Agenda: Azure Automation

- Introduction
- Azure Automation Account
- Automation Runbook
- PowerShell Runbook
- Adding Parameters to Runbook
- Starting a Runbook with a Webhook
- Scheduling a Runbook
- Graphical Runbook
- Error Handling in Graphical Runbook
- Runbook Input Parameters
- Azure Automation DSC

Azure Automation and Runbook Introduction

- Microsoft Azure Automation provides a way for users to automate the manual, long-running, error-prone, and frequently repeated tasks that are commonly performed in a cloud and across external systems.
- Azure uses a highly scalable and reliable workflow execution engine to simplify cloud management.
- It saves time and increases the reliability of regular administrative tasks and even schedules them to be automatically performed at regular intervals.
- **Runbook** is a **set of tasks** that perform some automated process in Azure Automation. It may be a simple process such as starting a virtual machine and creating a log entry, or you may have a complex runbook that combines other smaller runbooks to perform a complex process across multiple resources or even multiple clouds and on-premises environments.

For example, you might have an existing manual process for truncating a SQL database if it's approaching maximum size that includes multiple steps such as

a. connecting to the server
b. connecting to the database
c. get the current size of database
d. check if threshold has exceeded and then truncate it and notify user.

Instead of manually performing each of these steps, you could create a runbook that would perform all of these tasks as a single process. You would start the runbook, provide the required information such as the SQL server name, database name, and recipient e-mail and then sit back while the process completes.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Create a standalone Azure Automation Account

With the Automation account, you are able to authenticate runbooks managing resources.

Create a new Automation Account from the Azure portal

1. Azure Portal → +New → **Automation**
2. Name = MyAutomationDemos, . . ., Create Azure **Run As account** = Yes
3. Create

The **Run As account** is the preferred authentication method, because it uses certificate authentication instead of a password that might expire or change frequently.

When you create an Automation account in the Azure portal, you automatically create two authentication entities:

- **A Run As account.** This account creates a **service principal** in Azure Active Directory (Azure AD) and a certificate. It also assigns the **Contributor** role-based access control (RBAC), which manages Resource Manager resources by using runbooks.
- **A Classic Run As account**. This account uploads a management certificate, which is used to manage classic resources by using runbooks.

When the Automation account is successfully created, several resources are automatically created for you:

Resource	Description
AzureAutomationTutorial Runbook	An example **Graphical** runbook that demonstrates how to authenticate using the **Run As account** and gets all the Resource Manager resources.
AzureAutomationTutorialScript Runbook	An example **PowerShell** runbook that demonstrates how to authenticate using the Run As account and gets all the Resource Manager resources.
AzureAutomationTutorialPython2 Runbook	An example **python** runbook that demonstrates how to authenticate using the Run As account and then lists the resource groups present in the specified subscription.
AzureRunAsCertificate	Certificate asset automatically created during Automation account creation or using the PowerShell script below for an existing account. It allows you to authenticate with Azure so that you can manage Azure Resource Manager resources from runbooks. This certificate has a one-year lifespan.
AzureRunAsConnection	Connection asset automatically created during Automation account creation or using the PowerShell script below for an existing account.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

To create a new Azure Automation runbook

You can create your own runbooks from scratch or modify runbooks from the Runbook Gallery for your own requirements.

Types of runbook

Type	Description
PowerShell	Text runbook based on Windows PowerShell script.
PowerShell Workflow	Text runbook based on Windows PowerShell Workflow.
Graphical	Based on Windows PowerShell and created and edited completely in graphical editor in Azure portal.
Graphical PowerShell Workflow	Based on Windows PowerShell Workflow and created and edited completely in the graphical editor in Azure portal.
Python	Text runbook based on Python.

To create a new Azure Automation runbook with Azure Portal

1. Azure Portal → Automation Account → Select Account → Runbooks
2. Add a runbook → Create a new runbook
3. Name=TestAuthenticationRunbook, Type="**PowerShell**" → Create

To create a new Azure Automation runbook with Windows PowerShell:

```
New-AzureRmAutomationRunbook -AutomationAccountName MyAccount -Name NewRunbook -ResourceGroupName
MyResourceGroup -Type PowerShell
```

Edit Textual PowerShell Runbook

4. Select the New Runbook → Edit → Enter the following Script

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

   "Logging in to Azure..."
   Add-AzureRmAccount `
      -ServicePrincipal `
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
      -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
  }
}

#Get all ARM resources from all resource groups
$ResourceGroups = Get-AzureRmResourceGroup

foreach ($ResourceGroup in $ResourceGroups)
{
  Write-Output ("Showing resources in resource group " + $ResourceGroup.ResourceGroupName)
  $Resources = Find-AzureRmResource -ResourceGroupNameContains $ResourceGroup.ResourceGroupName | Select
ResourceName, ResourceType
  foreach ($Resource in $Resources)
  {
    Write-Output ($Resource.ResourceName + " of type " +  $Resource.ResourceType)
  }
  Write-Output ("")
}
```

5. Click on **Test Pane** and Start the Runbook.
6. You can now **Publish** the Runbook.
7. If required, you can now create a **Schedule** for the runbook (Once or Recurring)

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Adding Parameters to the Runbook

8. Add the following PowerShell on the top

```
param
(
  [parameter(Mandatory=$false)]
  [string] $ResourceGroupName
)
```

9. Update the script

10. To get all ARM resources from a **given** resource group otherwise **all** resource groups

```
if (!$ResourceGroupName)
{  $ResourceGroups = Get-AzureRmResourceGroup  }
else
{  $ResourceGroups = Get-AzureRmResourceGroup -Name $ResourceGroupName }
```

Automation Variables

Automation variables are useful for the following scenarios:

- Share a value between multiple runbooks or DSC configurations.
- Share a value between multiple jobs from the same runbook or DSC configuration.
- Manage a value from the portal or from the PowerShell command line that is used by runbooks or DSC configurations, such as a set of common configuration items like specific list of VM names, a specific resource group, an AD domain name, and more.

To create a new variable with the Azure portal

1. From your Automation account, click the **Assets** tile and then on the **Assets** blade, select **Variables**.
2. On the **Variables** tile, select **Add a variable**.
3. Complete the options on the **New Variable** blade and click **Create** save the new variable.

To create a new automation variable with Windows PowerShell

```
New-AzureRmAutomationVariable -ResourceGroupName "DemoRG" –AutomationAccountName "MyAutomationAccount"
–Name 'MyStringVariable' –Encrypted $false –Value 'My String'
```

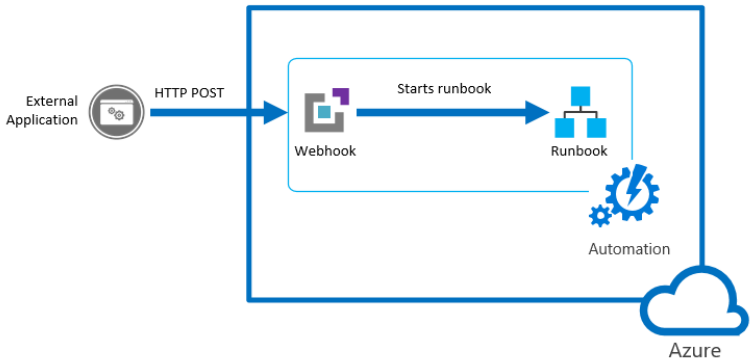
To get the value of Automation Variable:

```
$string = (Get-AzureRmAutomationVariable -ResourceGroupName "DemoRG" –AutomationAccountName
"MyAutomationAccount" –Name 'MyStringVariable').Value
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Starting a Runbook with a webhook

A *webhook* allows you to start a particular runbook in Azure Automation through a single HTTP request. This allows external services such as Azure DevOps Services, GitHub, Azure Monitor logs, or custom applications to start runbooks without implementing a full solution using the Azure Automation API.



Details of a Webhook

- Name
- URL
- Expiration Date
- Enabled

To receive data from the client, the runbook can accept a single parameter called **$WebhookData**. The **$WebhookData** object has the following properties:

Property	Description
WebhookName	The name of the webhook.
RequestHeader	Hash table containing the headers of the incoming POST request.
RequestBody	The body of the incoming POST request. This retains any formatting such as string, JSON, XML, or form encoded data.

Step1: Create a Sample Runbook

```
param
```

```
(
    [Parameter (Mandatory = $false)]
    [object] $WebhookData
)

# If runbook was called from Webhook, WebhookData will not be null.
if ($WebhookData) {
    # Check header for message to validate request
    if ($WebhookData.RequestHeader.message -eq 'StartedbySandeep')
    {
        Write-Output "Header has required information"
    }
    else
    {
        Write-Output "Header missing required information";
        exit;
    }

    # Retrieve VM's from Webhook request body
    $vms = (ConvertFrom-Json -InputObject $WebhookData.RequestBody)

    # Authenticate to Azure by using the service principal and certificate. Then, set the subscription.
    Write-Output "Authenticating to Azure with service principal and certificate"
    $ConnectionAssetName = "AzureRunAsConnection"
    $Conn = Get-AutomationConnection -Name $ConnectionAssetName
    if ($Conn -eq $null)
    {
            throw "Could not retrieve connection asset: $ConnectionAssetName. Check that this asset exists in the
Automation account."
    }
    Write-Output "Authenticating to Azure with service principal."
    Add-AzureRmAccount -ServicePrincipal -Tenant $Conn.TenantID -ApplicationId $Conn.ApplicationID -
CertificateThumbprint $Conn.CertificateThumbprint
```

```
  # Start each virtual machine
  foreach ($vm in $vms)
  {
        $vmName = $vm.Name
        $ResourceGroup = $vm.ResourceGroup
        Write-Output "Starting $vmName"
        Start-AzureRMVM -Name $vm.Name -ResourceGroup $ResourceGroup
  }
}
else {
  # Error
  write-Error "This runbook is meant to be started from an Azure alert webhook only."
}
```

Step2: Create a Webhook

1. From the **Runbooks page** in the Azure portal, click the runbook that the webhook starts to view its detail page. Ensure the runbook **Status** is **Published**.
2. Click **Webhook** at the top of the page to open the **Add Webhook** page.

Note: We should **note down the URL** securely as it serves the purpose of a password. Any user having this URL can issue an HTTP POST request and run the runbook without any further authentication due to the presence of the security token in the URL. **Moreover, we can see this URL only during the webhook creation time**

Step3: Testing the Sample using PowerShell

```
$uri = "<webHook Uri>"
$vms  = @(
        @{ Name="vm01";ResourceGroup="vm01"},
        @{ Name="vm02";ResourceGroup="vm02"}
    )
$body = ConvertTo-Json -InputObject $vms
$header = @{ message="StartedbySandeep"}
$response = Invoke-WebRequest -Method Post -Uri $uri -Body $body -Headers $header
$response
$jobid = (ConvertFrom-Json ($response.Content)).jobids[0]
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
$jobid
```

Sample JSON Input for Request Body

```
[
  {
    "Name": "vm01",
    "ResourceGroup": "myResourceGroup"
  },
  {
    "Name": "vm02",
    "ResourceGroup": "myResourceGroup"
  }
]
```

Invoking from C#

```
string uri = "https://s16events.azure-
automation.net/webhooks?token=Ut5T7M%2bZdFTQV5uSx9YTwQz4eqVQmDM7ghbwlRbQHKI%3d";

HttpWebRequest request = (HttpWebRequest)HttpWebRequest.Create(uri);
string data = "[{\"Name\": \"vm01\",\"ResourceGroup\": \"myResourceGroup\"}]";
request.Method = "POST";
request.ContentType = "text/plain;charset=utf-8";

System.Text.UTF8Encoding encoding = new System.Text.UTF8Encoding();
byte[] bytes = encoding.GetBytes(data);

request.ContentLength = bytes.Length;

WebHeaderCollection col = new WebHeaderCollection();
col["message"] = "StartedbySandeep";
request.Headers = col;

using (Stream requestStream = request.GetRequestStream())
```

```
{
   requestStream.Write(bytes, 0, bytes.Length);
}
request.BeginGetResponse((x) =>
{
   using (HttpWebResponse response = (HttpWebResponse)request.EndGetResponse(x))
   {
      using (Stream stream = response.GetResponseStream())
      {
         StreamReader reader = new StreamReader(stream, Encoding.UTF8);
         String responseString = reader.ReadToEnd();
         Console.WriteLine("Webhook Triggered at " + System.DateTime.Now + " \n Job Details : " + responseString);
      }
   }
}, null);
```

Scheduling a Runbook

To schedule a runbook in Azure Automation to start at a specified time, you link it to one or more schedules. A schedule can be configured to either run once or on a reoccurring hourly or daily schedule for runbooks in the Azure portal. You can also schedule them for weekly, monthly, specific days of the week or days of the month, or a particular day of the month. A runbook can be linked to multiple schedules, and a schedule can have multiple runbooks linked to it.

To create a new schedule in the Azure portal:

1. In the Azure portal, from your automation account, select **Schedules** under the section **Shared Resources** on the left.
2. Click **Add a schedule** at the top of the page.
3. On the **New schedule** pane, type a **Name** and optionally a **Description** for the new schedule.
4. Select whether the schedule runs one time, or on a reoccurring schedule by selecting **Once** or **Recurring**. If you select **Once** specify a **Start time**, and then click **Create**. If you select **Recurring**, specify a **Start time** and for **Recur every**, select the frequency for how often you want the runbook to repeat - by **hour**, **day**, **week**, or by **month**.

Linking a schedule to a runbook:

1. In the Azure portal → Automation account → select **Runbooks**.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

2. If the runbook isn't currently linked to a schedule, then you're offered the option to create a new schedule or link to an existing schedule.
3. If the runbook has parameters, you can select the option **Modify run settings (Default:Azure)** and the **Parameters** pane is presented where you can enter the information.

Scheduling runbooks more frequently

The most frequent interval a schedule in Azure Automation can be configured for is one hour. If you require schedules to execute more frequently than that, there are two options:

1. Create a webhook for the runbook and use **Azure Scheduler** to call the webhook. Azure Scheduler provides more fine-grained granularity when defining a schedule.
2. Create four schedules all starting within 15 minutes of each other running once every hour. This scenario allows the runbook to run every 15 minutes with the different schedules.

Graphical Runbook

Graphical Authoring allows you to create runbooks for Azure Automation without the complexities of the underlying Windows PowerShell or PowerShell Workflow code. You add activities to the canvas from a library of cmdlets and runbooks, link them together and configure to form a workflow.

Get-AutomationConnection	AzureRunAsConnection (Constant Value) = **AzureRunAsConnection**
Add-AzureRmAccount	**Parameter sets** = **ServicePrincipalCertificate** **APPLICAITONID**: **Data source**=Activity output, **Select data**=Get-AutomationConnection, **Field path**=ApplicationId **CERTIFICATETHUMBPRINT**: Data source=Activity output, Select data=Get-AutomationConnection, Field path= CertificateThumbprint **SERVICEPRINCIPAL**: Constant value=true **TENANTID:** Data source=Activity output, Select data=Get-AutomationConnection, Field path= TenantId
Get-AzureRmResourceGroup	-

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Find-AzureRmResource	**Parameter sets** = List the resource based on specified scope. **RESOURCEGROUPNAMECONTAINS**: Data source=Activity output, Select data=Get-AzureRmResourceGroup, Field path=ResourceGroupName **RESOURCETYPE**: Constant value = microsoft.web/sites
Remove-AzureRmResource	**Parameter set:** The resource Id **Resource Id:** Data source=Activity output, Select data=Find-AzureRmResource, **Field path**=ResourceId

Configure input parameters in graphical runbooks

1. Select the Runbook → Edit → **Input and output**
2. Add input → set values…
3. Add Write-Output Activity
4. Parameters → INPUTOBJECT = Data source=Runbook input, Select Parameter.

Graphical Runbooks Artifacts

- **Links and workflow**
  - **Pipeline**: The destination activity is run once for each object output from the source activity. The destination activity does not run if the source activity results in no output. Output from the source activity is available as an object.
  - **Sequence**: The destination activity runs only once. Output from the source activity is available as an array of objects.
- **Retry activity:** allows an activity to be run multiple times until a particular condition is met, much like a loop.
- **Starting activity:** A graphical runbook starts with any activities that do not have an incoming link. If multiple activities do not have an incoming link, then the runbook starts by running them in parallel.
- **Conditions**: When you specify a condition on a link, the destination activity is only run if the condition resolves to true. You typically use an **$ActivityOutput** variable in a condition to retrieve the output from the source activity

  Eg: **$ActivityOutput**['Get Azure Resource Groups'].ResourceGroupName -match "DemoRG"
- **Junctions:** A junction is a special activity that waits until all incoming branches have completed. This allows you to run multiple activities in parallel and ensure that all have completed before moving on.

Error Handling In Graphical Runbook

The types of PowerShell errors that can occur during execution are terminating or non-terminating.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

- **Terminating error**: A serious error during execution that halts the command (or script execution) completely. Examples include non-existent cmdlets, syntax errors that prevent a cmdlet from running, or other fatal errors.
- **Non-terminating error**: A non-serious error that allows execution to continue despite the failure. Examples include operational errors such as file not found errors and permissions problems.

You can now turn exceptions into non-terminating errors and create error links between activities. This process allows a runbook author to catch errors and manage realized or unexpected conditions.

For each activity that can produce an error, the runbook author can add an error link pointing to any other activity. The destination activity can be of any type, including code activities, invoking a cmdlet, invoking another runbook, and so on. After configuring this setting, you create an activity that handles the error. If an activity produces any error, then the outgoing error links are followed, and the regular links are not, even if the activity produces regular output as well.

Example: Consider a runbook that tries to start a VM and install an application on it. If the VM doesn't start correctly, it performs two actions:

1. It sends a notification about this problem.
2. It starts another runbook that automatically provisions a new VM instead.

The **Get-AutomationVariable** activity can be configured to convert exceptions to errors. If there are problems getting the variable, then errors is generated.

Azure PowerShell Worflow Runbook

PowerShell Workflow Runbooks in Azure Automation are implemented as **Windows PowerShell Workflows**. A Windows PowerShell Workflow is a Windows PowerShell script that uses **Windows Workflow Foundation**.

Differences between PowerShell and PowerShell Workflow

PowerShell runbooks have the same lifecycle, capabilities, and management as PowerShell Workflow runbooks but there are some differences and limitations:

1. PowerShell runbooks run **fast** compared to PowerShell Workflow runbooks as they don't have **compilation** step.
2. PowerShell Workflow runbooks support **checkpoints**, using checkpoints, PowerShell Workflow runbooks can **resume** from any point in the runbook. PowerShell runbooks can only resume from the beginning.
3. PowerShell Workflow runbooks support **parallel** and serial execution. PowerShell runbooks can only execute commands serially.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

4. There are also some **syntactic differences** between a native PowerShell runbook and a PowerShell Workflow runbook.

Learning key Windows PowerShell Workflow concepts for Automation runbooks

https://docs.microsoft.com/en-us/azure/automation/automation-powershell-workflow

Automation to Start or Stop Azure VM's:

1. Goto Automation Account → Settings → Credentails → Add a Credential, Name = **AzureCredential**
2. Goto Automation Account → Settings → Runbooks → **Create a runbook**,
   a. Name= "Stop-Start-AzureVM-PowershellWorkflow"
   b. Runbook type = PowerShell Workflow
3. Edit the Runbook as below

```
Workflow Stop-Start-AzureVM-PowershellWorkflow
{
  Param
  (
    [Parameter(Mandatory=$true)][ValidateNotNullOrEmpty()]
    [String]
    $AzureSubscriptionId,
    [Parameter(Mandatory=$true)][ValidateNotNullOrEmpty()]
    [String]
    $AzureVMList="All",
    [Parameter(Mandatory=$true)][ValidateSet("Start","Stop")]
    [String]
    $Action
  )

  $credential = Get-AutomationPSCredential -Name 'AzureCredential'
  Login-AzureRmAccount -Credential $credential
  Select-AzureRmSubscription -SubscriptionId $AzureSubscriptionId

  if($AzureVMList -ne "All")
  {
    $AzureVMs = $AzureVMList.Split(",")
```

```
    [System.Collections.ArrayList]$AzureVMsToHandle = $AzureVMs
  }
  else
  {
    $AzureVMs = (Get-AzureRmVM).Name
    [System.Collections.ArrayList]$AzureVMsToHandle = $AzureVMs
   }

  foreach($AzureVM in $AzureVMsToHandle)
  {
    if(!(Get-AzureRmVM | ? {$_.Name -eq $AzureVM}))
    {
      throw " AzureVM : [$AzureVM] - Does not exist! - Check your inputs "
    }
  }

  if($Action -eq "Stop")
  {
    Write-Output "Stopping VMs";
    foreach -parallel ($AzureVM in $AzureVMsToHandle)
    {
      Get-AzureRmVM | ? {$_.Name -eq $AzureVM} | Stop-AzureRmVM -Force
    }
  }
  else
  {
    Write-Output "Starting VMs";
    foreach -parallel ($AzureVM in $AzureVMsToHandle)
    {
      Get-AzureRmVM | ? {$_.Name -eq $AzureVM} | Start-AzureRmVM
    }
  }
}
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

CheckPoint or Suspend-Worflow

A *checkpoint* is a snapshot of the current state of the workflow that includes the current value for variables and any output generated to that point. If a workflow ends in error or is suspended, then the next time it is run it will start from its last checkpoint instead of the start of the workflow. You can set a checkpoint in a workflow with the **Checkpoint-Workflow** activity.

You should set checkpoints in a workflow after activities that may be prone to exception and should not be repeated if the workflow is resumed.

Example: The following example copies multiple files to a network location and sets a checkpoint after each file. If the network location is lost, then the workflow ends in error. When it is started again, it will resume at the last checkpoint meaning that only the files that have already been copied are skipped.

```
Workflow Copy-Files
{
  $files = @("C:\LocalPath\File1.txt","C:\LocalPath\File2.txt","C:\LocalPath\File3.txt")

  ForEach ($File in $Files)
  {
    Copy-Item -Path $File -Destination \\NetworkPath
    Write-Output "$File copied."
    Checkpoint-Workflow
  }

  Write-Output "All files copied."
}
```

Example: To Create VM

```
workflow CreateTestVms
{
  $Cred = Get-AzureAutomationCredential -Name "MyCredential"
  $null = Connect-AzureRmAccount -Credential $Cred

  $VmsToCreate = Get-AzureAutomationVariable -Name "VmsToCreate"
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
  foreach ($VmName in $VmsToCreate)
  {
    # Do work first to create the VM (code not shown)

    # Now add the VM
    New-AzureRmVm -VM $Vm -Location "WestUs" -ResourceGroupName "ResourceGroup01"

    # Checkpoint so that VM creation is not repeated if workflow suspends
    $Cred = $null
    Checkpoint-Workflow # OR use Suspend-Worflow
    $Cred = Get-AzureAutomationCredential -Name "MyCredential"
    $null = Connect-AzureRmAccount -Credential $Cred
  }
}
```

Because username credentials are not persisted after you call the Suspend-Workflow activity or after the last checkpoint, you need to set the credentials to null and then retrieve them again from the asset store after **Suspend-Workflow** or **checkpoint** is called.

Simple Example:

```
Workflow Demo
{
  Param
  (
    [Parameter(Mandatory=$true)][ValidateNotNullOrEmpty()]
    [String]
    $NamesList
  )

  $Names = $NamesList.Split(",")
  [System.Collections.ArrayList]$NamesArray = $Names
  $count = 0
  foreach($name in $NamesArray)
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
  {
    if ($name -eq "S")
    {
      $ConnectionAssetName = "AzureRunAsConnection"
      $Conn = Get-AutomationConnection -Name $ConnectionAssetName
      if ($Conn -eq $null)
      {
        throw "Could not retrieve connection asset: $ConnectionAssetName. Check that this asset exists in the
Automation account."
      }
      Add-AzureRmAccount -ServicePrincipal -Tenant $Conn.TenantID -ApplicationId $Conn.ApplicationID -
CertificateThumbprint $Conn.CertificateThumbprint
      $actionValue = (Get-AzureRmAutomationVariable -ResourceGroupName "DemoRG" –AutomationAccountName
"dssdemo" –Name "action").Value
      Write-Output "Action:  $actionValue"
      if ($actionValue -eq "stop")
      {
        throw "Invalid Name"
      }
    }
    Write-Output "Name:  $name"
    Checkpoint-Workflow
  }
}
```

Azure Automation DSC

1. Create a VM in same location as Automation Account.
2. Creating a DSC configuration: **TestConfig.ps1**

```
configuration TestConfig
{
  Node IsWebServer
  {
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
    WindowsFeature IIS
    {
      Ensure          = 'Present'
      Name            = 'Web-Server'
      IncludeAllSubFeature = $true
    }
  }
  Node NotWebServer
  {
    WindowsFeature IIS
    {
      Ensure          = 'Absent'
      Name            = 'Web-Server'
    }
  }
}
```

3. Importing a configuration into Azure Automation
    a. Azure Automation → State Configuration (DSC) → +Add → Import Configuration → TestConfig.ps1 → OK
4. Viewing a configuration in Azure Automation
    a. DSC Configurations → TestConfig → **View Configuration source**
5. Compiling a configuration in Azure Automation
    a. DSC Configurations → TestConfig → **Compile** → Yes
    b. The compiled configurations are now listed in **Compiled Configuration tab**.
6. Onboarding an Azure VM for management with Azure Automation DSC
    a. Automation account blade → State Configuration (DSC) → Select Nodes Tab
    b. Select Azure VM → Select virtual machines to onboard → + Connect
    c. Provide the details in registration page.
7. Viewing the list of DSC nodes
    a. Automation account blade → State Configuration (DSC) → Select Nodes Tab.