



Azure Automation

Microsoft Services



Agenda

- Assets
- Runbooks
- Author Runbooks
- Desired State Configuration
- Monitoring & Troubleshooting

Automation Today

- Automation today is often manual, error-prone, and frequently repeated tasks.
- Some of the challenges with automation today are:
 - Unable to manage enterprise wide automation tasks from a single location.
 - Inconsistent credential usage for automated tasks.
 - Command line only option for scheduled task management on remote servers.
 - Laborious to monitor automated tasks.



What is Azure Automation

- Azure Automation is a service that allows you to automate the creation, deployment, monitoring, and maintenance of resources in your Azure environment and across external systems.
- Uses a highly scalable and reliable workflow execution engine to execute tasks.
- Applications or services that have an Internet Protocol based API or understand PowerShell can use Azure Automation.
- Includes integration into many Azure services such as:
 - Web Sites (management)
 - Cloud Services (management)
 - Virtual Machines (management and WinRM support)
 - Storage (management)
 - SQL Server (management and SQL support)



Benefits of Azure Automation

- Automate time consuming, error prone operational tasks across Azure & 3rd party systems.
- Manage enterprise wide automation tasks from a single location.
- Ensure consistent credential usage for automated tasks.
- Improved monitoring of automated tasks.
- Lower operational costs by reduced management time.
- Integrate with existing systems.



Pricing

- Current pricing posted at: <http://azure.microsoft.com/en-us/pricing/details/automation/>
- Process Automation
 - Billed by the minute according to actual run time of your jobs
 - Free tier gives you 500 minutes of job run time per month
 - Basic tier costs \$0.12 per hour and gives you unlimited job run time
- Desired State Configuration (DSC)
 - Free tier has 5 nodes
 - Basic tier costs \$6 a node per month & number of nodes is unlimited
- SLA 99.9% guarantees that planned jobs start within 30 minutes of their planned start times.





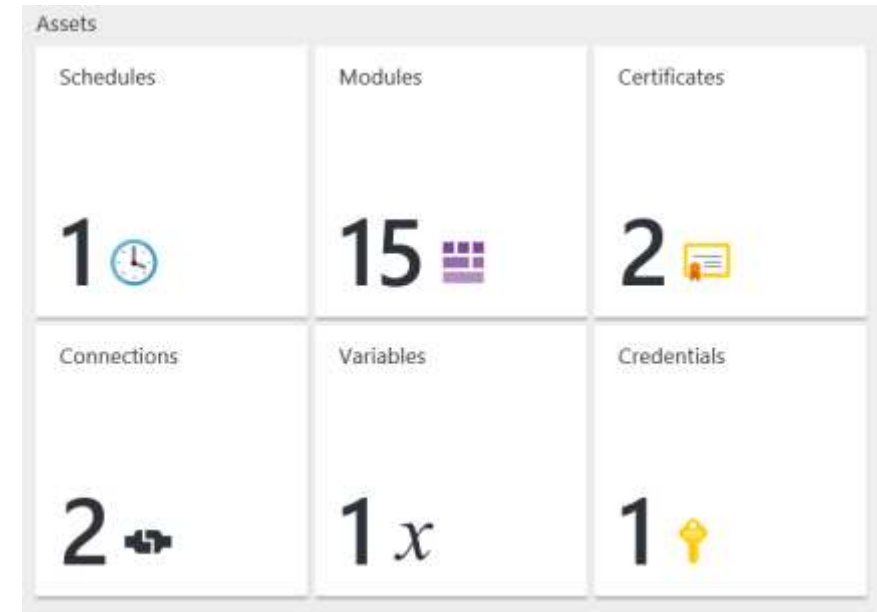
Assets

Microsoft Services



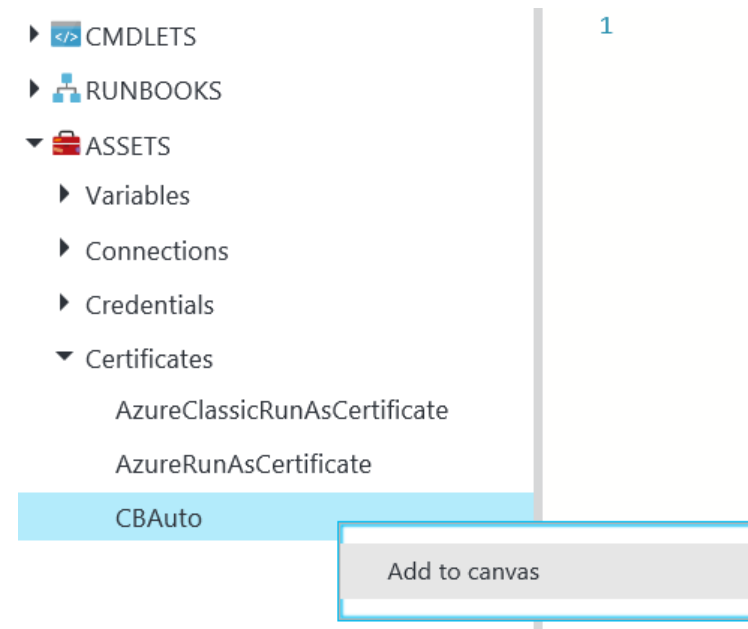
Automation Assets

- Azure Automation Assets are settings that are saved and made globally available to be used in or associated with a Runbook.
- Assets are encrypted and stored in Azure Automation using a unique key that is generated for each Automation account.
- The unique key is encrypted by a master certificate and stored in Azure Automation.
- There are six different types of Automation Assets:
 - Certificates
 - Connections
 - Credentials
 - Integration Modules
 - Schedules
 - Variables



Certificates

- Certificates can be securely stored in Azure Automation so they can be used by Runbooks or DSC configurations to encrypt sensitive information.
- This allows you to create Runbooks and DSC configurations that use certificates for authentication or adds them to Azure or third party resources.
- Import and export both .cer and .pfx files to Azure Automation using PowerShell or the Azure portal.
- Certificate must be marked as exportable during import if it is to be exported at another time.



Connections

- Connections define the information required to connect to a service or application from a Runbook or DSC configuration.
- May include information required for authentication such as a username and password in addition to connection information such as a URL or a port.
- Connection types are defined by the PowerShell integration modules that are installed e.g. installing the Azure PowerShell module allows you to create a connection to an Azure subscription.
- Connections can be created using PowerShell or the Azure portal.

* Name ✓

Description

* Type ⓘ ▼

* AutomationCertificateName ✓

* SubscriptionID ✓

Credentials

- An Automation credential asset holds a PSCredential object which contains security credentials such as a username and password.
- Simplifies Runbook and DSC configurations that may use cmdlets that accept a PSCredential object for authentication.
- Credentials can be created using PowerShell or the Azure portal.

* Name

DomainAdmin



Description

* User name

contoso\administrator



* Password

••••••••



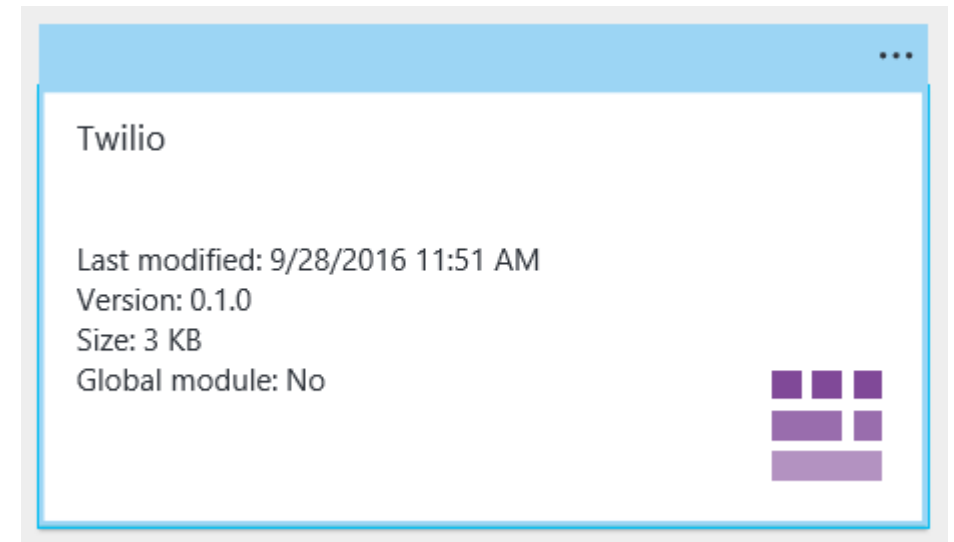
* Confirm password

••••••••



Integration Modules

- An Integration Module is a PowerShell module that is imported into an Azure Automation account.
- Cmdlets from integration modules can be used in Runbooks or DSC configurations.
- Import PowerShell modules from a local machine or from the Automation Integration Modules gallery.
- Imported modules from a local machine must be compressed with a .zip extension and be smaller than 100MB.
- Integration modules are imported via the Azure portal.



Schedules


- Azure Automation Schedules are used to schedule Runbooks to run automatically.
- Could be either a single date and time or it could be a recurring hourly, daily, weekly, or monthly schedule to start the Runbook multiple times.
- Schedules can be created using PowerShell or the Azure portal.
- Schedules do not currently support Azure Automation DSC configurations.

* Name

PM Schedule ✓

Description

* Starts ⓘ

2016-09-29  6:00:00 AM

Denmark - Central European Time ▼

Recurrence

Once Recurring

Variables

- Variable assets are values that are available to all Runbooks and DSC configurations in your automation account.
- Can be created, modified, and retrieved from the Azure portal, PowerShell, and from within a Runbook or DSC configuration.
- Automation variables are useful for:
 - Sharing a value between multiple Runbooks or DSC configurations.
 - Sharing a value between multiple jobs from the same Runbook or DSC configuration.
 - Managing a value from the portal or from a PowerShell command line that is used by Runbooks or DSC configurations.
- Variables can be stored encrypted.

* Name
Location ✓

Description

Type ⓘ
String ▼

* Value
westeurope ✓

* Encrypted

Demo: Automation Assets





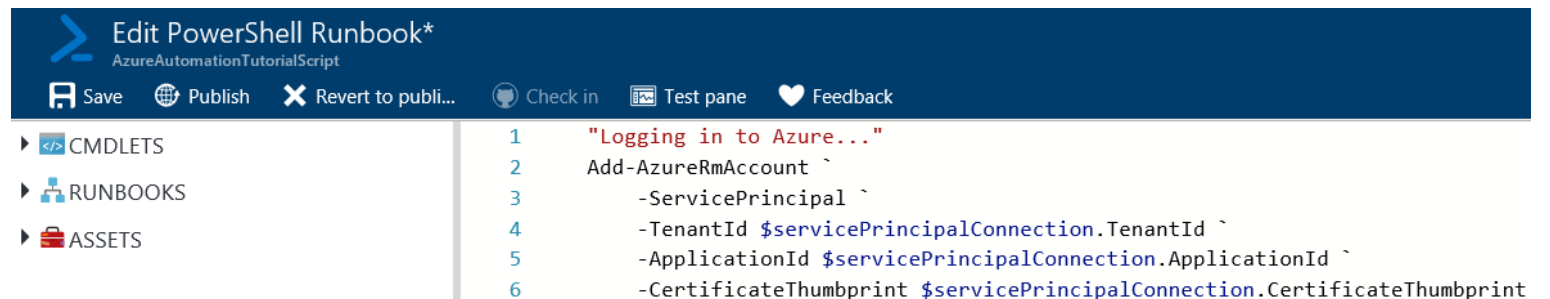
Runbooks

Microsoft Services



What is a Runbook

- In Azure Automation, a Runbook is a file that contains a set of procedures and operations.
- Runbooks are used as input for the Azure Automation service to process.
- The contents of a Runbook file is written using PowerShell or PowerShell Workflow commands.
- PowerShell Workflow is a PowerShell extension that allows you to run a PowerShell script on multiple devices in parallel with added functionality such as checkpoints, suspend & restart.

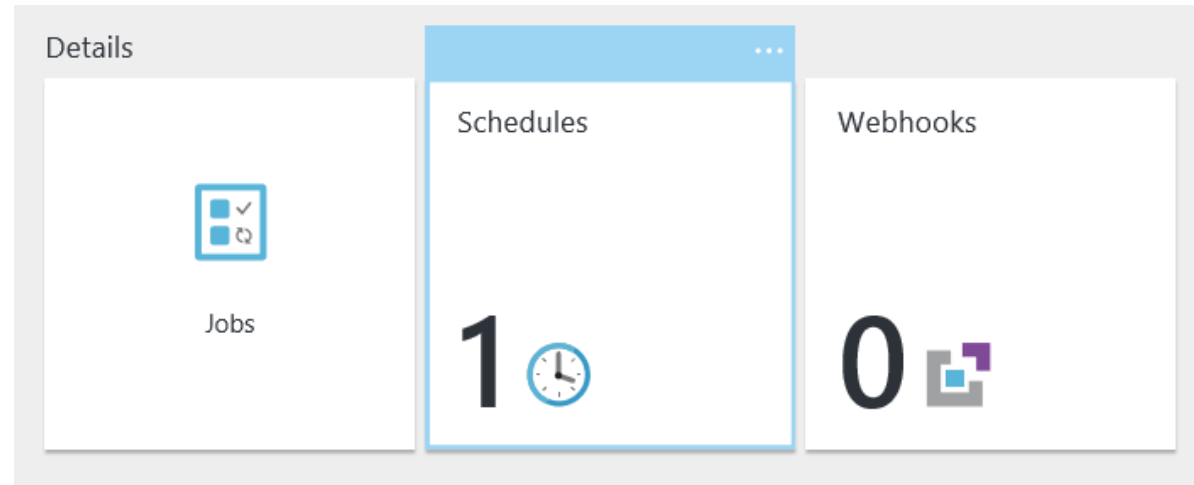


The screenshot displays the 'Edit PowerShell Runbook*' interface in the Azure Automation console. The title bar includes the Azure logo and the text 'Edit PowerShell Runbook*' and 'AzureAutomationTutorialScript'. Below the title bar, there is a toolbar with icons for 'Save', 'Publish', 'Revert to publi...', 'Check in', 'Test pane', and 'Feedback'. The main area is divided into a left sidebar and a right pane. The sidebar contains three expandable sections: 'CMDLETS' (with a PowerShell icon), 'RUNBOOKS' (with a document icon), and 'ASSETS' (with a folder icon). The right pane shows a PowerShell script with line numbers 1 through 6. The script starts with a comment 'Logging in to Azure...' and then uses the 'Add-AzureRmAccount' cmdlet with several parameters: '-ServicePrincipal', '-TenantId', '-ApplicationId', and '-CertificateThumbprint', all using variable references.

```
1 "Logging in to Azure..."
2 Add-AzureRmAccount `
3     -ServicePrincipal `
4     -TenantId $servicePrincipalConnection.TenantId `
5     -ApplicationId $servicePrincipalConnection.ApplicationId `
6     -CertificateThumbprint $servicePrincipalConnection.CertificateThumbprint
```

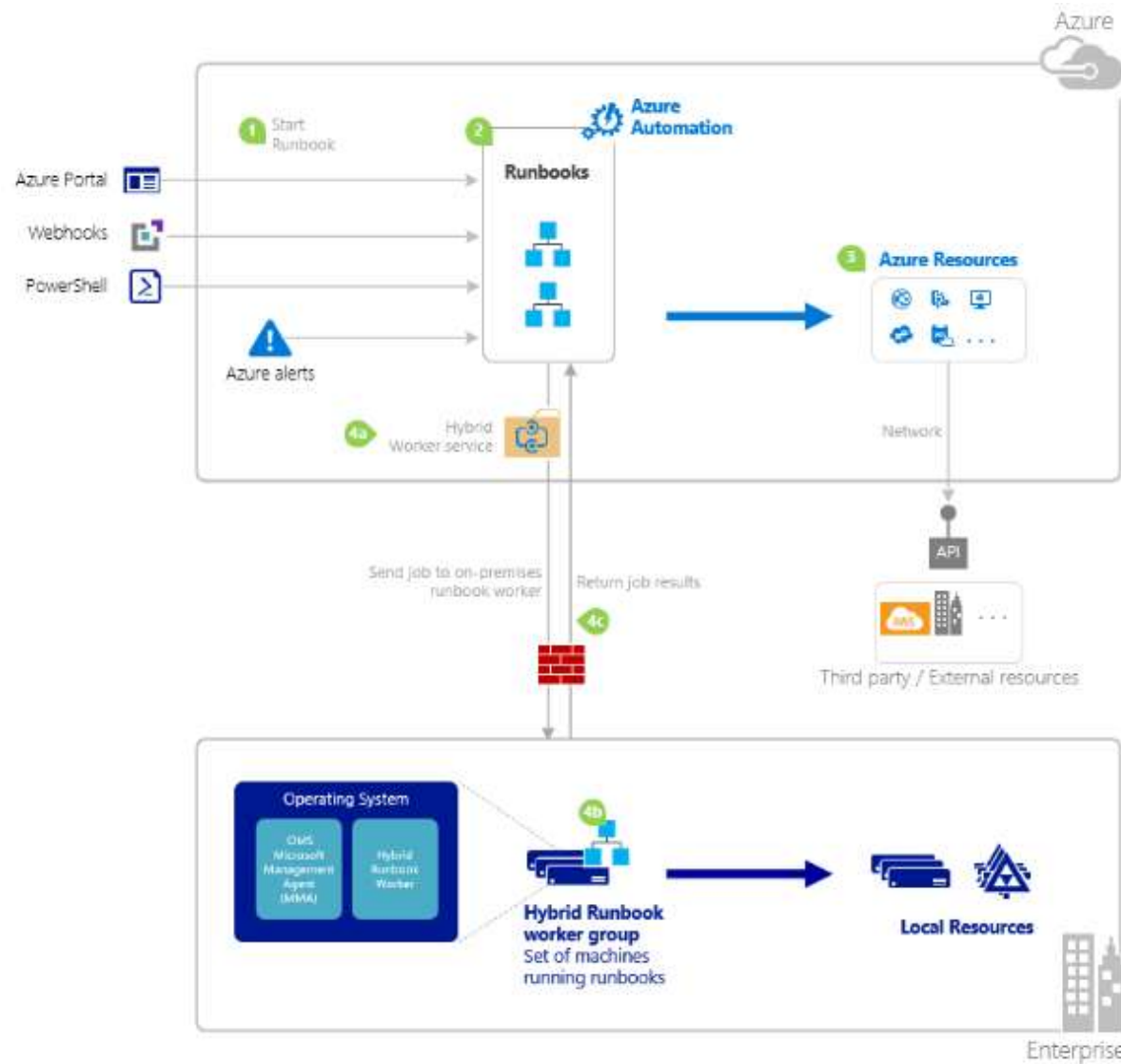
Start a Runbook

- There are 7 different ways in which you can start a Runbook.
 - Azure Portal
 - PowerShell
 - Azure Automation API
 - Webhooks
 - Respond to Azure Alert
 - Schedule
 - From Another Runbook



- The most common methods in use today are Schedule, PowerShell, the Azure Portal and Webhooks.

Runbook Processing



1. An Actor starts a runbook
2. Azure Automation notes that the runbook should be started
3. Cloud resources – Runbook acts on local Azure resources or other external resources reachable via the network
- 4a. On-Premises – Hybrid runbook group sends the runbook to an on-premises machine to run
- 4b. Runbook acts on its local networked resources
- 4c. Job results are returned from on-premises

Runbook Authentication

- Runbooks are authenticated using a Run As account, this allows the Runbook to execute its tasks under this account's security context.
- A Run As account can be created during the creation of an Azure Automation account or an existing account can be added later using PowerShell.
- The Run As account that is created during the creation of an Azure Automation account is granted the Contributor Role for the Azure subscription.
- Existing accounts that are used must be granted the appropriate permissions in order for the Runbook to complete its tasks.

* Create Azure Run As account ⓘ

Yes

No



The Run As account feature will create a Run As account and a Classic Run As account. [Click here to learn more about Run As accounts.](#)

Start a Runbook using a Schedule

- Automatically start a Runbook on an hourly, daily, or weekly schedule.
- Manipulate the schedule through the Azure portal, PowerShell cmdlets, or Azure API.
- Provide parameter values to be used with the schedule.

* Name
PM Schedule ✓

Description
This is the afternoon schedule. ✓

* Starts ⓘ
2016-09-16 📅 12:00:00 PM
Denmark - Central European Time ▼

Recurrence
Once Recurring

* Recur every
1 Day ▼

Set expiration
Yes No

Expires
Never

Create

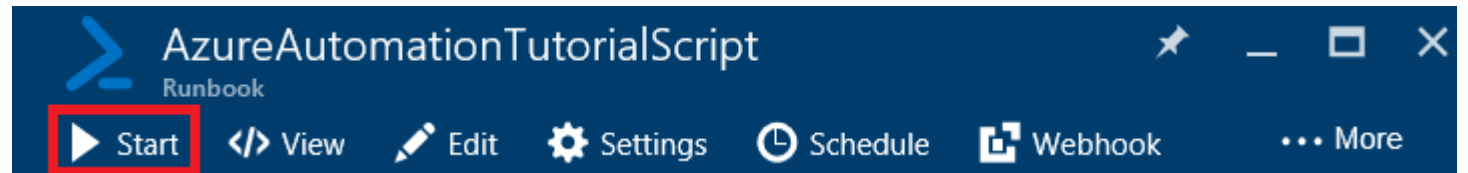
Start a Runbook using PowerShell

- Runbooks can be called from a command line with Windows PowerShell cmdlets.
- The call can be included in an automated solution with multiple steps.
- The request is authenticated with a certificate or OAuth user principal / service principal.
- Provide simple and complex parameter values to start the Runbook.
- Track job state.
- Client required to support PowerShell cmdlets.

```
PS C:\> Start-AzureRmAutomationRunbook -Name AzureAutomationTutorialScript -ResourceGroupName cbauto -AutomationAccountName cbauto
```

Start a Runbook using the Azure Portal

- Is the simplest method with an interactive user interface.
- Forms based to provide simple parameter values.
- Easily track job state.
- Access authenticated with Azure logon.



Start a Runbook using Webhooks

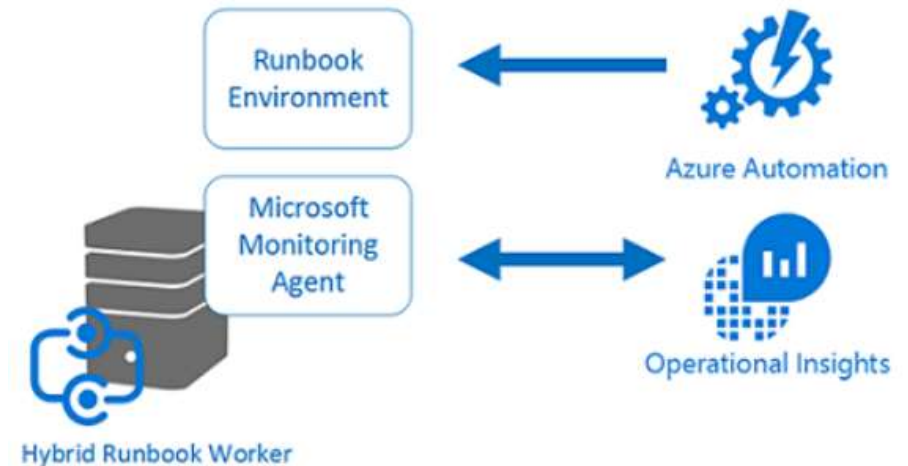
- A Webhook is a HTTP POST request that is sent to a specific URL, on receiving the POST request, some action is taken i.e. start a Runbook.
- The URL is generated during the creation of a Webhook and includes a built in security token that is used to authenticate the request when it is received by the Azure Automation service.
- The Webhook URL must be specified in the application that will be making the request.
- You cannot specify a custom URL and the URL expiration date cannot be changed after the Webhook as been created.
- No ability to track job state through a Webhook URL.

A Webhook URL

```
https://s2events.azure-automation.net/webhooks?token=HiQsFa4HqNbrmAhbD4jxwvGbcwVkOwI5E%2bsIQ3rKmdk%3d
```

Hybrid Runbook Workers

- Hybrid Runbook Workers allow you to run Runbooks on machines located in your local data center in order to manage local resources.
- Runbooks are stored and managed in Azure Automation and downloaded by one or more designated on-premises machines via an agent.
- Outbound TCP 443 is required since the agent on the local computer initiates all communication with Azure Automation.
- Whitelist URL: *.azure-automation.net
- Use the RunOn option in the Azure portal to select the name of the Hybrid Runbook Worker to start a Runbook.



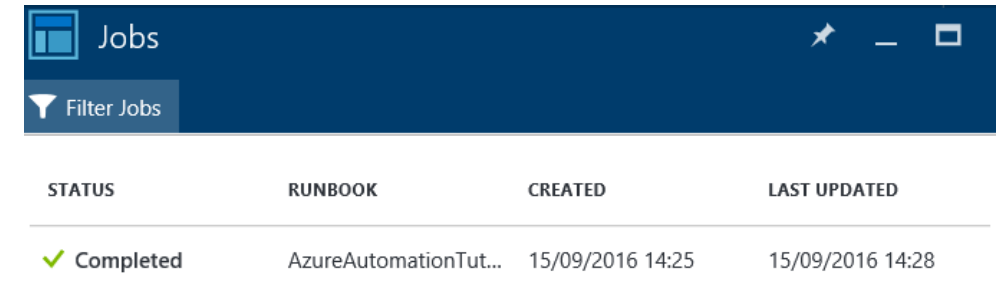
Schedule a Runbook

- To schedule a Runbook in Azure Automation, it must be linked to a pre-existing schedule.
- A schedule can be configured to either run once or on a reoccurring hourly or daily basis.
- A schedule can also be configured to run 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.

Schedules		
AzureAutomationTutorialScript		
Add a schedule Refresh		
NAME	NEXT RUN	STATUS
AM Schedule	Expired	✓ On

Runbook Job Status

- Runbook job status can be monitored using the Jobs tile in the Azure Portal or using `Get-AzureRmAutomationJob` in PowerShell.
- Runbook jobs running for more than 3 hours will be temporarily unloaded and resumed from their last checkpoint, this is known as the Fair Share Limit.
- PowerShell Runbooks will be started from the beginning since they don't support checkpoints.
- Runbooks are terminated with a status of 'Failed, waiting for resources' if they are restarted 3 consecutive times.

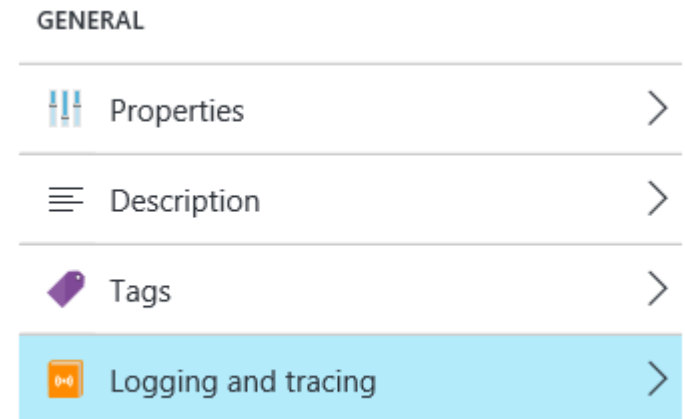


The screenshot shows the 'Jobs' tile in the Azure Portal. It has a dark blue header with the title 'Jobs' and a 'Filter Jobs' button. Below the header is a table with four columns: STATUS, RUNBOOK, CREATED, and LAST UPDATED. The table contains one row with the status 'Completed' (indicated by a green checkmark), the runbook name 'AzureAutomationTut...', and timestamps for creation and last update.

Jobs			
Filter Jobs			
STATUS	RUNBOOK	CREATED	LAST UPDATED
✓ Completed	AzureAutomationTut...	15/09/2016 14:25	15/09/2016 14:28

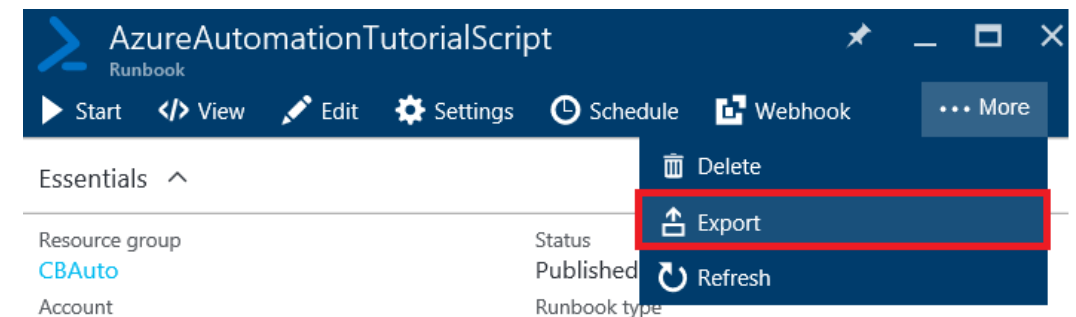
Runbook Settings

- A Runbook in Azure Automation has multiple settings that help it to be identified and to change its logging behavior.
 - **Name & Description** - You cannot change the name of a Runbook after it has been created. The Description is optional and can be up to 512 characters.
 - **Tags** - Tags allow you to assign distinct words and phrases to help identify a runbook.
 - **Logging** - By default, Verbose and Progress records are not written to job history. You can change the settings for a particular runbook to log these records.



Managing Azure Automation Data

- When a resource is deleted in Azure Automation, it is retained for 90 days for auditing purposes before being permanently removed.
- When you delete an automation account in Microsoft Azure, all objects in the account are deleted including runbooks, modules, configurations, settings, jobs, and assets.
- Azure Automation data should be manually backed up e.g. export Runbooks, DSC configuration etc.
- Geo-Replication in Azure Automation is used by default to ensure data recovery in the event of an Azure datacenter outage.



Demo: Creating & Running a Runbook





Lab: Automation with WebHooks

Microsoft Services





Author Runbooks

Microsoft Services



Runbook Types

- There are 4 different types of Runbooks in Azure Automation:
 - **Graphical** - Based on PowerShell but created and edited completely in a graphical editor in the Azure portal.
 - **Graphical PowerShell Workflow** - Based on PowerShell Workflow but created and edited completely in a graphical editor in the Azure portal.
 - **PowerShell** - Based on a PowerShell script.
 - **PowerShell Workflow** - Based on PowerShell Workflow.

* Name ⓘ

StartVM ✓

* Runbook type ⓘ

▼

PowerShell

Graphical

Other

PowerShell Workflow

Graphical PowerShell Workflow

Graphical Runbooks

- Graphical Runbooks are created and edited with the graphical editor only in the Azure portal.
- Can be exported to a file and then imported into another automation account.
- Generate PowerShell code which cannot be directly viewed or modified
- Cannot be converted to one of the text formats, nor can a text Runbook be converted to graphical format.
- Can be converted to Graphical PowerShell Workflow Runbooks during import and vice-versa.

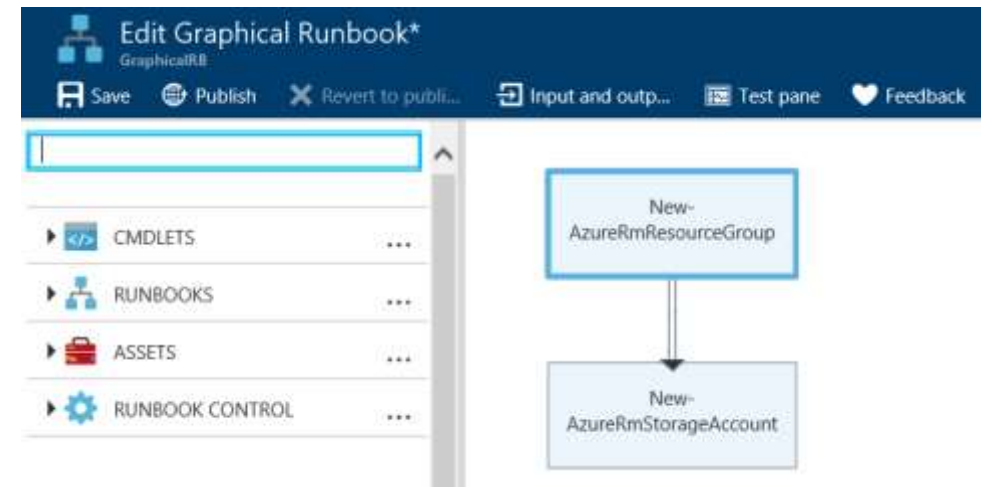
Graphical Runbooks Advantages & Limitations

Advantages

- Create Runbooks with minimal knowledge of PowerShell.
- Visually represent management processes.
- Include other Runbooks as child Runbooks to create high level workflows.

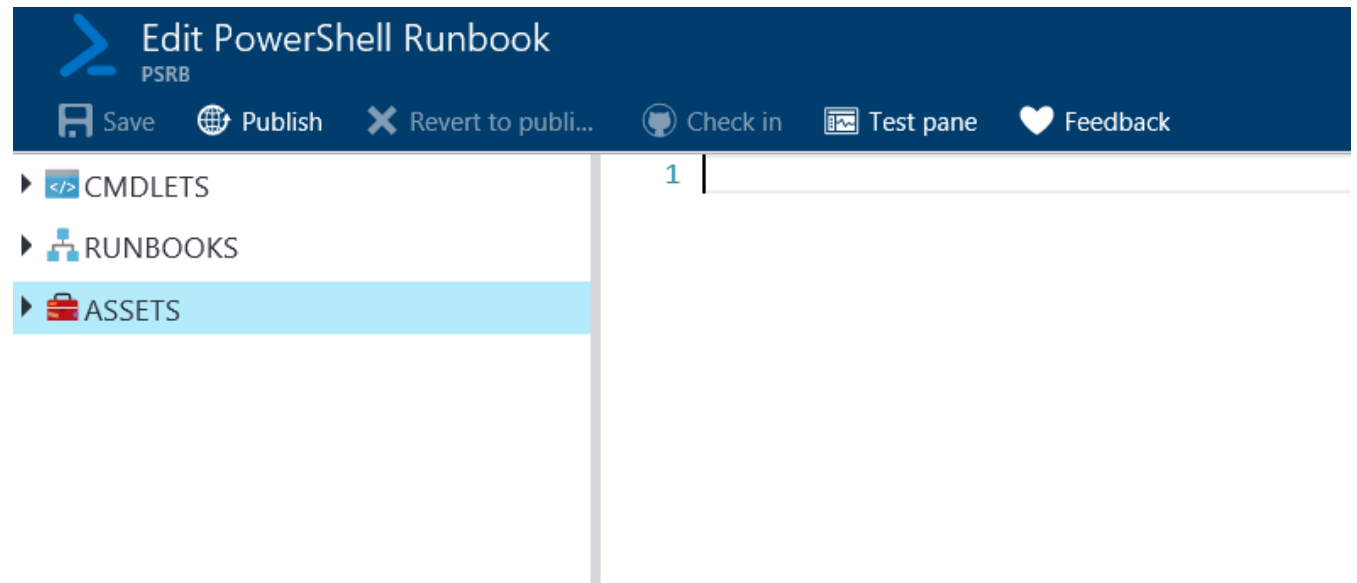
Limitations

- Can't edit a Runbook outside of Azure portal.
- May require PowerShell code to perform complex logic.
- Can't view or directly edit the PowerShell code that is created by the graphical workflow.



PowerShell Runbooks

- PowerShell Runbooks are based on PowerShell.
- Directly edit the code of the Runbook using the text editor in the Azure portal.
- Can also use any offline text editor and import the Runbook into Azure Automation.



PowerShell Runbooks Advantages & Limitations

Advantages

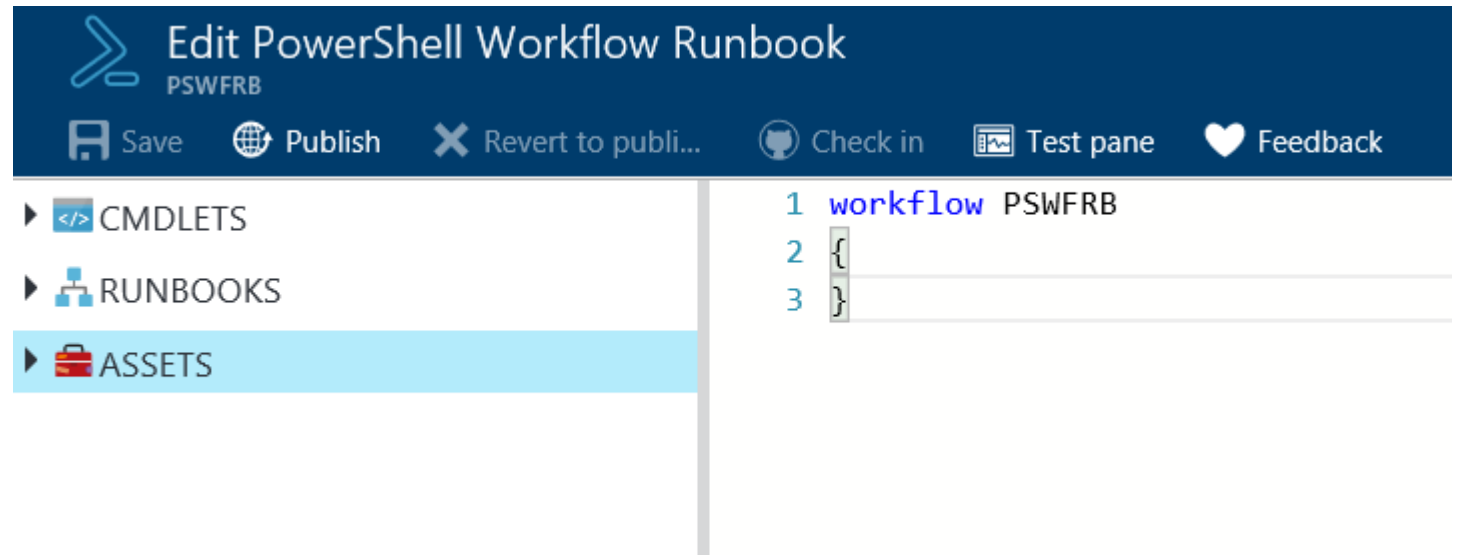
- Implement all complex logic with PowerShell code without the additional complexities of PowerShell Workflow.
- Runbook starts faster than Graphical or PowerShell Workflow Runbooks since it doesn't need to be compiled before running.

Limitations

- Must be familiar with PowerShell scripting.
- Can't use parallel processing to perform multiple actions in parallel.
- Can't use checkpoints to resume Runbook in case of error.

PowerShell Workflow Runbooks

- PowerShell Workflow Runbooks are text Runbooks based on PowerShell Workflow.
- Directly edit the code of the Runbook using the text editor in the Azure portal.
- Can also use any offline text editor and import the Runbook into Azure Automation.



PowerShell Workflow Runbooks Advantages & Limitations

Advantages

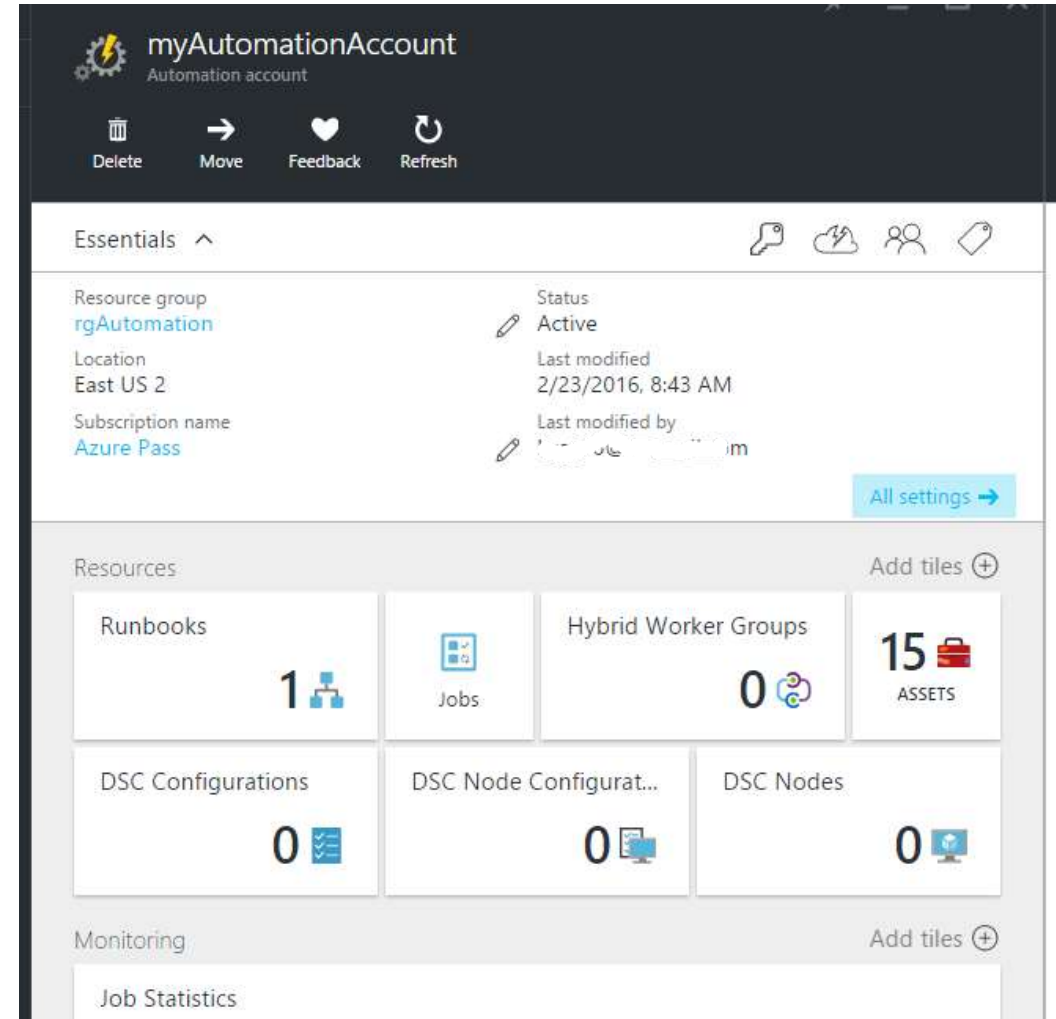
- Implement all complex logic with PowerShell Workflow code.
- Use checkpoints to resume Runbook in case of error.
- Use parallel processing to perform multiple actions in parallel.
- Include other Graphical Runbooks and PowerShell Workflow Runbooks as child Runbooks to create high level workflows.

Limitations

- Author must be familiar with PowerShell Workflow.
- Runbook must deal with the additional complexity of PowerShell Workflow such as deserialized objects.
- Runbook takes longer to start than PowerShell Runbooks since it needs to be compiled before running.

Automation Accounts

- All Runbooks, Hybrid Worker Groups, Assets and DSC configuration are stored in an Automation Account
- Security and Automation management boundary for Azure Automation
- Scope for RBAC
- Location ensures that all Automation components are stored in a particular Azure region
- Specifies the Azure subscription to be billed for Automation usage
- Limited to 25 accounts per subscription



Create & Import Runbooks

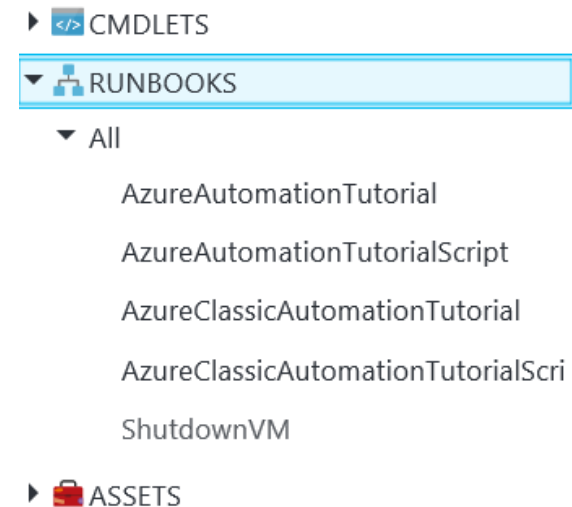
- You can add a Runbook to Azure Automation by either creating a new one or by importing an existing Runbook from a file or from the Runbook Gallery.
- Create a new Runbook using the Azure portal or Windows PowerShell.
- Use `New-AzureRmAutomationRunbook` to create an empty Runbook.
- When you create or import a new Runbook, it must be published before you can run it.

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

Copy

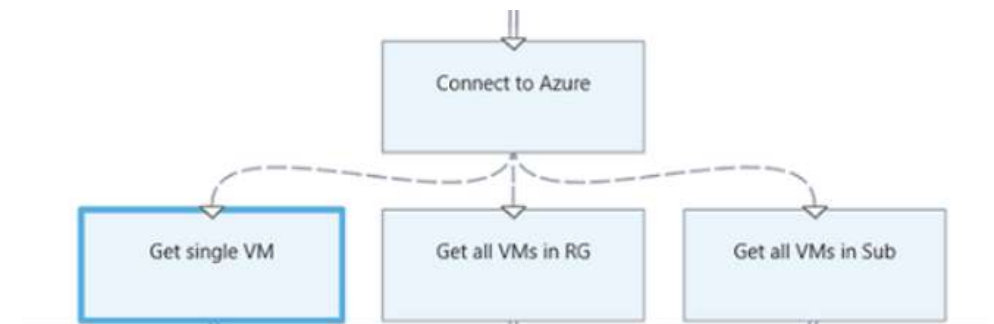
Edit Text Runbooks

- Textual Runbooks can be edited using the text editor in Azure Automation.
- It has the typical features of other code editors such as intellisense and color coding.
- Allows you to select from a list of pre-existing resources instead of typing code in.
- Each Runbook in Azure Automation has two versions, Draft and Published, you edit the Draft version of the Runbook and then publish it so it can be executed.
- To edit a Runbook with Windows PowerShell, use the editor of your choice and save it as a .ps1 file.



Edit Graphic Runbooks

- Graphic Runbooks can be edited using the graphic editor in Azure Automation.
- Select activities from the library control and place them onto the canvas, then edit their parameters in the configuration pane and save and publish.
- Use links to connect two activities in your canvas.
 - Sequence link – destination activity runs after source activity completes
 - Pipeline link – destination activity runs once an output object is received from the source activity
- Conditions can also be specified on a link so that destination activity will run only if true.
- Junctions allow you to run multiple activities in parallel and ensure that all have completed before moving on.
- Use Runbook input & output to provide and receive data to and from a Runbook e.g. VM name.



PowerShell Workflow

- PowerShell Workflow is a PowerShell extension that allows you to run a PowerShell script on multiple devices in parallel with added functionality such as checkpoints, suspend & restart.
- You can convert a PowerShell script to a PowerShell workflow by enclosing it with the Workflow keyword and naming it.
- Consider the following script that gets all running services.

```
Get-Service | Where-Object {$_.Status -eq "Running"}
```

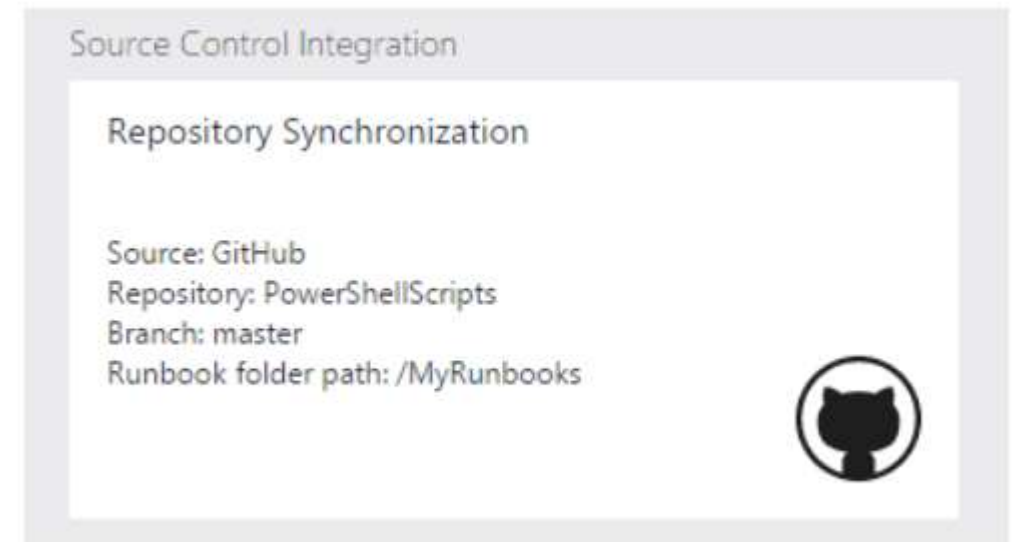
- Converted to a PowerShell Workflow, it is:

```
Workflow Get-RunningServices  
{  
    Get-Service | Where-Object -FilterScript {$_.Status -eq "Running"}  
}
```

- This syntax allows you to benefit from the PowerShell Workflow functionality such as checkpoints, suspend and restart.

Source Control Integration

- Source control integration allows you to push & pull Runbooks from Azure Automation to and from GitHub.
- Create a new or add an existing GitHub account to Azure Automation.
- Configure the GitHub folder path that will be synchronized.
- A check-in from Azure Automation will overwrite code that currently exists in GitHub.
- Synchronization is a manual process and syncs Runbooks only and not ARM templates.



Demo: Graphical Runbook Authoring





Desired State Configuration

Microsoft Services



Automation Desired State Configuration (DSC)

- Azure Automation Desired State Configuration (DSC) allows you to consistently deploy, monitor, and automatically update the desired state of all your IT resources, at scale from the cloud.
- Built on PowerShell DSC and can align machine configuration with a specific state across physical and virtual machines, using Windows or Linux, in the cloud or on-premises
- Builds on top of PowerShell DSC to provide an easier configuration management experience.
- Automation DSC includes:
 - Author and manage PowerShell DSC configurations
 - Import DSC Resources
 - Generate DSC Node configurations (MOF documents)
- DSC configuration files are stored on an Azure Automation DSC Pull server so that target nodes can download and apply them.

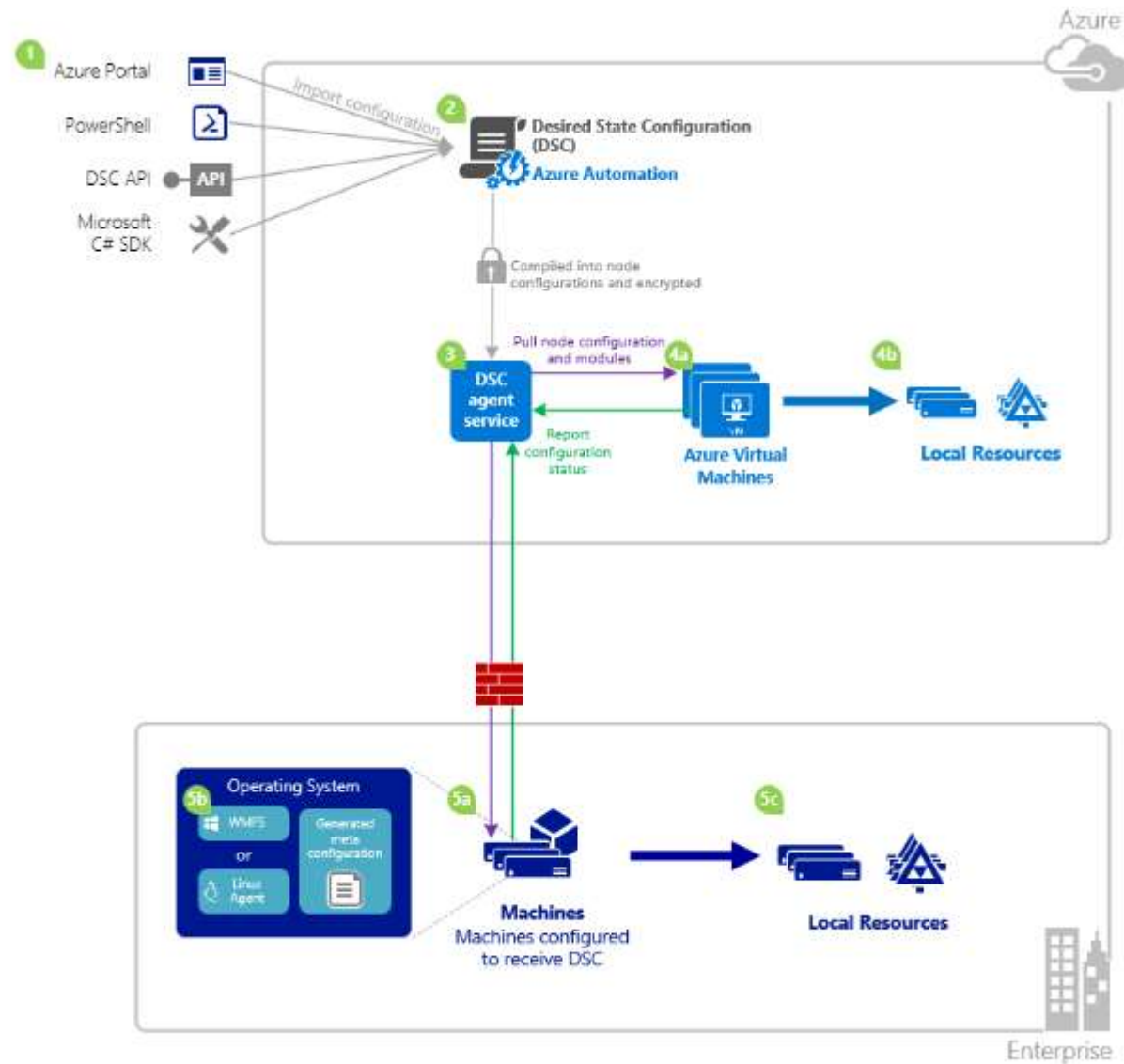


Automation DSC Terms

- Configuration – Introduced in PowerShell DSC and allows you to define the desired state of your environment using PowerShell syntax.
- Node Configuration – Is a file that is produced when a DSC configuration is compiled, this is typically the configuration document that nodes will apply.
- Node – Any machine that has its configuration managed by DSC.
- Resource – A PowerShell module that is used to define a DSC configuration. They are seen as the building blocks of DSC configuration.
- Compilation Job – An instance of compilation of a configuration to create a node configuration.
 - Similar to Azure Automation Runbook jobs, but they do not perform any task, only compile configurations
 - Automatically stored on an Azure DSC pull server
 - Overwrites previous versions of node configurations



Automation DSC Process



Onboarding Nodes for Management

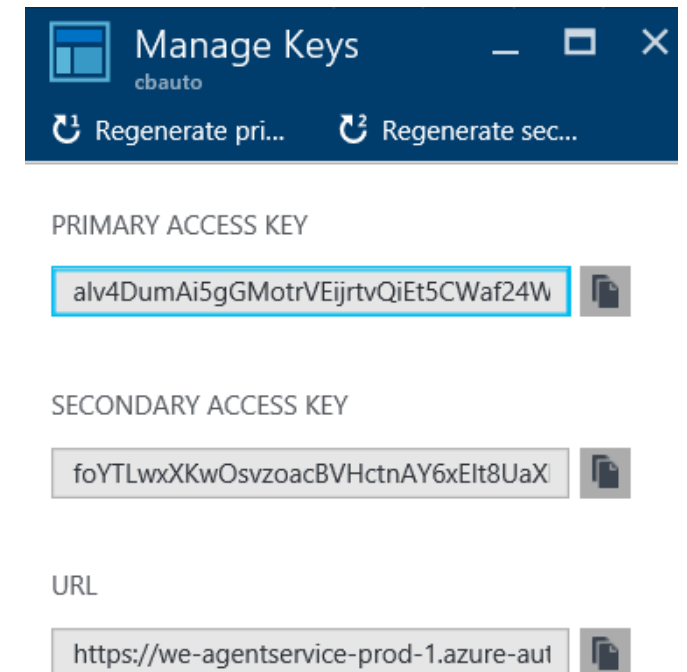
- Azure Automation DSC allows onboarding of the following machines:
 - Azure Classic VM's
 - Azure Resource Manager VM's
 - Amazon Web Services VM's
 - Physical or virtual Windows machines on-premises, or in a cloud other than Azure or AWS
 - Physical or virtual Linux machines on-premises, in Azure, or in a cloud other than Azure
- Azure classic VM's can be onboarded via the new Azure portal or PowerShell.
- Azure Resource Manager VM's can be onboarded via the new Azure portal, ARM templates or PowerShell.
- Amazon Web Services VM's can be onboarded using the AWS DSC Toolkit.
- Windows machines on-premises or in a cloud other than Azure or AWS must have WMF 5.0 installed and have the PowerShell DSC metaconfiguration applied.
- Linux machines on-premises, in Azure or in another cloud must have the latest DSC Linux agent installed and have the PowerShell DSC metaconfiguration applied.

DSC Node Registration Parameters

- **Registration key** – Specifies which Automation account Access Key to use for the DSC node to authenticate with.
- **Node Configuration Name** – Specifies the name of the DSC configuration file to be used.
- **Refresh Frequency** – Specifies how often the DSC node will contact the Pull server and download the latest node configuration.
- **Configuration Mode Frequency** – Specifies how often the downloaded node configuration will be applied.
- **Configuration Mode** – Specifies the mode of configuration e.g. ApplyAndMonitor, ApplyOnly.
- **Allow Module Override** – Specifies whether or not newer modules downloaded from the Pull server are allowed to overwrite older modules on the DSC node.
- **Reboot Node if Needed** – Specifies whether or not to reboot following a configuration update.
- **Action after Reboot** – Specifies actions to take following a reboot e.g. ContinueConfiguration or StopConfiguration.

DSC Metaconfiguration & Secure Registration

- DSC Metaconfiguration is a script that consists of the DSC engine settings that will be used to connect a node to a DSC pull server and keep it updated.
- DSC metaconfigurations for Azure Automation DSC can be generated using either a PowerShell DSC configuration, or the Azure Automation PowerShell cmdlets.
- Must be applied to on-premises or non Azure virtual machines in order to onboard the server to Automation DSC.
- Secure Registration is a registration protocol, allows a DSC node to authenticate to a PowerShell DSC V2 Pull server (including Azure Automation DSC).
- The node registers to the DSC Pull server at a Registration URL, and authenticates using a Registration key specified in the DSC Metaconfiguration.
- A certificate is generated and used for future communication between the node and the DSC Pull server.



Compiling Configuration

- DSC configuration can be compiled in two ways with Azure Automation, the Azure portal or with Windows PowerShell.

Azure portal

- Simplest method with interactive user interface
- Form to provide simple parameter values
- Easily track job state
- Access authenticated with Azure logon

Windows PowerShell

- Call from command line with PowerShell cmdlets
- Can be included in automated solution with multiple steps
- Provide simple and complex parameter values
- Track job state
- Client required to support PowerShell cmdlets
- Pass ConfigurationData
- Compile configurations that use credentials



Essentials ^

Demo: Desired State Configuration





Lab: Automation with DSC

Microsoft Services





Monitoring & Troubleshooting

Microsoft Services



Automation Monitoring

- Azure Automation provides monitoring on the status of Runbooks and DSC configurations.
- Job status logs can be viewed in the Azure portal or using the PowerShell cmdlet `Get-AzureRmAutomationJob`
- Advanced logging scenarios requires you to create custom PowerShell scripts or send job logs to Microsoft Operations Management Suite (OMS) Log Analytics.

The screenshot displays the Azure Automation job monitoring interface. It is divided into two main sections: 'Overview' and 'Status'.

Overview Section:

- Job Summary:** Displays job details including Job ID (420d838a-d854-473b-9142-2b33bd72eff2), creation time (9/20/2016 12:36 PM), last update time (9/20/2016 12:37 PM), and the environment (CBOOnPrem). A red 'X' icon and the word 'Failed' indicate the job's status.
- Runbook:** A button with a blue icon representing a runbook.
- INPUT:** A button with a '0' and an arrow icon, indicating no input parameters.
- Output:** A button with an arrow icon, indicating the output of the job.

Status Section:

- Errors:** A box showing '1' with a red 'X' icon, indicating one error.
- Warnings:** A box showing '0' with a yellow warning triangle icon, indicating no warnings.
- All Logs:** A button with an orange icon representing logs.

Exception Details:

The 'Exception' section is expanded, showing the following message:

The term 'Add-AzureRmAccount' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the spelling of the name, or if a path was included, verify that the path is correct and try again.

Microsoft Operations Management Suite (OMS) Log Analytics

- Log Analytics is a service in Operations Management Suite (OMS) that helps you collect and analyze data generated by resources in your cloud and on-premises environments.
- Provides real-time insights using integrated search and custom dashboards to readily analyze records across all of your workloads and servers regardless of their physical location.
- Log Analytics allows you to:
 - Get insight on your Automation jobs
 - Trigger an email or alert based on your Runbook job status (e.g. failed or suspended)
 - Write advanced queries across your job streams
 - Correlate jobs across Automation accounts
 - Visualize your job history over time



Automation & OMS Log Analytics

- Log analytics is billed per GB of data uploaded into the service, and offers a free data plan with up to 500MB of data per day with a retention period of 7 days.
- If the 500MB daily limit is reached, data analyzing will stop and resume at the start of the next day.
- Requirements in order to use OMS Log Analytics with Automation:
 - An OMS subscription with the OMS workspace and Automation account in the same Azure subscription.
 - A storage account that exists in the same region as the Automation account.
 - Azure PowerShell with version 1.0.8 or newer of the Operational Insights cmdlets.
 - Azure Diagnostic and Log Analytics PowerShell.
 - The PowerShell script `Enable-AzureDiagnostics.ps1`



Demo: Automation Monitoring



Automation Troubleshooting

- Automation logs are useful during troubleshooting Runbook and DSC configuration errors.
- Error logs can be viewed in the Azure portal or by using the PowerShell cmdlet `Get-AzureRmAutomationJob` and can be used as a starting point to help isolate the problem.
- Some of the most common troubleshooting scenarios include:
 - Authentication errors when working with Azure Automation Runbooks
 - Errors when working with Runbooks
 - Errors when importing modules
 - Errors when working with Desired State Configuration (DSC)

✓ Completed	ShutdownVM	21/09/2016 ...	21/09/2016 11:54
✓ Completed	AzureAutomationTutorialScript	20/09/2016 ...	20/09/2016 13:02
✗ Failed	AzureAutomationTutorialScript	20/09/2016 ...	20/09/2016 12:45
✗ Failed	AzureAutomationTutorialScript	20/09/2016 ...	20/09/2016 12:37

Sign in to Azure Account failed

Cause:

This error occurs if the credential asset name is not valid or if the username and password that you used to setup the Automation credential asset are not valid.

Error: You receive the error "Unknown_user_type: Unknown User Type" when working with the `Add-AzureAccount` or `Login-AzureRmAccount` cmdlets.

Resolution:

Make sure that you don't have any special characters, including the @ character in the Automation credential asset name that you are using to connect to Azure and check that you can use the username and password that are stored in the Azure Automation credential in your local PowerShell ISE editor.

Unable to find the Azure subscription

Cause:

This error occurs if the subscription name is not valid or if the Azure Active Directory user who is trying to get the subscription details is not configured as an admin of the subscription.

Error: You receive the error "The subscription named `<subscription name>` cannot be found" when working with the `Select-AzureSubscription` or `Select-AzureRmSubscription` cmdlets.

Resolution:

Make sure that you run the `Add-AzureAccount` before running the `Select-AzureSubscription` cmdlet and if you still see this error message, modify your code by adding the `Get-AzureSubscription` cmdlet following the `Add-AzureAccount` cmdlet and then execute the code. Then verify if the output of `Get-AzureSubscription` contains your subscription details.

Authentication to Azure failed due to MFA being enabled

Cause:

If you have multi-factor authentication on your Azure account, you can't use an Azure Active Directory user to authenticate to Azure. Instead, you need to use a certificate or a service principal to authenticate to Azure.

Error: You receive the error “Add-AzureAccount: AADSTS50079: Strong authentication enrollment (proof-up) is required” when authenticating to Azure with your Azure username and password.

Resolution:

Use a certificate or a service principal to authenticate to Azure.

Runbook fails because of deserialized object

Cause:

If your Runbook is a PowerShell Workflow, it stores complex objects in a deserialized format in order to persist your Runbook state if the workflow is suspended.

Error: Your runbook fails with the error "Cannot bind parameter `<ParameterName>`. Cannot convert the `<ParameterType>` value of type Deserialized `<ParameterType>` to type `<ParameterType>`".

Resolution:

If you are piping complex objects from one cmdlet to another, wrap these cmdlets in an InlineScript or pass the name or value that you need from the complex object instead of passing the entire object or use a PowerShell Runbook instead of a PowerShell Workflow Runbook.

Runbook job failed because the allocated quota is exceeded

Cause:

This error occurs when the job execution exceeds the 500-minute free quota for your account and applies to all types of job execution tasks such as testing a job, starting a job from the portal, executing a job by using Webhooks and scheduling a job to execute by using either the Azure portal or in your datacenter.

Error: Your runbook job fails with the error "The quota for the monthly total job run time has been reached for this subscription".

Resolution:

If you want to use more than 500 minutes of processing per month you will need to change your subscription from the Free tier to the Basic tier.

Cmdlet not recognized when executing a Runbook

Cause:

This error is caused when the PowerShell engine cannot find the cmdlet you are using in your Runbook. This could be because the module containing the cmdlet is missing from the account, there is a name conflict with a Runbook name, or the cmdlet also exists in another module and Automation cannot resolve the name.

Error: Your runbook job fails with the error "`<cmdlet name>`: The term `<cmdlet name>` is not recognized as the name of a cmdlet, function, script file, or operable program."

Resolution:

Check that you have entered the cmdlet name correctly or make sure the cmdlet exists in your Automation account and that there are no conflicts or if you do have a name conflict and the cmdlet is available in two different modules, you can resolve this by using the fully qualified name for the cmdlet.

Runbook job repeatedly evicted from the same checkpoint

Cause:

This is by design behavior due to the "Fair Share" monitoring of processes within Azure Automation, which automatically suspends a Runbook if it executes longer than 3 hours.

Error: A long running runbook consistently fails with the exception: "The job cannot continue running because it was repeatedly evicted from the same checkpoint".

Resolution:

Use Checkpoints in a workflow

Module fails to import or cmdlets can't be executed after importing

Cause:

The structure does not match the structure that Automation needs it to be in or the module is dependent on another module that has not been deployed to your Automation account or the module is missing its dependencies in the folder.

Error: A module fails to import or imports successfully, but no cmdlets are extracted.

Resolution:

Make sure that the module follows the following format: ModuleName.Zip -> ModuleName or Version Number -> (ModuleName.psm1, ModuleName.psd1) or open the .psd1 file and see if the module has any dependencies. If it does, upload these modules to the Automation account or make sure that any referenced .dlls are present in the module folder.

DSC node is in failed status with a "Not found" error

Cause:

This error typically occurs when the node is assigned to a configuration name (e.g. ABC) instead of a node configuration name (e.g. ABC.WebServer).

Error: The node has a report with **Failed** status and containing the error "The attempt to get the action from server https://<url>//accounts/<account-id>/Nodes(AgentId=<agent-id>)/GetDscAction failed because a valid configuration <guid> cannot be found."

Resolution:

Make sure that you are assigning the node with "node configuration name" and not the "configuration name". You can assign a node configuration to a node using Azure portal or with a PowerShell cmdlet.

No node configurations produced when a configuration is compiled

Cause:

When the expression following the Node keyword in the DSC configuration evaluates to \$null then no node configurations will be produced.

Error: Your DSC compilation job suspends with the error: "Compilation completed successfully, but no node configuration .mofs were generated".

Resolution:

Make sure that the expression next to the Node keyword in the configuration definition is not evaluating to \$null or if you are passing ConfigurationData when compiling the configuration, make sure that you are passing the expected values that the configuration requires from ConfigurationData.

The DSC node report becomes stuck “in progress” state

Cause:

You have upgraded your WMF version and have corrupted WMI.

Error: DSC Agent outputs “No instance found with given property values.”

Resolution:

Delete DSCEngineCache.mof by running the following command in an elevated PowerShell session (Run as Administrator):

```
Remove-Item -Path $env:SystemRoot\system32\Configuration\DSCEngineCache.mof
```

Unable to use a credential in a DSC configuration

Cause:

You have used a credential in a configuration but didn't provide proper ConfigurationData to set PSDscAllowPlainTextPassword to true for each node configuration.

Error: Your DSC compilation job was suspended with the error:

"System.InvalidOperationException error processing property 'Credential' of type `<some resource name>`: Converting and storing an encrypted password as plaintext is allowed only if PSDscAllowPlainTextPassword is set to true".

Resolution:

Make sure to pass in the proper ConfigurationData to set PSDscAllowPlainTextPassword to true for each node configuration mentioned in the configuration

