGroupName, ProvisioningState
```

```
Get-AzVm   | Where-Object {    ($_.Location -eq 'eastus2') -or  ($_.Location -eq 'westus2')    } |
       Format-List name, location, ResourceGroupName, ProvisioningState
```

```
# 3.) List all Virtual Machine that are in deallocated state. Display only VM name and PowerState
    # tip:  To check the powerstate we need to pass -status switch
     Get-AzVM  -Status | Select-Object  -Property Name, PowerState
```

```
# Step A.) All VMs in deallocated state
Get-AzVM  -Status | Where-Object {$_.PowerState -eq 'VM deallocated'}
```

```
# Step B.) VMs filtered and necessary columns selected in output
Get-AzVM  -Status | Where-Object {$_.PowerState -eq 'VM deallocated'} | select name, powerstate

#     {OR}

Get-AzVM  -Status | Where-Object {$_.PowerState -eq 'VM deallocated'}  |
    Select-Object  -Property Name, @{name='VM Power Status';  Expression = {$_.PowerState}}
```

```
# 4.) List Virtual Machines that are in running state and VMs are in eastus2 location.

Get-AzVM  -Status | Where-Object {$_.PowerState -eq 'VM running'  -and $_.Location -eq 'eastus2'}
```

```
# 5.) List Virtual Machines that are NOT in eastus2 region.


Get-AzVM  -Status | Where-Object { $_.Location -ne 'eastus2'}

#------ {OR}

Get-AzVM  -Status | Where-Object { -not ( $_.Location -eq 'eastus2') }
```

```
# Assignment Level II

# 6.) List all Virtual Machines with their OS. Display only VM name and OSType
     # tip: To get a OS property we need to expand its StorageProfile property

    Get-AzVm | select *


    Get-AzVm | select StorageProfile -ExpandProperty StorageProfile
```

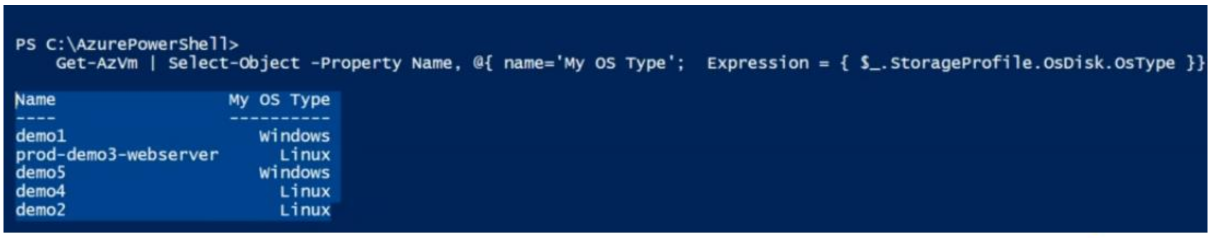
```
    Get-AzVm | select StorageProfile -ExpandProperty StorageProfile | select OsDisk -ExpandProperty OsDisk
```

```
    Get-AzVm | select StorageProfile -ExpandProperty StorageProfile |
            select OsDisk -ExpandProperty OsDisk |
            select OsType -ExpandProperty OsType
```

```
 Get-AzVm | select StorageProfile -ExpandProperty StorageProfile |
            select OsDisk -ExpandProperty OsDisk |
            select OsType -ExpandProperty OsType |
            select name, OsType
```

```
Get-AzVm | select StorageProfile -ExpandProperty StorageProfile |
           select OsDisk -ExpandProperty OsDisk |
           select OsType -ExpandProperty OsType |
           select name, OsType
```

```
    Get-AzVm | Select-Object -Property Name, @{ name='My OS Type';  Expression = { $_.StorageProfile.OsDisk.OsType }}
```



```
PS C:\AzurePowerShell>
    Get-AzVm | Select-Object -Property Name, @{ name='My OS Type';  Expression = { $_.StorageProfile.OsDisk.OsType }}

Name                   My OS Type
----                   ----------
demo1                     Windows
prod-demo3-webserver        Linux
demo5                     Windows
demo4                       Linux
demo2                       Linux
```

```
# 7.) List all Virtual Machines that has Linux OS
Get-AzVm | Where-Object  {$_.StorageProfile.OsDisk.OsType -eq 'Linux'}


     # To list Windows VMs
     #Get-AzVm | Where-Object  {$_.StorageProfile.OsDisk.OsType -eq 'Windows'}
```

```
# 8.) List all VMs with their VM Size
Get-AzVm | Select-Object -Property Name, @{name='Size';  Expression = {$_.HardwareProfile.VmSize}}
```

```
# 9.) 8.    List all Virtual Machines that are in D series of VM size
Get-AzVm | Where-Object  {$_.HardwareProfile.VmSize -like '*_D*'}
```

```
# 10.) List all Virtual Machines that are of D series Vm size and resource group name contains word 'demo'
        # Output should be a table with only 3 columns : VM Name, VM Size, VM OS
Get-AzVm | Select-Object HardwareProfile -ExpandProperty HardwareProfile | select VmSize


Get-AzVm | Where-Object  {$_.HardwareProfile.VMSize -like '*_D*'    -and $_.ResourceGroupName -like 'demo*' } |
            Select-Object -Property Name `
                                    , @{name='VMSize';  Expression = {$_.HardwareProfile.VmSize}} `
                                    , @{name='OsType';  Expression = {$_.StorageProfile.OsDisk.OsType}}
```

```
# 11.) List all Virtual Machine that satisfy below conditions
#     Status :          stopped
#     Region  :         eus2
#     Resource Group :  demo_group1
#     OS     :          Windows (any version)

# In output, display only  VM name, ProvisioningState, VMSize and OSType


Get-AzVm -Status | Where-Object  { $_.PowerState -like '*deallocated*' `
                            -and $_.Location -like 'eastus2' `
                            -and $_.ResourceGroupName -eq 'demo1_group' `
                            -and $_.StorageProfile.OsDisk.OsType -like '*windows*' `
                          } |
                    Select-Object -Property Name, ProvisioningState `
                                    , @{name='VMSize';  Expression = {$_.HardwareProfile.VmSize}} `
                                    , @{name='OsType';  Expression = {$_.StorageProfile.OsDisk.OsType}}
```

You just have to play whatever you have

```
Get-AzVM   | select Name,Type, Location, StatusCode
```

```
# VM data export to CSV
Get-AzVM   | select Name,Type, Location, StatusCode   |  Export-Csv -Path 'azure_vms.csv'

Get-AzVM   | select Name,Type, Location, StatusCode   |  Export-Csv -Path 'azure_vms.csv'  -NoTypeInformation
```

```
# VM data export to CSV
Get-AzVM   | select Name,Type, Location, StatusCode   |  Export-Csv -Path 'azure_vms.csv'

Get-AzVM   | select Name,Type, Location, StatusCode   |  Export-Csv -Path 'azure_vms.csv'  -NoTypeInformation
```

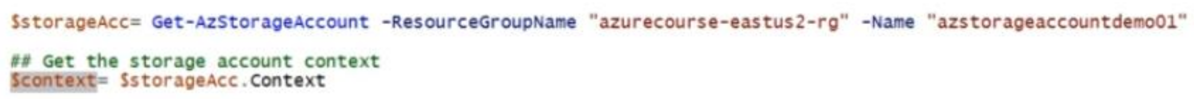
```
# Export all resources by resourcegroup name
Get-AzResource -ResourceGroupName 'demo*' |  Export-Csv -Path 'azure_resources_in_demo_rg.csv'  -NoTypeInformation
```

```
# Grabbing the necessary data and storing it in $data
$data = Get-AzResource |  Where-Object { $_.ResourceGroupName -like '*demo*'} |
                          Where-Object { $_.ResourceType -eq 'Microsoft.Compute/virtualMachines' } |
                          Select ResourceGroupName, Name, ResourceType
```

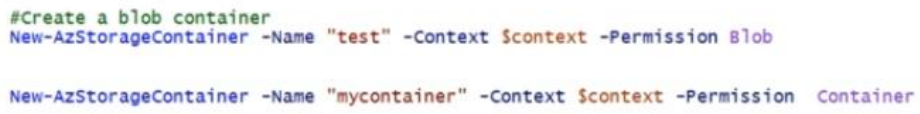
```
# Export to JSON Format
$data |  ConvertTo-Json  | Out-File "json_format_data.json"
```

```
#Create a new storage account. Make sure storage account name is unique
New-AzStorageAccount -ResourceGroupName 'azurecourse-eastus2-rg' `
   -Name azstorageaccountdemo01 `
   -Location northeurope `
   -SkuName Standard_RAGRS `
   -Kind StorageV2
```

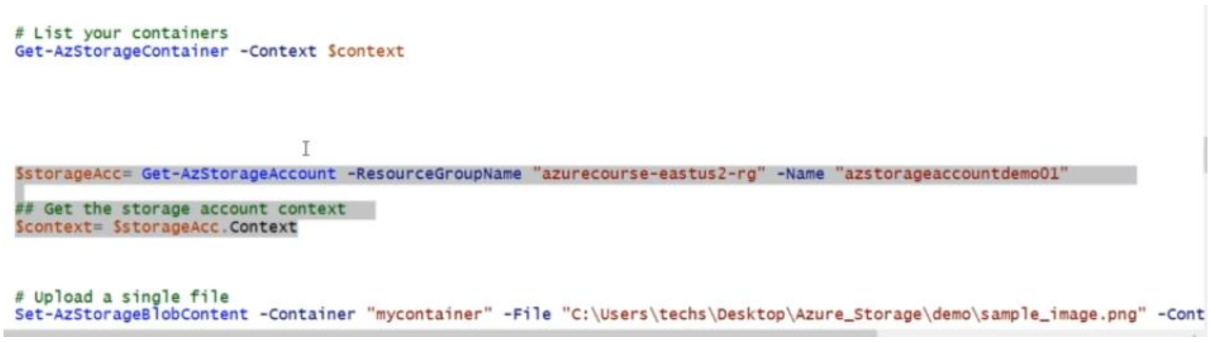
```
# To get the list of locations
 Get-AzLocation | select Location
```



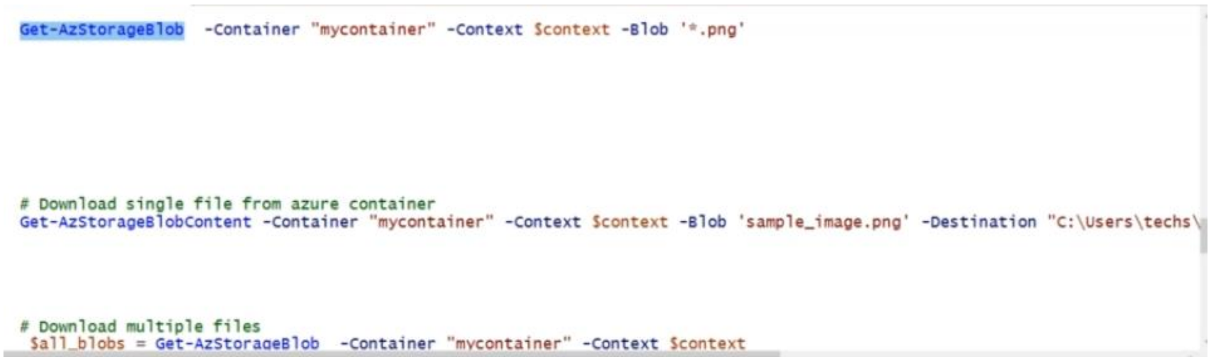
```
$storageAcc= Get-AzStorageAccount -ResourceGroupName "azurecourse-eastus2-rg" -Name "azstorageaccountdemo01"

## Get the storage account context
$context= $storageAcc.Context
```



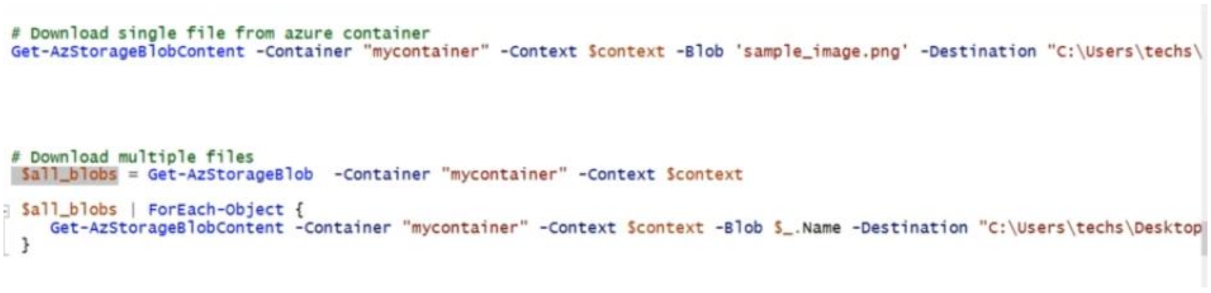
```
#Create a blob container
New-AzStorageContainer -Name "test" -Context $context -Permission Blob

New-AzStorageContainer -Name "mycontainer" -Context $context -Permission  Container
```



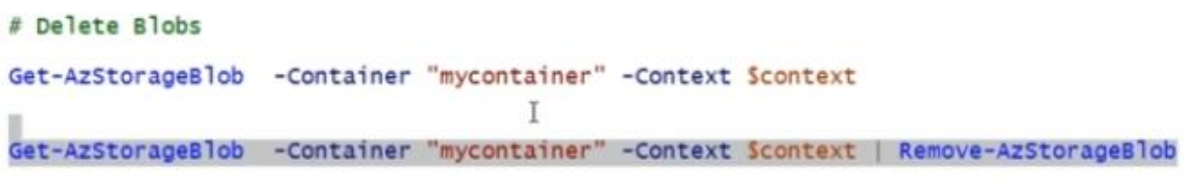
```
# List your containers
Get-AzStorageContainer -Context $context



$storageAcc= Get-AzStorageAccount -ResourceGroupName "azurecourse-eastus2-rg" -Name "azstorageaccountdemo01"

## Get the storage account context
$context= $storageAcc.Context


# Upload a single file
Set-AzStorageBlobContent -Container "mycontainer" -File "C:\Users\techs\Desktop\Azure_Storage\demo\sample_image.png" -Cont
```



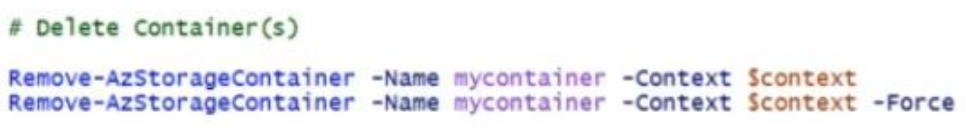
```
Get-AzStorageBlob  -Container "mycontainer" -Context $context -Blob '*.png'




# Download single file from azure container
Get-AzStorageBlobContent -Container "mycontainer" -Context $context -Blob 'sample_image.png' -Destination "C:\Users\techs\


# Download multiple files
 $all_blobs = Get-AzStorageBlob  -Container "mycontainer" -Context $context
```



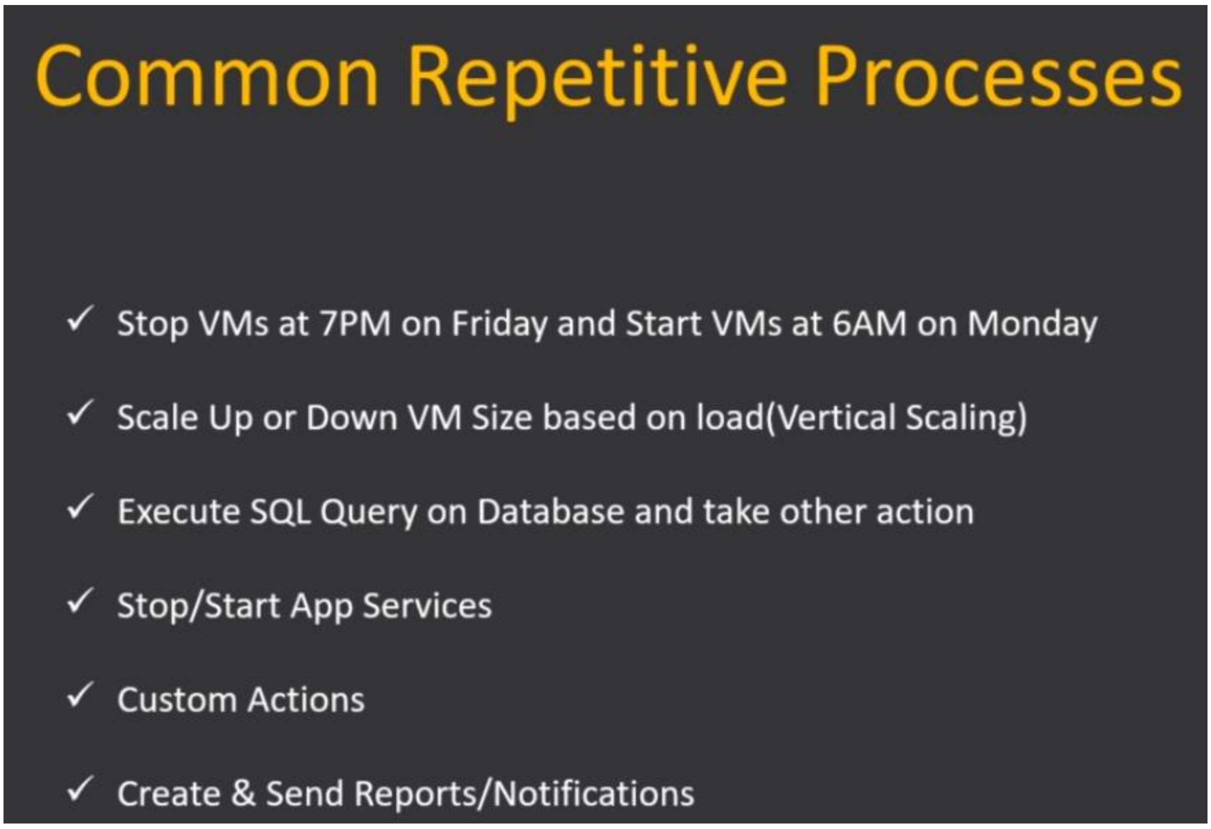
```
# Download single file from azure container
Get-AzStorageBlobContent -Container "mycontainer" -Context $context -Blob 'sample_image.png' -Destination "C:\Users\techs\


# Download multiple files
 $all_blobs = Get-AzStorageBlob  -Container "mycontainer" -Context $context

 $all_blobs | ForEach-Object {
    Get-AzStorageBlobContent -Container "mycontainer" -Context $context -Blob $_.Name -Destination "C:\Users\techs\Desktop
 }
```



```
# Delete Blobs

Get-AzStorageBlob  -Container "mycontainer" -Context $context

Get-AzStorageBlob  -Container "mycontainer" -Context $context | Remove-AzStorageBlob
```

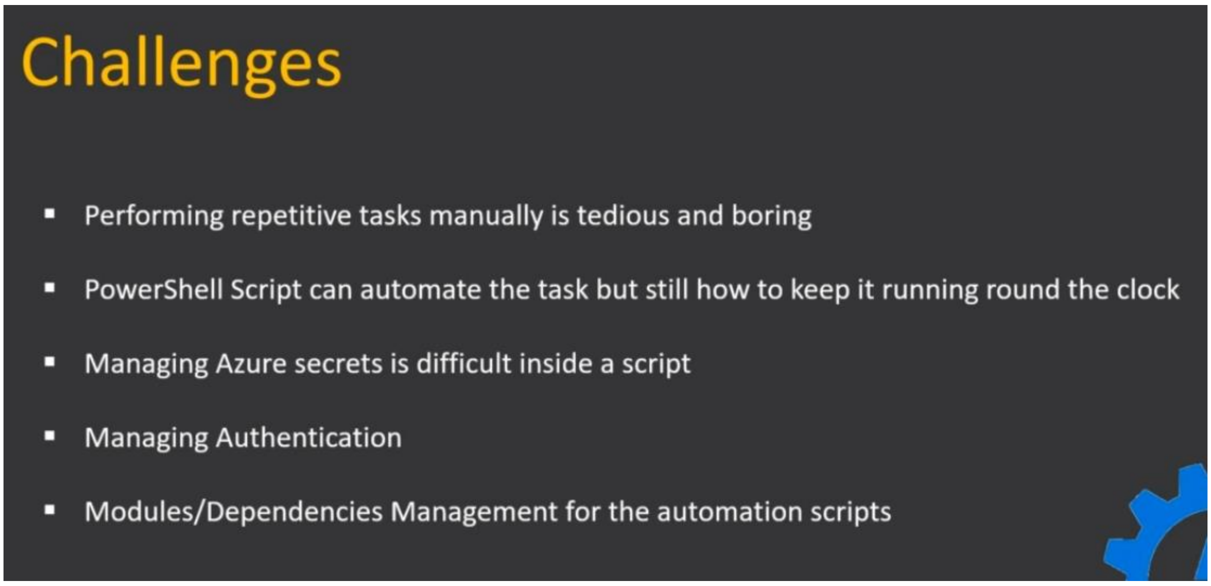


```
# Delete Container(s)

Remove-AzStorageContainer -Name mycontainer -Context $context
Remove-AzStorageContainer -Name mycontainer -Context $context -Force
```



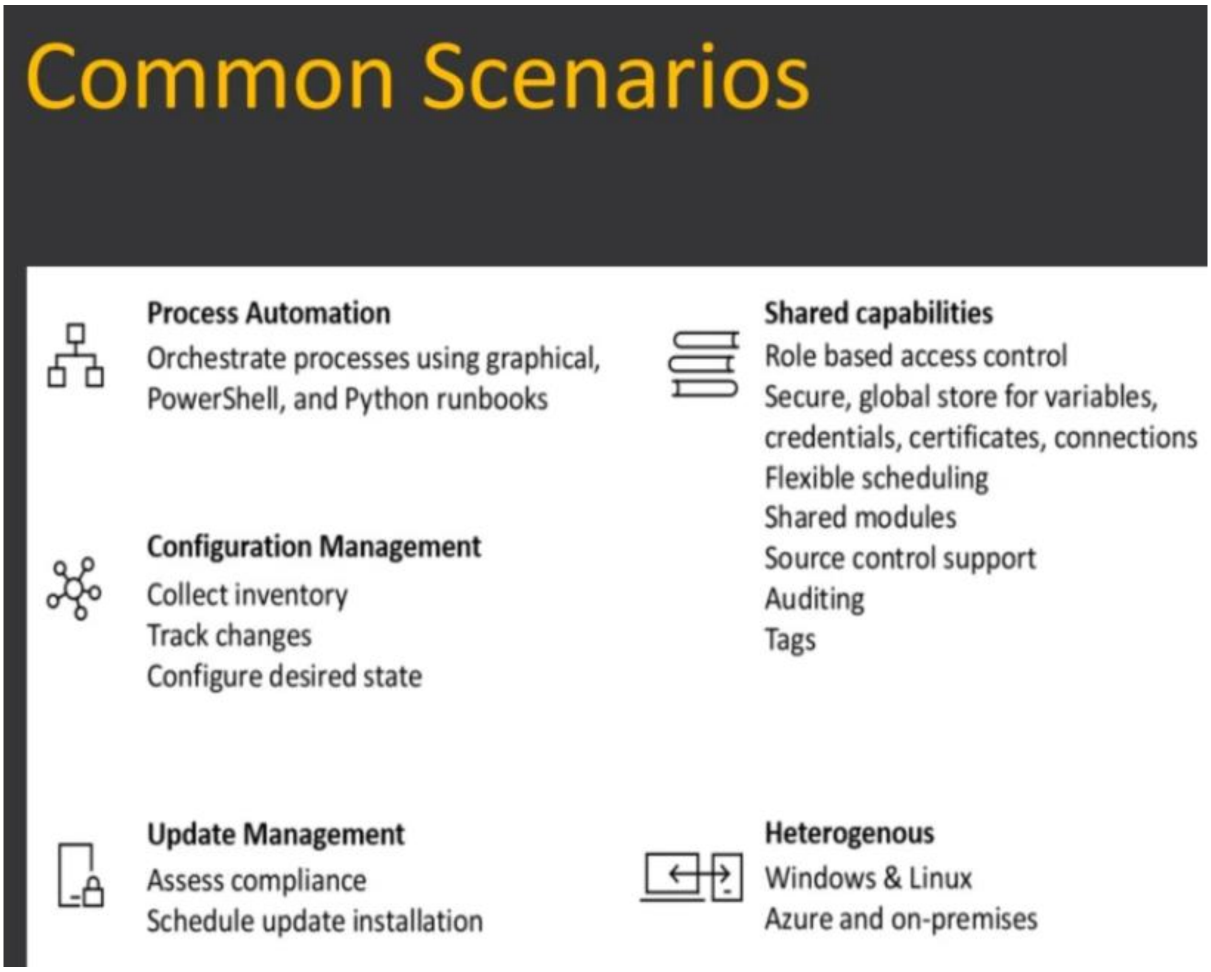
Common Repetitive Processes

- ✓ Stop VMs at 7PM on Friday and Start VMs at 6AM on Monday
- ✓ Scale Up or Down VM Size based on load(Vertical Scaling)
- ✓ Execute SQL Query on Database and take other action
- ✓ Stop/Start App Services
- ✓ Custom Actions
- ✓ Create & Send Reports/Notifications



Challenges

- Performing repetitive tasks manually is tedious and boring
- PowerShell Script can automate the task but still how to keep it running round the clock
- Managing Azure secrets is difficult inside a script
- Managing Authentication
- Modules/Dependencies Management for the automation scripts



Common Scenarios

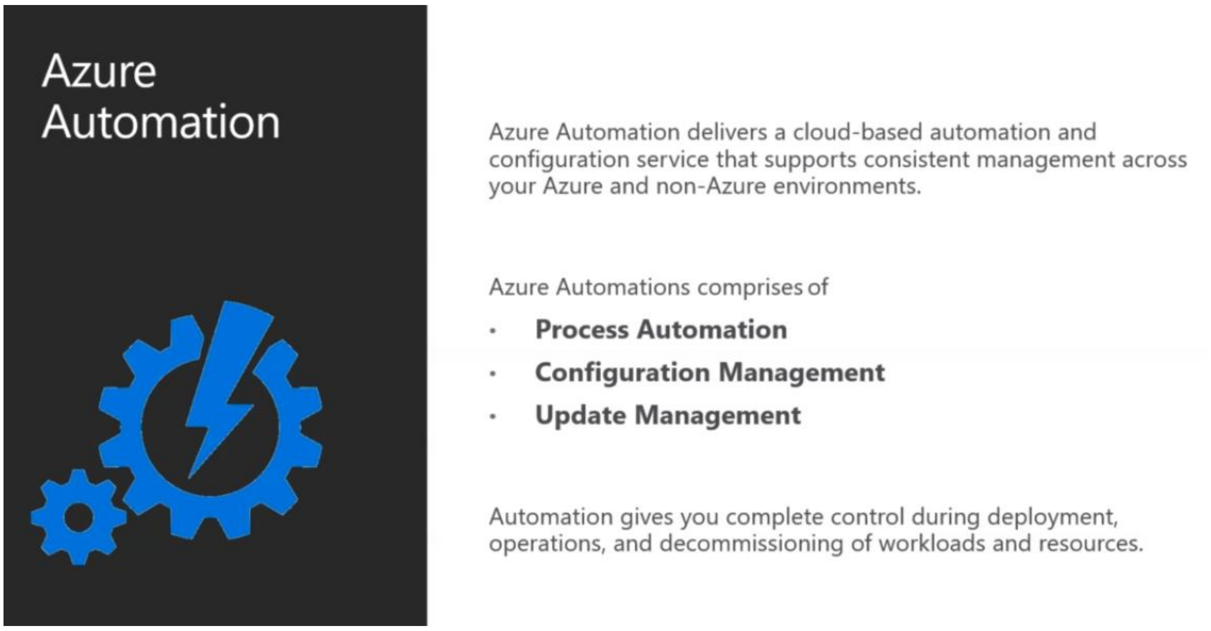
Process Automation
Orchestrate processes using graphical, PowerShell, and Python runbooks

Configuration Management
Collect inventory
Track changes
Configure desired state

Update Management
Assess compliance
Schedule update installation

Shared capabilities
Role based access control
Secure, global store for variables, credentials, certificates, connections
Flexible scheduling
Shared modules
Source control support
Auditing
Tags

Heterogenous
Windows & Linux
Azure and on-premises



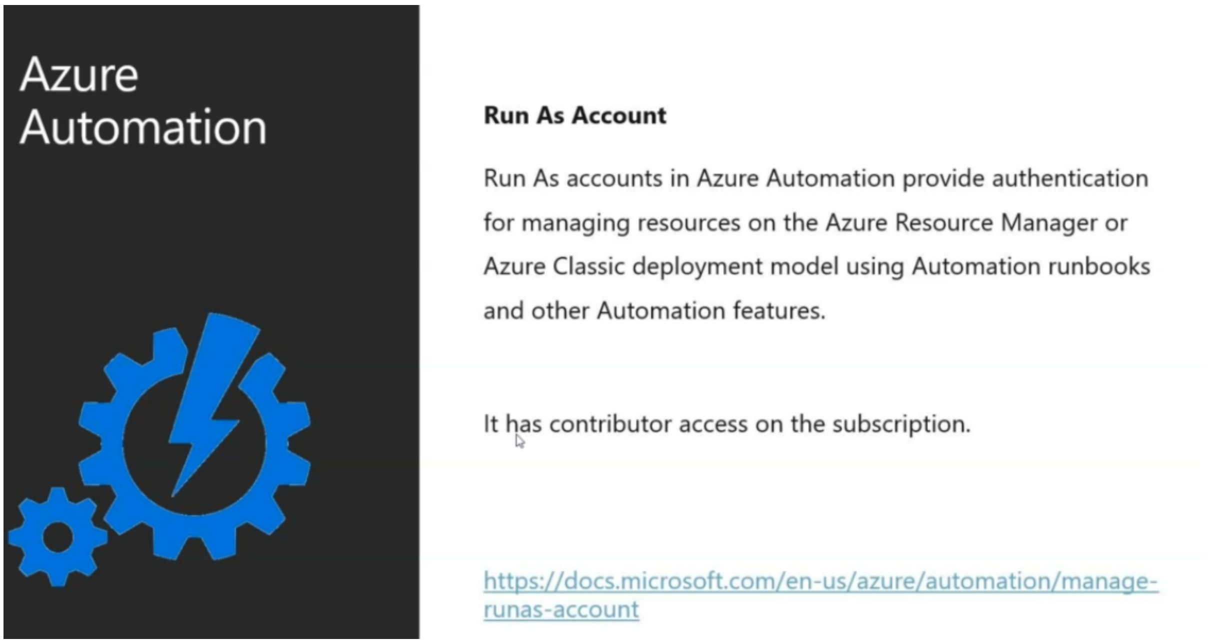
Azure Automation

Azure Automation delivers a cloud-based automation and configuration service that supports consistent management across your Azure and non-Azure environments.

Azure Automations comprises of

- **Process Automation**
- **Configuration Management**
- **Update Management**

Automation gives you complete control during deployment, operations, and decommissioning of workloads and resources.



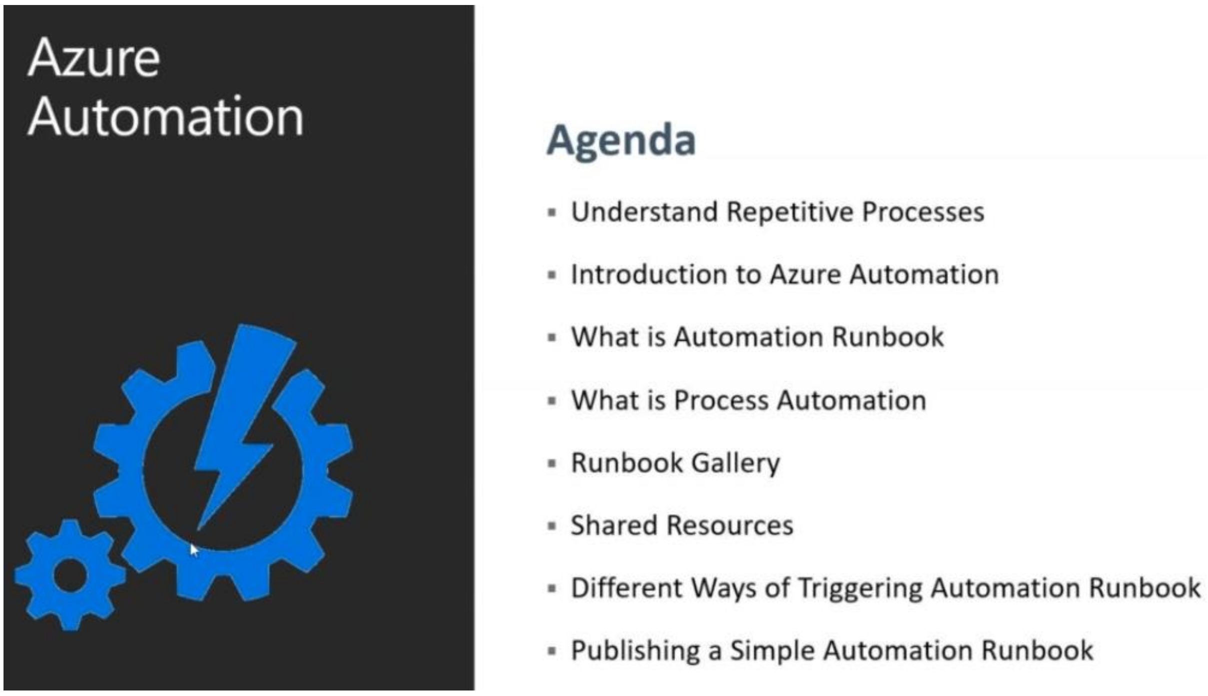
Azure Automation

Run As Account

Run As accounts in Azure Automation provide authentication for managing resources on the Azure Resource Manager or Azure Classic deployment model using Automation runbooks and other Automation features.

It has contributor access on the subscription.

https://docs.microsoft.com/en-us/azure/automation/manage-runas-account



Azure Automation

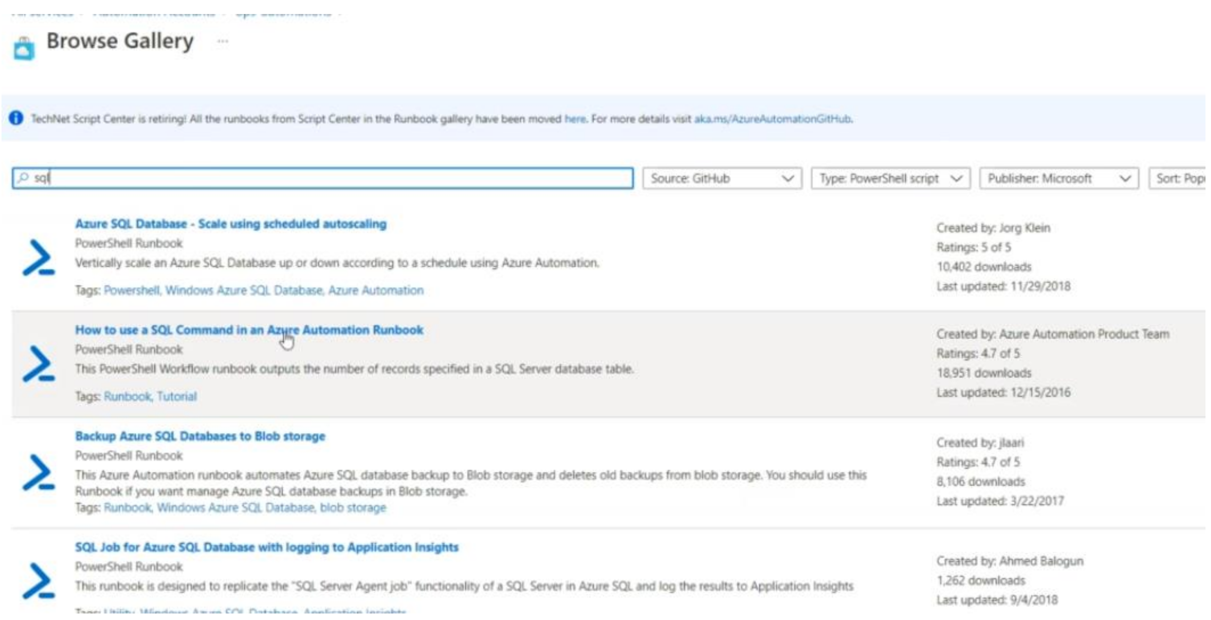
Agenda

- Understand Repetitive Processes
- Introduction to Azure Automation
- What is Automation Runbook
- What is Process Automation
- Runbook Gallery
- Shared Resources
- Different Ways of Triggering Automation Runbook
- Publishing a Simple Automation Runbook

Process Automation

Process Automation in Azure Automation allows you to automate frequent, time-consuming, and error-prone cloud management tasks.

This service helps you focus on work that adds business value. By reducing errors and boosting efficiency, it also helps to lower your operational costs.



Browse Gallery

TechNet Script Center is retiring! All the runbooks from Script Center in the Runbook gallery have been moved here. For more details visit aka.ms/AzureAutomationGitHub.

sql | Source: GitHub | Type: PowerShell script | Publisher: Microsoft | Sort: Pop

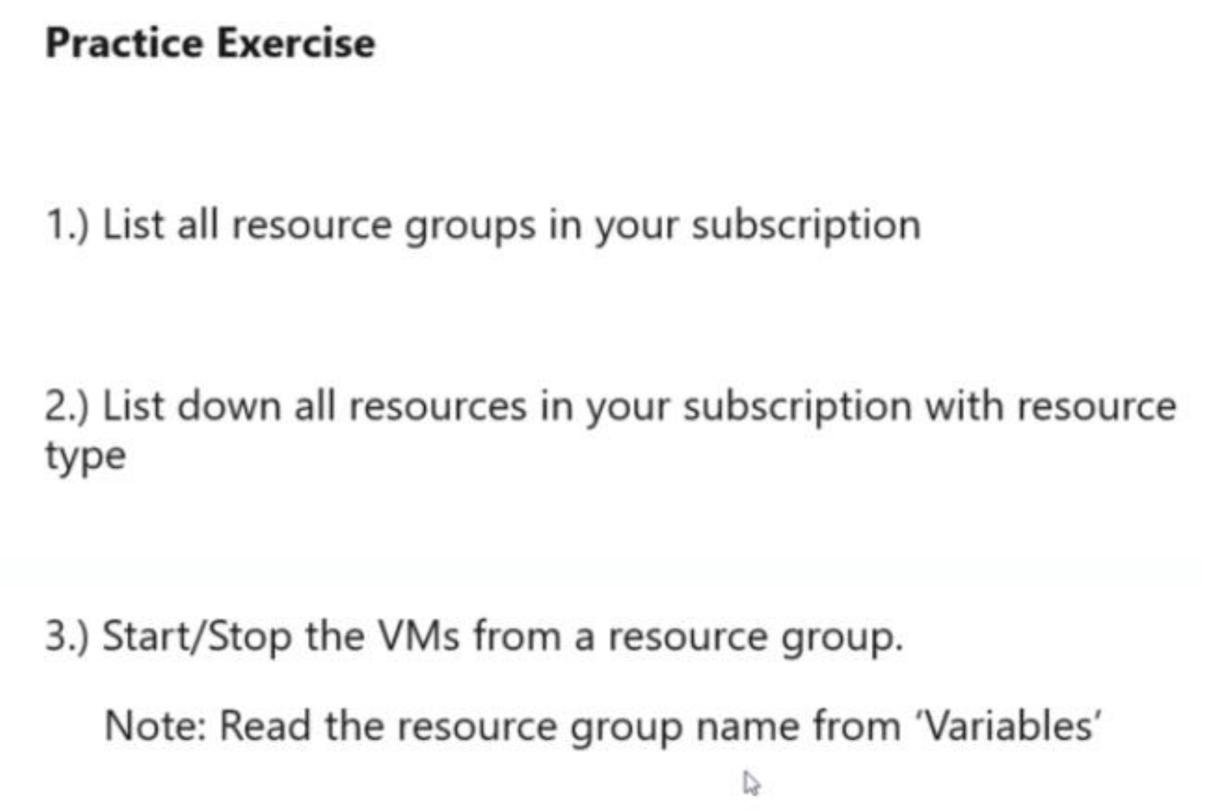
Azure SQL Database - Scale using scheduled autoscaling
PowerShell Runbook
Vertically scale an Azure SQL Database up or down according to a schedule using Azure Automation.
Tags: Powershell, Windows Azure SQL Database, Azure Automation
Created by: Jorg Klein
Ratings: 5 of 5
10,402 downloads
Last updated: 11/29/2018

How to use a SQL Command in an Azure Automation Runbook
PowerShell Runbook
This PowerShell Workflow runbook outputs the number of records specified in a SQL Server database table.
Tags: Runbook, Tutorial
Created by: Azure Automation Product Team
Ratings: 4.7 of 5
18,951 downloads
Last updated: 12/15/2016

Backup Azure SQL Databases to Blob storage
PowerShell Runbook
This Azure Automation runbook automates Azure SQL database backup to Blob storage and deletes old backups from blob storage. You should use this Runbook if you want manage Azure SQL database backups in Blob storage.
Tags: Runbook, Windows Azure SQL Database, blob storage
Created by: jlaari
Ratings: 4.7 of 5
8,106 downloads
Last updated: 3/22/2017

SQL Job for Azure SQL Database with logging to Application Insights
PowerShell Runbook
This runbook is designed to replicate the "SQL Server Agent job" functionality of a SQL Server in Azure SQL and log the results to Application Insights
Created by: Ahmed Balogun
1,262 downloads
Last updated: 9/4/2018

- Above are the runbooks created by azure we can use them by importing



Practice Exercise

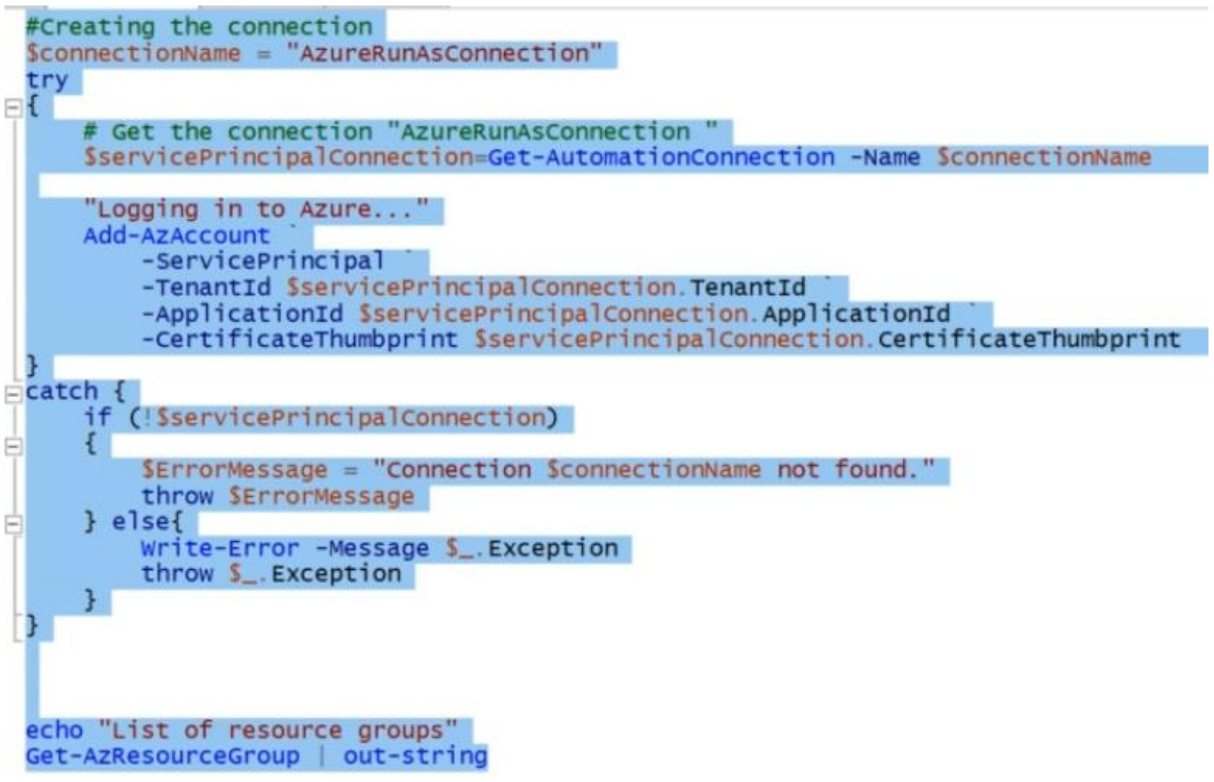
1.) List all resource groups in your subscription

2.) List down all resources in your subscription with resource type

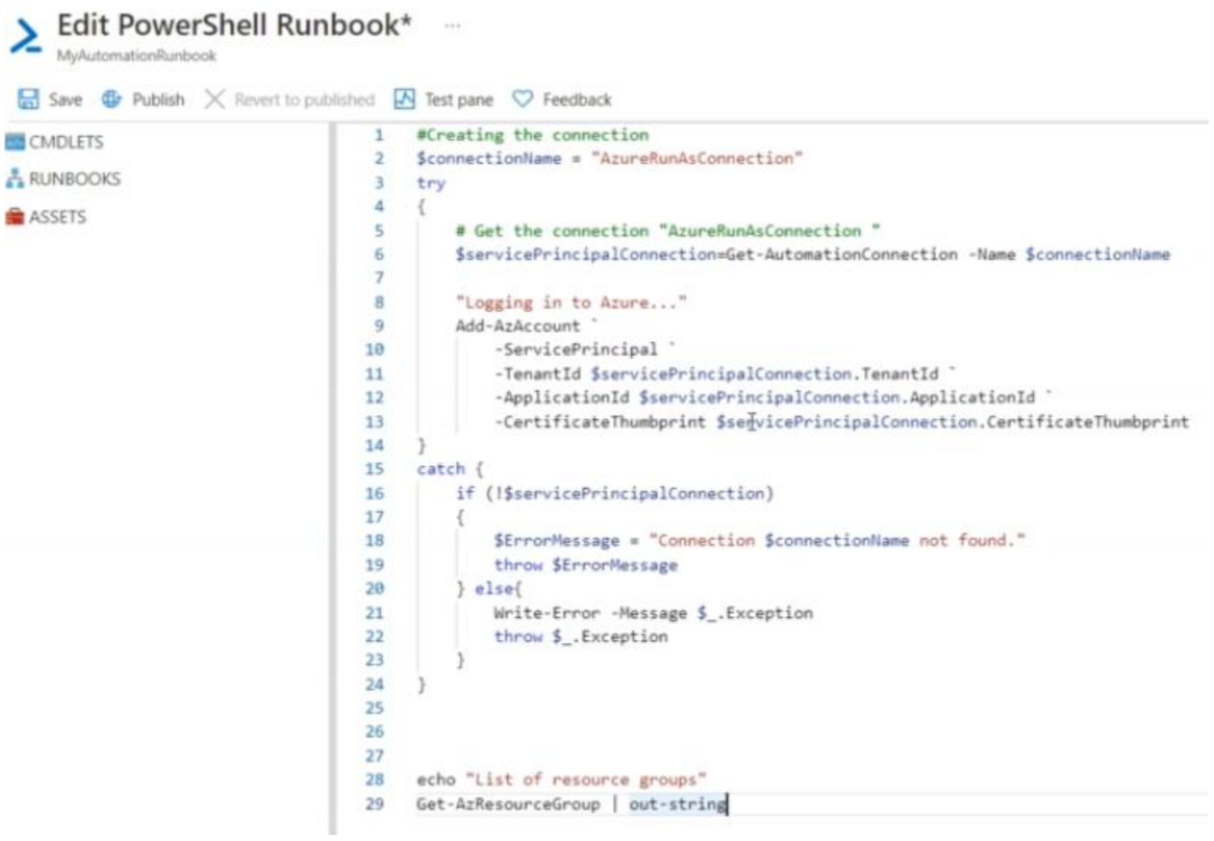
3.) Start/Stop the VMs from a resource group.

Note: Read the resource group name from 'Variables'

- Before start writing run books we need to install Az.resources module on automation account on moules then brows for az.resources and az.accounts and import both one by one
- Before writing and executing the runbood we need to add service connection information on script as below

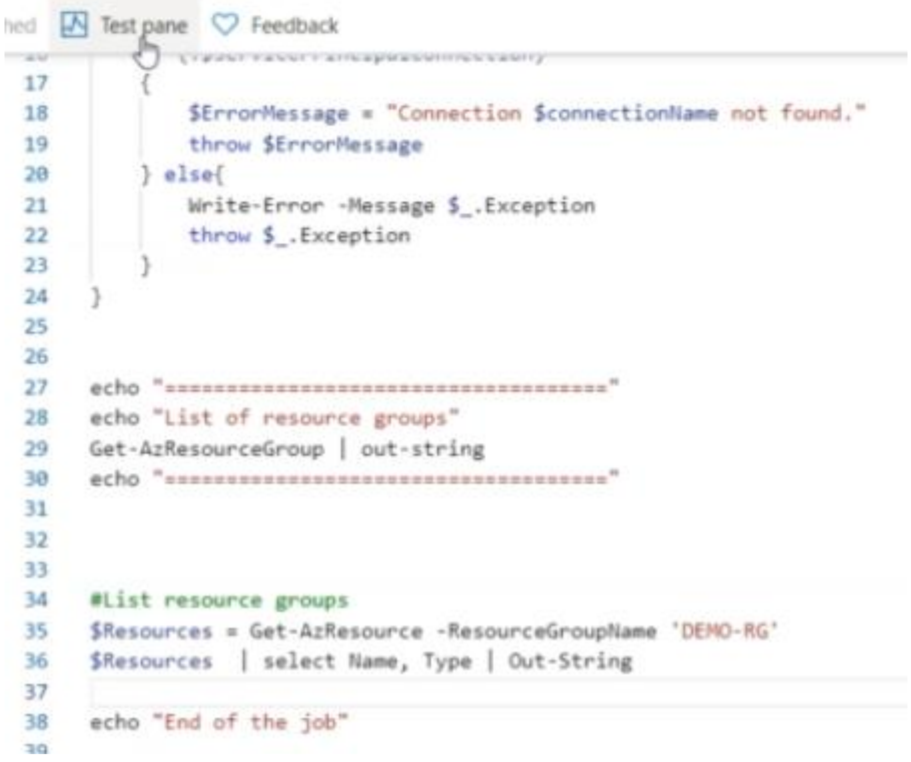


```
#Creating the connection
$connectionName = "AzureRunAsConnection"
try
{
    # Get the connection "AzureRunAsConnection "
    $servicePrincipalConnection=Get-AutomationConnection -Name $connectionName

    "Logging in to Azure..."
    Add-AzAccount `
        -ServicePrincipal `
        -TenantId $servicePrincipalConnection.TenantId `
        -ApplicationId $servicePrincipalConnection.ApplicationId `
        -CertificateThumbprint $servicePrincipalConnection.CertificateThumbprint
}
catch {
    if (!$servicePrincipalConnection)
    {
        $ErrorMessage = "Connection $connectionName not found."
        throw $ErrorMessage
    } else{
        Write-Error -Message $_.Exception
        throw $_.Exception
    }
}



echo "List of resource groups"
Get-AzResourceGroup | out-string
```



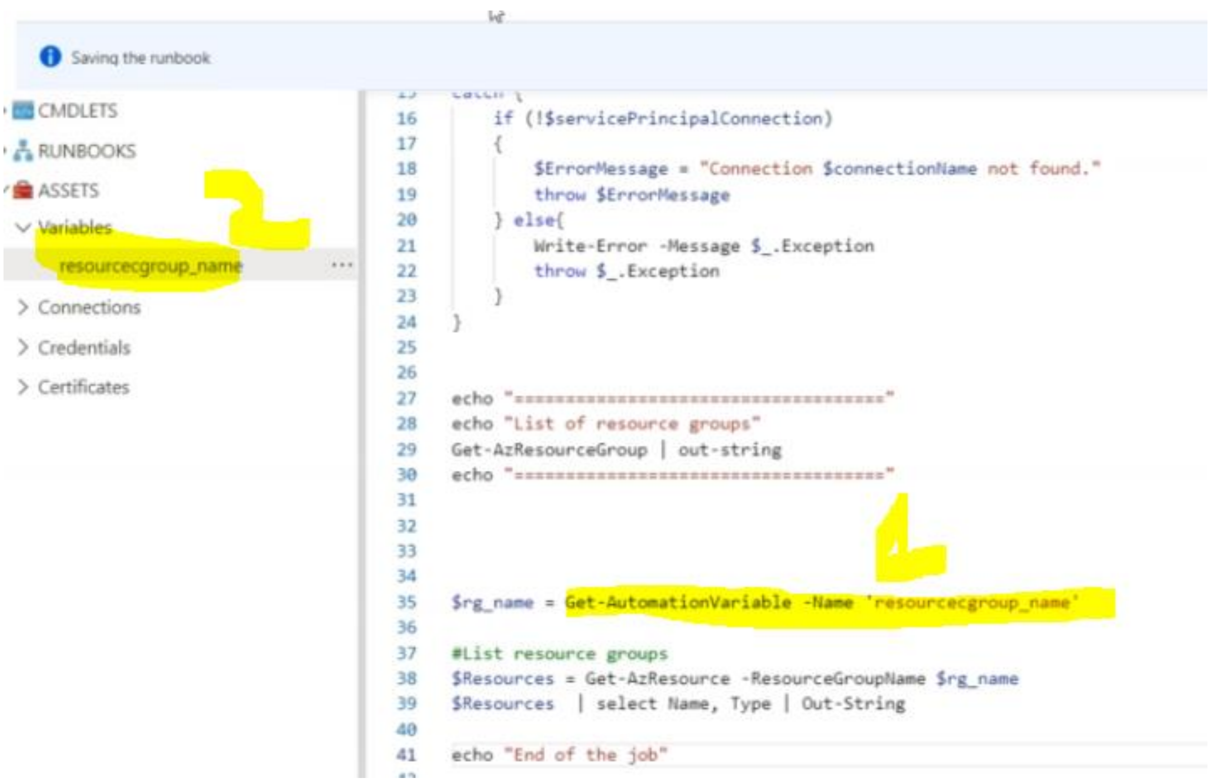
```
#Creating the connection
$connectionName = "AzureRunAsConnection"
try
{
    # Get the connection "AzureRunAsConnection "
    $servicePrincipalConnection=Get-AutomationConnection -Name $connectionName

    "Logging in to Azure..."
    Add-AzAccount `
        -ServicePrincipal `
        -TenantId $servicePrincipalConnection.TenantId `
        -ApplicationId $servicePrincipalConnection.ApplicationId `
        -CertificateThumbprint $servicePrincipalConnection.CertificateThumbprint
}
catch {
    if (!$servicePrincipalConnection)
    {
        $ErrorMessage = "Connection $connectionName not found."
        throw $ErrorMessage
    } else{
        Write-Error -Message $_.Exception
        throw $_.Exception
    }
}



echo "List of resource groups"
Get-AzResourceGroup | out-string
```

- Save the script and test



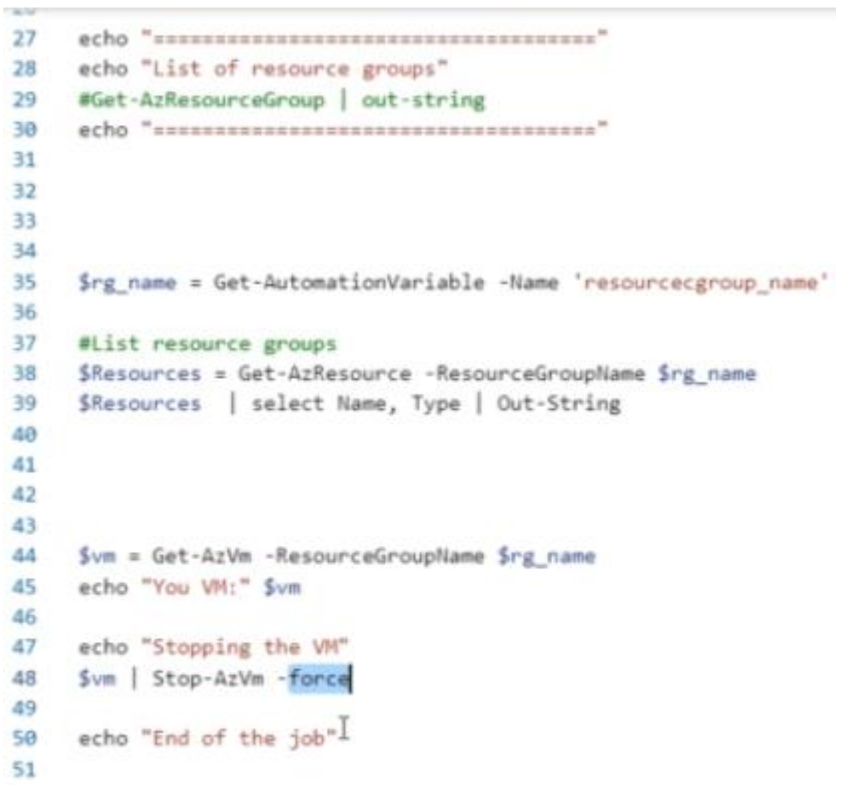
```
    {
        $ErrorMessage = "Connection $connectionName not found."
        throw $ErrorMessage
    } else{
        Write-Error -Message $_.Exception
        throw $_.Exception
    }
}


echo "========================================"
echo "List of resource groups"
Get-AzResourceGroup | out-string
echo "========================================"



#List resource groups
$Resources = Get-AzResource -ResourceGroupName 'DEMO-RG'
$Resources  | select Name, Type | Out-String

echo "End of the job"
```

- Now we need to create variables on automation account that can be used any runbook
- Now in our below script we have used the variables from highlighted colour 2

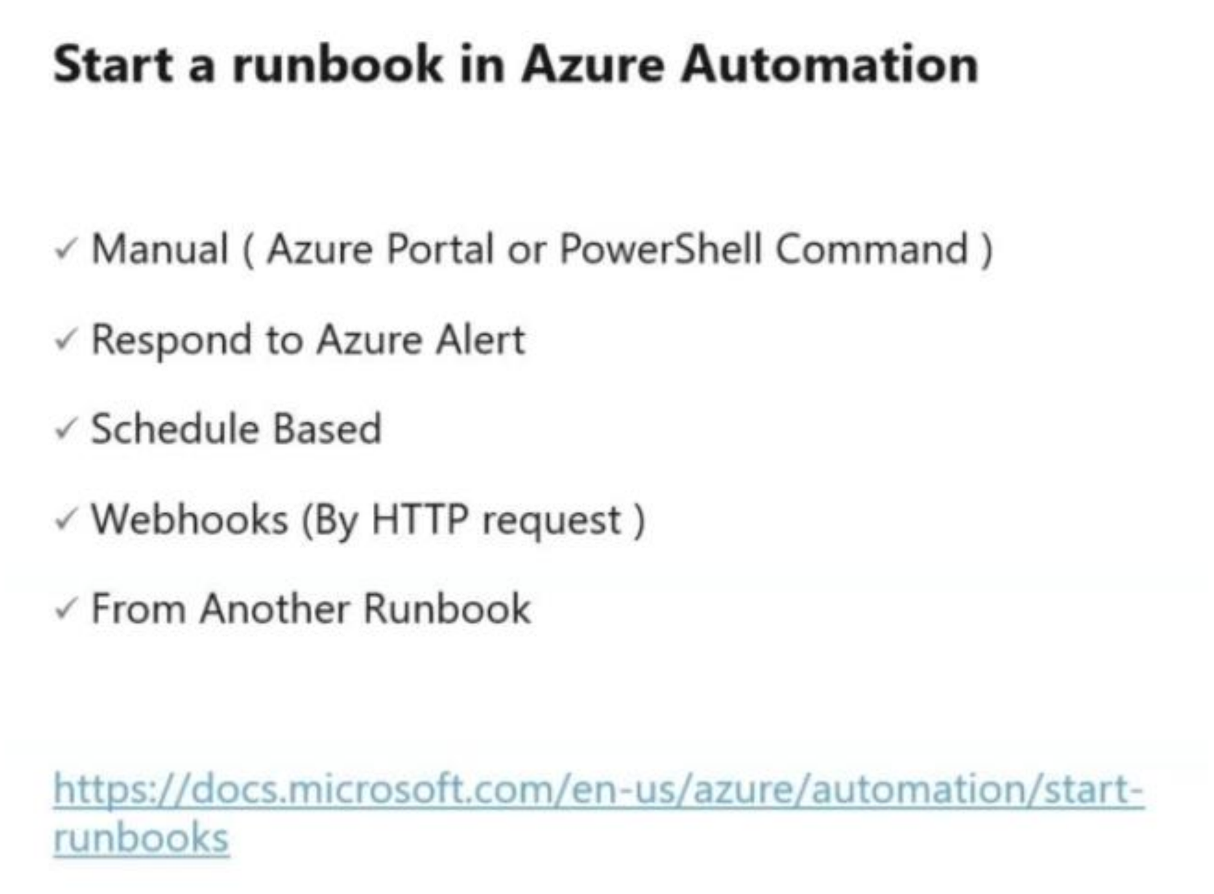


```
    if (!$servicePrincipalConnection)
    {
        $ErrorMessage = "Connection $connectionName not found."
        throw $ErrorMessage
    } else{
        Write-Error -Message $_.Exception
        throw $_.Exception
    }
}


echo "====================================="
echo "List of resource groups"
Get-AzResourceGroup | out-string
echo "====================================="




$rg_name = Get-AutomationVariable -Name 'resourcecgroup_name'

#List resource groups
$Resources = Get-AzResource -ResourceGroupName $rg_name
$Resources  | select Name, Type | Out-String

echo "End of the job"
```

- Before running the below command we need to import “az.compute” module



```
echo "====================================="
echo "List of resource groups"
#Get-AzResourceGroup | out-string
echo "====================================="




$rg_name = Get-AutomationVariable -Name 'resourcecgroup_name'

#List resource groups
$Resources = Get-AzResource -ResourceGroupName $rg_name
$Resources  | select Name, Type | Out-String




$vm = Get-AzVm -ResourceGroupName $rg_name
echo "You VM:" $vm

echo "Stopping the VM"
$vm | Stop-AzVm -force

echo "End of the job"
```



Start a runbook in Azure Automation

✓ Manual (Azure Portal or PowerShell Command)

✓ Respond to Azure Alert

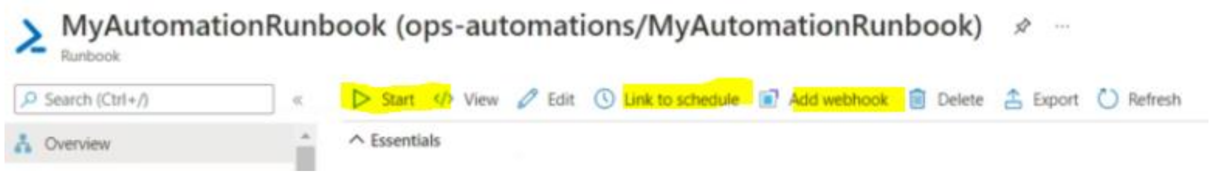
✓ Schedule Based

✓ Webhooks (By HTTP request)

✓ From Another Runbook

https://docs.microsoft.com/en-us/azure/automation/start-runbooks

- Once we write and save the runbook it will not show start, link to schedule and webhook
- To get above options we need to publish our runbook
- As below we can get options



- Create automation run book to scale in vm size by hitting CPU percentage, we need to set the alert and action group as runbook
- The runbook will create scale in the vm