

Contents

Azure Automation User Documentation

Overview

[What is Automation?](#)

Quickstarts

[Create an Automation account](#)

[Create a runbook](#)

[Create a DSC configuration](#)

Tutorials

[Track updated files with watcher tasks](#)

[Manage updates for your VM](#)

[Onboard update and change tracking by runbook](#)

[Identify software in your environment](#)

[Troubleshoot changes on a VM](#)

How to

Onboarding

[Onboard multiple VMs from the portal](#)

[Onboard from an Azure VM](#)

[Onboard from Automation account](#)

Update Management

[Update Management overview](#)

[Manage updates for multiple VMs](#)

[Integrate SCCM with Update Management](#)

Change Tracking

[Change Tracking overview](#)

[View file contents of tracked files](#)

Inventory

[Manage a VM with inventory collection](#)

Authentication and security

[Create standalone Automation account](#)

- [Create Azure AD user account](#)
- [Configure authentication with AWS](#)
- [Create Automation Run As account](#)
- [Validate Automation account config](#)
- [Manage role-based access control](#)
- [Manage Automation account](#)
- [Create runbooks](#)
 - [Runbook types](#)
 - [Create and import runbooks](#)
 - [Create a graphical runbook](#)
 - [Create a PowerShell runbook](#)
 - [Create a PowerShell workflow runbook](#)
 - [Create a Python runbook](#)
 - [Edit textual runbooks](#)
 - [Edit graphical runbooks](#)
 - [Test a runbook](#)
 - [Learning PowerShell Workflow](#)
 - [Child runbooks](#)
 - [Runbook output](#)
 - [Source control integration](#)
- [Automate](#)
 - [Start a runbook](#)
 - [Start a runbook from a webhook](#)
 - [Configure runbook input parameters](#)
 - [Error handling in graphical runbooks](#)
 - [Track a runbook job](#)
 - [Change runbook settings](#)
 - [Manage Azure Automation data](#)
 - [Call Azure Automation Runbook from Log Analytics alert](#)
 - [Pass a JSON object to an Azure Automation runbook](#)
- [Hybrid Runbook Worker](#)
 - [Hybrid Runbook Worker overview](#)

- [Deploy Windows Hybrid Runbook Worker](#)
- [Deploy Linux Hybrid Runbook Worker](#)
- [Run runbooks on a Hybrid Runbook Worker](#)
- [Deploy configuration management \(DSC\)](#)
 - [Desired State Configuration \(DSC\) overview](#)
 - [Getting started](#)
 - [Configure servers to a desired state and manage drift with Azure Automation](#)
 - [Onboarding machines for management](#)
 - [Compiling DSC configurations](#)
 - [Continuous deployment using Chocolatey](#)
 - [Forward Azure Automation DSC reporting data to OMS Log Analytics](#)
- [Manage assets](#)
 - [Certificates](#)
 - [Connections](#)
 - [Credentials](#)
 - [Integration modules](#)
 - [Schedules](#)
 - [Variables](#)
 - [Update Azure PowerShell modules](#)
- [Scenarios](#)
 - [Runbook gallery](#)
 - [Monitor runbooks with metric alerts](#)
 - [Start a runbook from Azure alerts](#)
 - [Create Amazon Web Service VM](#)
 - [Start/stop VMs during off-hours](#)
 - [Remove resource group](#)
 - [Source control integration with GitHub Enterprise](#)
 - [Source control integration with VSTS](#)
 - [Call Azure Automation Runbook from Log Analytics alert](#)
 - [Deploy an Azure Resource Manager template in an Azure Automation PowerShell runbook](#)
 - [Integrate Azure Automation with Event Grid and Microsoft Teams](#)
- [Monitor](#)

[Forward Azure Automation job data to Log Analytics](#)

[Unlink Azure Automation account from Log Analytics](#)

Migrate

[Migrate from Orchestrator](#)

[Move Automation account](#)

Troubleshoot

[Onboarding](#)

[Runbooks](#)

[Desired State Configuration \(DSC\)](#)

[Hybrid Runbook Worker](#)

[Update Management](#)

Reference

[Azure PowerShell](#)

[Azure PowerShell \(Classic\)](#)

[.NET](#)

[REST](#)

[REST \(Classic\)](#)

Resources

[Automation introduction video](#)

[Azure Roadmap](#)

[MSDN forum](#)

[Pricing](#)

[Pricing calculator](#)

[Release notes](#)

[Service updates](#)

[Stack Overflow](#)

[Videos](#)

An introduction to Azure Automation

5/10/2018 • 4 minutes to read • [Edit Online](#)

Azure Automation delivers a cloud-based automation and configuration service that provides consistent management across your Azure and non-Azure environments. It consists of process automation, update management, and configuration features. Azure Automation provides complete control during deployment, operations, and decommissioning of workloads and resources. This article provides a brief overview of Azure Automation and answers some common questions. For more information about the different capabilities, visit the links throughout this overview.

Azure Automation capabilities

	Process Automation Orchestrate processes using graphical, PowerShell, and Python runbooks		Shared capabilities Role based access control Secure, global store for variables, credentials, certificates, connections Flexible scheduling Shared modules Source control support Auditing Tags
	Configuration Management Collect inventory Track changes Configure desired state		
	Update Management Assess compliance Schedule update installation		Heterogenous Windows & Linux Azure and on-premises

Process automation

Azure Automation provides you the ability to automate frequent, time-consuming, and error-prone cloud management tasks. This automation helps you focus on work that adds business value. By reducing errors and boosting efficiency, it also helps to lower your operational costs. You can integrate Azure services and other public systems that are required in deploying, configuring, and managing your end to end processes. The service allows you to [author runbooks](#) graphically, in PowerShell, or Python. By using a hybrid Runbook worker, you can unify management by orchestrating across on-premises environments. [Webhooks](#) provide a way to fulfill requests and ensure continuous delivery and operations by triggering automation from ITSM, DevOps, and monitoring systems.

Configuration management

Azure Automation [desired state configuration](#) is a cloud-based solution for PowerShell DSC that provides services required for enterprise environments. Manage your DSC resources in Azure Automation and apply configurations to virtual or physical machines from a DSC Pull Server in the Azure cloud. It provides rich reports that inform you of important events such as when nodes have deviated from their assigned configuration. You can monitor and automatically update machine configuration across physical and virtual machines, Windows or Linux, in the cloud or on-premises.

You can get inventory about in-guest resources for visibility into installed applications and other configuration items. A rich reporting and search capabilities are available to quickly find detailed information to help understand what is configured within the operating system. You can track changes across services, daemons, software, registry,

and files to quickly identify what might be causing issues. Additionally, DSC can help you diagnose and alert when unwanted changes occur in your environment.

Update management

Update Windows and Linux systems across hybrid environments with Azure Automation. You get visibility of update compliance across Azure, on-premises, and other clouds. You can create schedule deployments to orchestrate the installation of updates within a defined maintenance window. If an update should not be installed on a machine, you can exclude those updates from a deployment.

Shared capabilities

Azure Automation consists of a set of shared resources that make it easier to automate and configure your environments at scale.

- **Role-based access control** - Control access to the account with an Automation operator role that enables tasks to be run without giving authoring capabilities.
- **Variables** - Provide a way to hold content that can be used across runbooks and configurations. You can change values without having to modify any of the runbooks and configurations that reference them.
- **Credentials** - Securely store sensitive information that can be used by runbooks and configurations at runtime.
- **Certificates** - Store and make available at runtime so they can be used for authentication and securing deployed resources.
- **Connections** - Store a name / value pairs of information that contains common information when connecting to systems in connection resources. Connections are defined by the module author for use at runtime in runbooks and configurations.
- **Schedules** - Used in the service to trigger automation on predefined times.
- **Integration with source control** - Promotes configuration as code where runbooks or configurations can be checked into a source control system.
- **PowerShell modules** - Modules are used to manage Azure and other systems. Import into the Automation account for Microsoft, third party, community, or custom defined cmdlets and DSC resources.

Windows and Linux

Azure Automation is designed to work across your hybrid cloud environment and also for Windows & Linux. It delivers a consistent way to automate and configure workloads deployed and the operating system they are running on.

Community gallery

Browse the [Automation gallery](#) for runbooks and modules to quickly get started integrating and authoring your processes from PowerShell gallery and Microsoft Script Center.

Common scenarios for Automation

Azure Automation manages across the lifecycle of your infrastructure and applications. Transfer knowledge into the system on how the organization delivers and maintains workloads. Author in common languages like PowerShell, desired state configuration, Python, and graphical runbooks. Get a complete inventory of deployed resources for targeting, reporting, and compliance. Identify changes that can cause misconfiguration and improve operational compliance.

- **Build / Deploy resources** - Deploy VMs across a hybrid environment using Runbooks and Azure Resource Manager templates. Integrate into development tools like Jenkins and Visual Studio Team services.
- **Configure VMs** - Assess and configure Windows and Linux machines with the desired configuration for the infrastructure and application.
- **Monitor** - Identify changes on machines that are causing issues and remediate or escalate to management systems.
- **Protect** - Quarantine VM if security alert is raised. Set in-guest requirements.

- **Govern** - Set up role-based access control for teams. Recover unused resources.

Pricing for Automation

You can review the price for Azure Automation on the [pricing](#) page.

Next steps

[Create an automation account](#)

Create an Azure Automation account

5/10/2018 • 2 minutes to read • [Edit Online](#)

Azure Automation accounts can be created through Azure. This method provides a browser-based user interface for creating and configuring Automation accounts and related resources. This quickstart steps through creating an Automation account and running a runbook in the account.

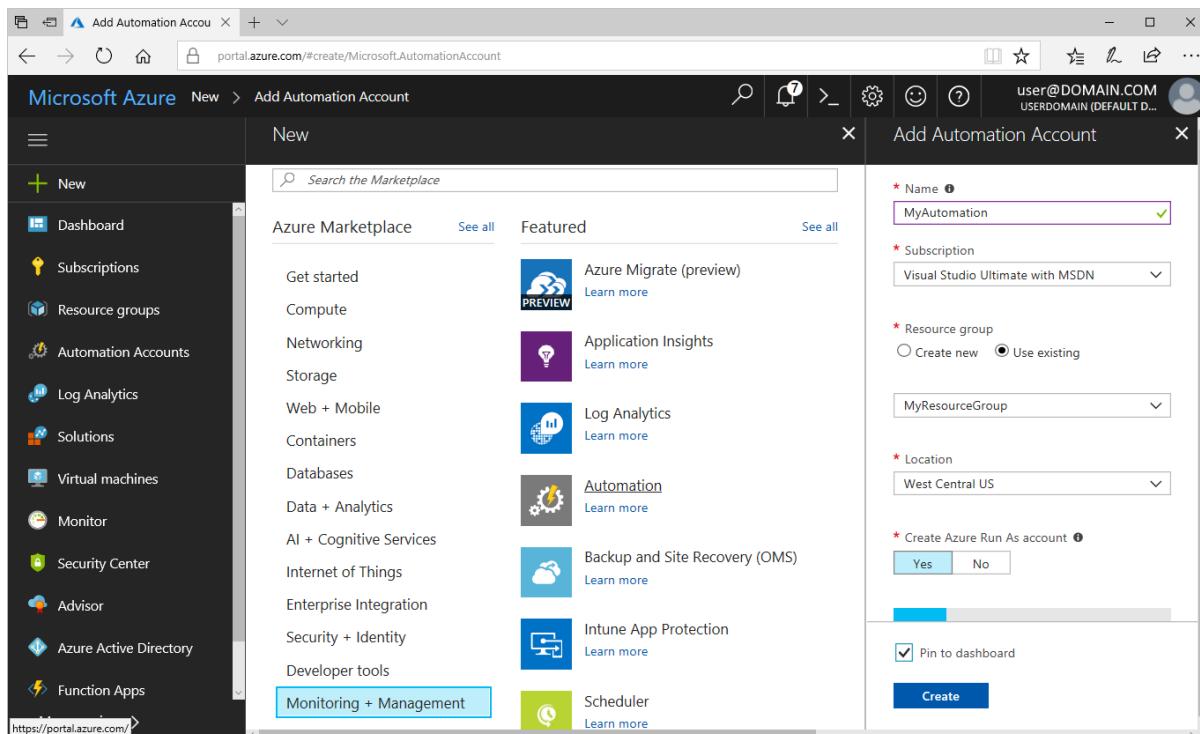
If you don't have an Azure subscription, create a [free Azure account](#) before you begin.

Log in to Azure

Log in to Azure at <https://portal.azure.com>

Create Automation account

1. Click the **Create a resource** button found on the upper left-hand corner of Azure.
2. Select **Monitoring + Management**, and then select **Automation**.
3. Enter the account information. For **Create Azure Run As account**, choose **Yes** so that the artifacts to simplify authentication to Azure are enabled automatically. When complete, click **Create**, to start the Automation account deployment.



4. The Automation account is pinned to the Azure dashboard. When the deployment has completed, the Automation account overview automatically opens.

The screenshot shows the Azure portal interface for the 'MyAutomation' Automation Account. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Configuration Management (Inventory, Change tracking, DSC nodes, DSC configurations, DSC node configurations), Update Management (Update management), and Process Automation (Runbooks, Jobs). The main content area displays the 'Essentials' section with details about the resource group: MyResourceGroup, Status: Active, Location: West Central US, and Last modified: 12/11/2017 8:15 PM. Below this is a 'Monitoring' section titled 'Job Statistics' featuring a donut chart and a table of job counts:

Status	Count
FAILED	0
SUSPENDED	0
COMPLETED	0

Legend: QUEUED (purple), RUNNING (light blue), COMPLETED (yellow-green), FAILED (pink), STOPPED (orange), SUSPENDED (cyan).

Run a runbook

Run one of the tutorial runbooks.

1. Click **Runbooks** under **PROCESS AUTOMATION**. The list of runbooks is displayed. By default several tutorial runbooks are enabled in the account.

The screenshot shows the 'Runbooks' section of the Azure Automation blade. On the left, a navigation menu includes 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', 'Diagnose and solve problems', 'Configuration Management' (with 'Inventory (Preview)', 'Change tracking (Preview)', 'DSC nodes', 'DSC configurations', 'DSC node configurations'), 'Update Management' (with 'Update management (Preview)'), 'Process Automation' (with 'Runbooks' selected), and 'Jobs'. The main area displays a table of runbooks:

Name	Authoring Status	Last Modified	Tags
AzureAutomationTutorial	✓ Published	12/11/2017 8:15 PM	
AzureAutomationTutorialPython2	✓ Published	12/11/2017 8:15 PM	
AzureAutomationTutorialScript	✓ Published	12/11/2017 8:15 PM	
AzureClassicAutomationTutorial	✓ Published	12/11/2017 8:15 PM	
AzureClassicAutomationTutorial...	✓ Published	12/11/2017 8:15 PM	

2. Select the **AzureAutomationTutorialScript** runbook. This action opens the runbook overview page.

The screenshot shows the 'AzureAutomationTutorialScript' runbook overview page. The left sidebar contains sections for 'Overview', 'Activity log', 'Tags', 'Diagnose and solve problems', 'RESOURCES' (with 'Jobs', 'Schedules', 'Webhooks'), 'RUNBOOK SETTINGS' (with 'Properties', 'Description', 'Logging and tracing'), 'SETTINGS' (with 'Locks', 'Automation script'), and 'SUPPORT + TROUBLESHOOTING'. The main content area has tabs for 'Start', 'View', 'Edit', 'Schedule', 'Webhook', 'Delete', 'Export', and 'Refresh'. The 'Essentials' tab is selected, displaying the following details:

Resource group	Status
MyResourceGroup	Published
Account	Runbook type
MyAutomation	PowerShell Runbook
Location	Last modified
West Central US	12/11/2017 8:15 PM
Subscription name	
Visual Studio Ultimate with MSDN	

The 'Details' tab shows a summary of the runbook's configuration.

3. Click **Start**, and on the **Start Runbook** page, click **OK** to start the runbook.

The screenshot shows the Azure Automation Job Overview page for a runbook named "AzureAutomationTutorialScript". The job was created on 12/11/2017 at 8:40 PM and is currently running. It was run as a User and ran on Azure. The runbook is titled "AzureAutomationTutorialScript". The "Input" section shows 0 items. The "Output" section shows 0 items. The "Logs" section shows 0 items. The "Errors" section shows 0 errors (indicated by a red X icon). The "Warnings" section shows 0 warnings (indicated by an orange exclamation mark icon). The "Exception" section shows "None".

4. After the **Job status** becomes **Running**, click **Output** or **All Logs** to view the runbook job output. For this tutorial runbook, the output is a list of your Azure resources.

Clean up resources

When no longer needed, delete the resource group, Automation account, and all related resources. To do so, select the resource group for the Automation account and click **Delete**.

Next steps

In this quickstart, you've deployed an Automation account, started a runbook job, and viewed the job results. To learn more about Azure Automation, continue to the quickstart for creating your first runbook.

[Automation Quickstart - Create Runbook](#)

Create an Azure Automation runbook

5/10/2018 • 2 minutes to read • [Edit Online](#)

Azure Automation runbooks can be created through Azure. This method provides a browser-based user interface for creating Automation runbooks. In this quickstart you walk through creating, editing, testing, and publishing an Automation PowerShell runbook.

If you don't have an Azure subscription, create a [free Azure account](#) before you begin.

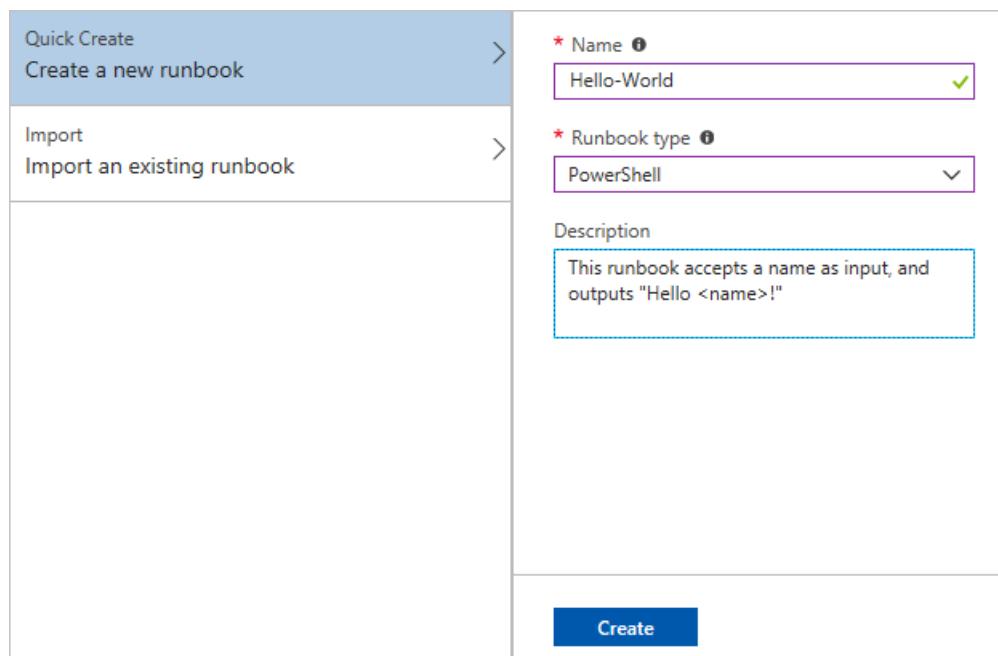
Log in to Azure

Log in to Azure at <https://portal.azure.com>

Create runbook

First, create a runbook. The sample runbook created in this quickstart outputs `Hello World` by default.

1. Open your Automation account.
2. Click **Runbooks** under **PROCESS AUTOMATION**. The list of runbooks is displayed.
3. Click the **Add a runbook** button found at the top of the list. On the **Add Runbook** page, select **Quick Create**.
4. Enter "Hello-World" for the runbook **Name**, and select **PowerShell** for **Runbook type**. Click **Create**.



5. The runbook is created and the **Edit PowerShell Runbook** page opens.

Edit PowerShell Runbook
Hello-World

Save Publish Revert to published Check in Test pane Feedback

CMDLETS RUNBOOKS ASSETS

- Type or copy and paste the following code into the edit pane. It creates an optional input parameter called "Name" with a default value of "World", and outputs a string that uses this input value:

```
param
(
    [Parameter(Mandatory=$false)]
    [String] $Name = "World"
)

"Hello $Name!"
```

- Click **Save**, to save a draft copy of the runbook.

Edit PowerShell Runbook
Hello-World

Save Publish Revert to published Check in Test pane Feedback

CMDLETS RUNBOOKS ASSETS

```
1 param
2 (
3     [Parameter(Mandatory=$false)]
4     [String] $Name = "World"
5 )
6
7 "Hello $Name!"
```

Test the runbook

Once the runbook is created, you test the runbook to validate that it works.

- Click **Test pane** to open the **Test** page.
- Enter a value for **Name**, and click **Start**. The test job starts and the job status and output display.

Test
Hello-World

Start Stop Suspend Resume View last test

Parameters

NAME Optional, String, Default: "World"

Azure

Run Settings

Run on Azure Hybrid Worker

Completed

Hello Azure!

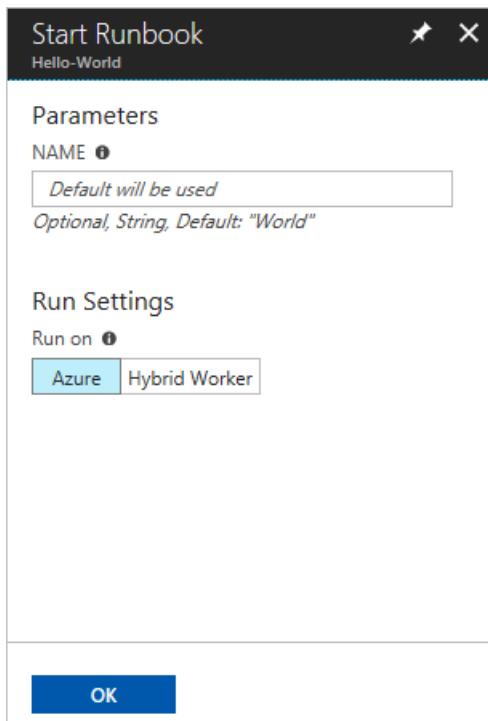
- Close the **Test** page by clicking the **X** in the upper right corner. Select **OK** in the popup that appears.
- In the **Edit PowerShell Runbook** page, click **Publish** to publish the runbook as the official version of the runbook in the account.

```
param
(
    [Parameter(Mandatory=$false)]
    [String] $Name = "World"
)
"Hello $Name!"
```

Run the runbook

Once the runbook is published, the overview page is shown.

1. In the runbook overview page, click **Start** to open the **Start Runbook** configuration page for this runbook.



2. Leave **Name** blank, so that the default value is used, and click **OK**. The runbook job is submitted, and the job page appears.

The screenshot shows the 'Hello-World' runbook job details. At the top, there are buttons for Resume, Stop, and Suspend. Below that is the 'Essentials' section with details like Job Id, Created date, Job status (Completed), Last Update date, Runbook name ('Hello-World'), and Source snapshot link. The 'Overview' section includes a summary of Input (0), Output (highlighted with a red box), and All Logs. It also shows Error (0) and Warning (0) counts. The 'Exception' section indicates 'None'.

- When the **Job status** is **Running** or **Completed**, click **Output** to open the **Output** pane and view the runbook output.

The screenshot shows the 'Output' pane for the 'Hello-World' runbook. The title is 'Output' and the subtitle is 'Hello-World 12/14/2017 8:04 PM'. The main content area displays the single line of output: 'Hello World!'

Clean up resources

When no longer needed, delete the runbook. To do so, select the runbook in the runbook list, and click **Delete**.

Next steps

In this quickstart, you've created, edited, tested, and published a runbook and started a runbook job. To learn more about Automation runbooks, continue to the article on the different runbook types that you can create and use in Automation.

[Automation How To - Runbook Types](#)

Configure a Linux virtual machine with Desired State Configuration

5/10/2018 • 4 minutes to read • [Edit Online](#)

By enabling Desired State Configuration (DSC), you can manage and monitor the configurations of your Windows and Linux servers. Configurations that drift from the desired configuration can be identified or auto-corrected. This quickstart steps through onboarding a Linux VM and deploying a LAMP stack with DSC.

Prerequisites

To complete this quickstart, you need:

- An Azure subscription. If you don't have an Azure subscription, [create a free account](#).
- An Azure Automation account. For instructions on creating an Azure Automation Run As account, see [Azure Run As Account](#).
- An Azure Resource Manager VM (not Classic) running Red Hat Enterprise Linux, CentOS, or Oracle Linux. For instructions on creating a VM, see [Create your first Linux virtual machine in the Azure portal](#)

Log in to Azure

Log in to Azure at <https://portal.azure.com>

Onboard a virtual machine

There are many different methods to onboard a machine and enable Desired State Configuration. This quickstart covers onboarding through an Automation account. You can learn more about different methods to onboard your machines to Desired State Configuration by reading the [onboarding](#) article.

1. In the left pane of the Azure portal, select **Automation accounts**. If it is not visible in the left pane, click **All services** and search for it in the resulting view.
2. In the list, select an Automation account.
3. In the left pane of the Automation account, select **DSC Nodes**.
4. Click the menu option to **Add Azure VM**
5. Find the virtual machine you would like to enable DSC for. You can use the search field and filter options to find a specific virtual machine.
6. Click on the virtual machine, and then select **Connect**
7. Select the DSC settings appropriate for the virtual machine. If you have already prepared a configuration, you can specify it as *Node Configuration Name*. You can set the [configuration mode](#) to control the configuration behavior for the machine.
8. Click **OK**

While the Desired State Configuration extension is deployed to the virtual machine, it shows *Connecting*.

Import modules

Modules contain DSC Resources and many can be found on the [PowerShell Gallery](#). Any resources that are used in your configurations must be imported to the Automation Account before compiling. For this tutorial, the module named **nx** is required.

1. In the left pane of the Automation account, select **Modules Gallery** (under Shared Resources).
2. Search for the module that you would like to import by typing part of its name: *nx*
3. Click on the module you would like to import
4. Click **Import**

Import the configuration

This quickstart uses a DSC configuration that configures Apache HTTP Server, MySQL, and PHP on the machine.

For information about DSC configurations, see [DSC configurations](#).

In a text editor, type the following and save it locally as `LAMPServer.ps1`.

```

configuration LAMPServer {
    Import-DSCResource -module "nx"

    Node localhost {

        $requiredPackages = @("httpd","mod_ssl","php","php-mysql","mariadb","mariadb-server")
        $enabledServices = @("httpd","mariadb")

        #Ensure packages are installed
        ForEach ($package in $requiredPackages){
            nxPackage $Package{
                Ensure = "Present"
                Name = $Package
                PackageManager = "yum"
            }
        }

        #Ensure daemons are enabled
        ForEach ($service in $enabledServices){
            nxService $service{
                Enabled = $true
                Name = $service
                Controller = "SystemD"
                State = "running"
            }
        }
    }
}

```

To import the configuration:

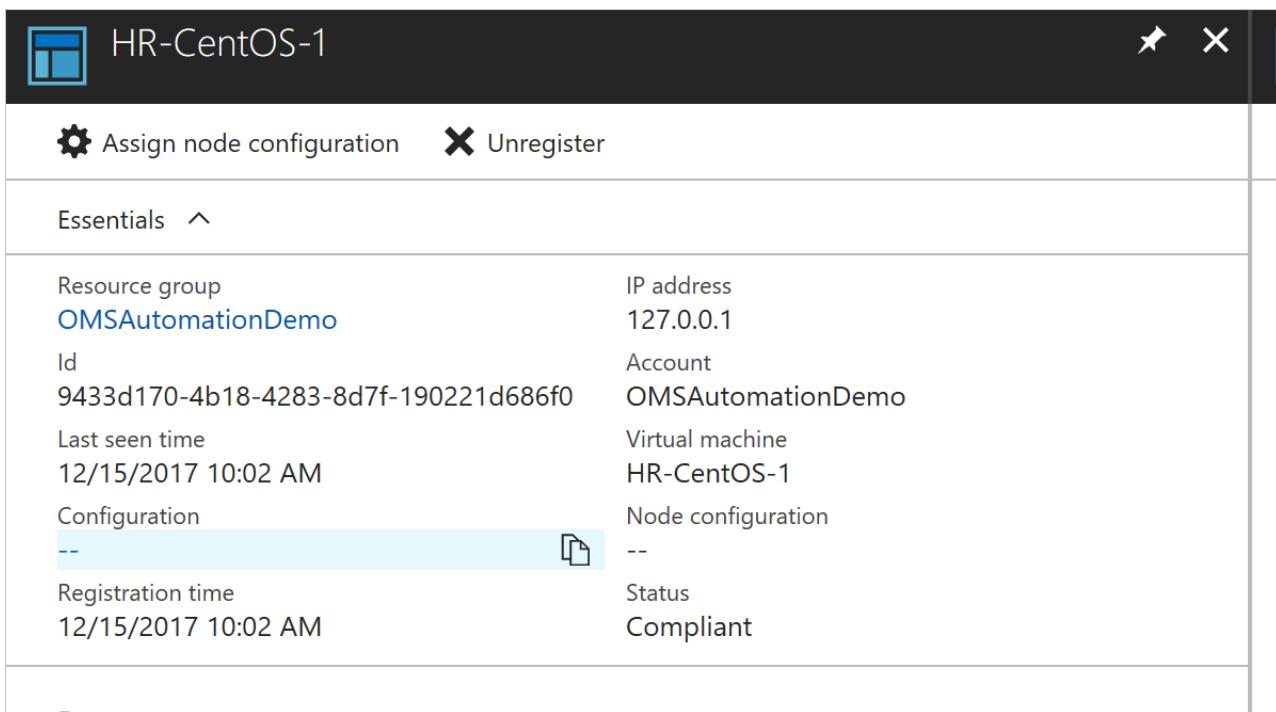
1. In the left pane of the Automation account, select **DSC Configurations**.
2. Click the menu option to **Add a Configuration**
3. Select the *Configuration file* that you saved in the prior step
4. Click **OK**

Compile a configuration

DSC Configurations must be compiled to a Node Configuration (MOF document) before being assigned to a node. Compilation validates the configuration and allows for the input of parameter values. To learn more about compiling a configuration, see: [Compiling Configurations in Azure Automation DSC](#)

To compile the configuration:

1. In the left pane of the Automation account, select **DSC Configurations**.
2. Select the configuration you imported in a prior step, "LAMPServer"
3. From the menu options, click **Compile** and then **Yes**
4. In the Configuration view, you see a new *Compilation job* queued. When the job has completed successfully, you are ready to move on to the next step. If there are any failures, you can click on the Compilation job for details.



The screenshot shows the Azure portal interface for a node named 'HR-CentOS-1'. At the top, there are buttons for 'Assign node configuration' and 'Unregister'. Below this, a section titled 'Essentials' is expanded. The node's details are listed in pairs:

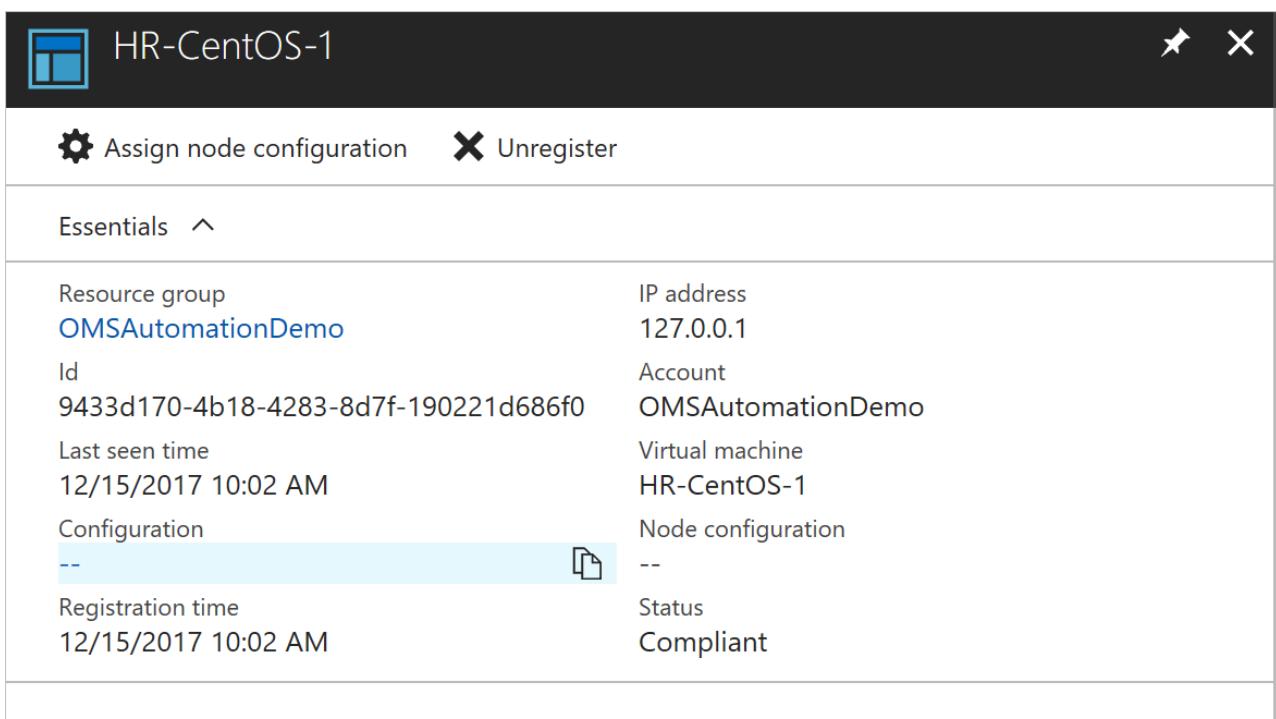
Resource group	IP address
OMSAutomationDemo	127.0.0.1
Id	Account
9433d170-4b18-4283-8d7f-190221d686f0	OMSAutomationDemo
Last seen time	Virtual machine
12/15/2017 10:02 AM	HR-CentOS-1
Configuration	Node configuration
--	--
Registration time	Status
12/15/2017 10:02 AM	Compliant

Assign a node configuration

A compiled *Node Configuration* can be assigned to DSC Nodes. Assignment applies the configuration to the machine and monitors (or auto-corrects) for any drift from that configuration.

1. In the left pane of the Automation account, select **DSC Nodes**
2. Select the node you would like to assign a configuration to
3. Click **Assign Node Configuration**
4. Select the *Node Configuration* - **LAMPServer.localhost** - to assign and click **OK**
5. The compiled configuration is now be assigned to the node, and the node status changes to *Pending*. On the next periodic check, the node retrieves the configuration, apply it, and report status back. It can take up to 30 minutes for the node to retrieve the configuration, depending on the node's settings. To force an immediate check, you can run the following command locally on the Linux virtual machine:

```
sudo /opt/microsoft/dsc/Scripts/PerformRequiredConfigurationChecks.py
```



The screenshot shows the Azure portal interface for the same node 'HR-CentOS-1'. The 'Assign node configuration' button is still present at the top. The 'Essentials' section is expanded, showing the updated configuration details:

Resource group	IP address
OMSAutomationDemo	127.0.0.1
Id	Account
9433d170-4b18-4283-8d7f-190221d686f0	OMSAutomationDemo
Last seen time	Virtual machine
12/15/2017 10:02 AM	HR-CentOS-1
Configuration	Node configuration
--	--
Registration time	Status
12/15/2017 10:02 AM	Compliant

Viewing node status

The status of all managed nodes can be found in the **DSC Nodes** view of the Automation Account. You can filter the display by status, node configuration, or name search.

NAME	STATUS	NODE CONFIGURATION	LAST SEEN	VM DSC EXTENSION VERSION...
HR-CentOS-1	Compliant	lampserver.localhost	12/15/2017 10:31 AM	Yes
HR-Ubuntu-2	Compliant		12/15/2017 10:30 AM	Yes

Next steps

In this quickstart, you onboarded a Linux VM to DSC, created a configuration for a LAMP stack, and deployed it to the VM. To learn how you can use Automation DSC to enable continuous deployment, continue to the article:

[Continuous deployment to a VM using DSC and Chocolatey](#)

- To learn more about PowerShell Desired State Configuration, see [PowerShell Desired State Configuration Overview](#).
- To learn more about managing Automation DSC from PowerShell, see [Azure PowerShell](#)
- To learn how to forward DSC reports to Log Analytics for reporting and alerting, see [Forwarding DSC Reporting to Log Analytics](#)

Create an Azure Automation watcher tasks to track file changes on a local machine

5/21/2018 • 4 minutes to read • [Edit Online](#)

Azure Automation uses watcher tasks to watch for events and trigger actions. This tutorial walks you through creating a watcher task to monitor when a new file is added to a directory.

In this tutorial, you learn how to:

- Import a watcher runbook
- Create an Automation variable
- Create an action runbook
- Create a watcher task
- Trigger a watcher
- Inspect the output

Prerequisites

To complete this tutorial, the following are required:

- Azure subscription. If you don't have one yet, you can [activate your MSDN subscriber benefits](#) or sign up for a [free account](#).
- [Automation account](#) to hold the watcher and action runbooks and the Watcher Task.
- A [hybrid runbook worker](#) where the watcher task runs.

Import a watcher runbook

This tutorial uses a watcher runbook called **Watch-NewFile** to look for new files in a directory. The watcher runbook retrieves the last known write time to the files in a folder and looks at any files newer than that watermark. In this step, you import this runbook into your automation account.

1. Open your Automation account, and click on the **Runbooks** page.
2. Click on the **Browse gallery** button.
3. Search for "Watcher runbook", select **Watcher runbook that looks for new files in a directory** and select **Import**.

Watcher runbook that looks for new files in a directory

[View Source](#)

 Import

Watcher runbook that looks for new files in a directory and triggers an action runbook to process the files. This is a sample to show how to create and use watcher tasks in an Automation account. It requires that a variable called "Watch-NewFileTimestamp" be created first.

Created by: SC Automation Product Team - Microsoft

Tags: [Runbook](#), [Automation Watcher](#)

[View Source Project](#)

85 downloads

Last updated: 11/17/2017

```

1 <#
2 .SYNOPSIS
3     This sample automation runbook is designed to be used in a watcher task that
4     looks for new files in a directory. When a new file is found, it calls the action
5     runbook associated with the watcher task.
6
7 .DESCRIPTION
8     This sample automation runbook is designed to be used in a watcher task that
9     looks for new files in a directory. When a new file is found, it calls the action
10    runbook associated with the watcher task. It requires that a variable called
11    "Watch-NewFileTimestamp"
12    be created in the automaiton account that is used to hold the timestamp of the last file
13    processed.
14
15 .PARAMETER FolderPath
16     Required. The name of a folder that you wish to watch for new files.
17
18 .PARAMETER Extension
19     Optional. The extension of files that you want to filter on. Default is *.*.
20
21 .PARAMETER Recurse
22     Optional. Determines whether to look for all files under all directories or just the
      specific
      folder. Default is the folder only.
23

```

ATTENTION
 Each runbook is licensed to you under a license agreement by its owner, not Microsoft. Microsoft is not responsible for runbooks provided & licensed by the community members and does not screen for security, compatibility or performance. The runbooks are not supported under any Microsoft support program or service. The runbooks are provided AS IS without warranty of any kind.

4. Give the runbook a name and description and select **OK** to import the runbook into your Automation account.
5. Select **Edit** and then click **Publish**. When prompted select **Yes** to publish the runbook.

Create an Automation variable

An [automation variable](#) is used to store the timestamps that the preceding runbook reads and stores from each file.

1. Select **Variables** under **SHARED RESOURCES** and select **+ Add a variable**.
2. Enter "Watch-NewFileTimestamp" for the name
3. Select **DateTime** for Type.
4. Click on the **Create** button. This creates the automation variable.

Create an action runbook

An action runbook is used in a watcher task to act on the data passed to it from a watcher runbook. In this step, you update import a pre-defined action runbook called "Process-NewFile".

1. Navigate to your automation account and select **Runbooks** under the **PROCESS AUTOMATION** category.
2. Click on the **Browse gallery** button.
3. Search for "Watcher action" and select **Watcher action that processes events triggered by a watcher runbook** and select **Import**.

Watcher action that processes events triggered by a watcher runbook

[View Source](#)

Import

This sample automation runbook is designed to be used in a watcher task that takes action on data passed in from a watcher runbook. It is required to have a parameter called \$EVENTDATA in watcher action runbooks to receive information from the watcher runbook.

Created by: SC Automation Product Team - Microsoft

Tags: Runbook, Automation Watcher

[View Source Project](#)

Ratings: 5 of 5
26 downloads
Last updated: 11/17/2017

```

1 <#
2 .SYNOPSIS
3     This sample automation runbook is designed to be used in a watcher task that takes action
4     on data passed in from a watcher runbook.
5
6 .DESCRIPTION
7     This sample automation runbook is designed to be used in a watcher task that takes action
8     on data passed in from a watcher runbook. It is required to have a parameter called
9     $EVENTDATA in
10    watcher action runbooks to receive information from the watcher runbook.
11 .PARAMETER EVENTDATA
12     Optional. Contains the information passed in from the watcher runbook.
13
14 .NOTES
15     AUTHOR: Automation Team
16     LASTEDIT: Nov 12th, 2017
17 #>
18
19 param(
20     $EVENTDATA
21 )
22
23 Write-Output("Message is " + $EVENTDATA.EventProperties.Message)

```

ATTENTION
Each runbook is licensed to you under a license agreement by its owner, not Microsoft. Microsoft is not responsible for runbooks provided & licensed by the community members and does not screen for security, compatibility or performance. The runbooks are not supported under any Microsoft support program or service. The runbooks are provided AS IS without warranty of any kind.

4. Give the runbook a name and description and select **OK** to import the runbook into your Automation account.
5. Select **Edit** and then click **Publish**. When prompted select **Yes** to publish the runbook.

Create a watcher task

The watcher task contains two parts. The watcher and the action. The watcher runs at an interval defined in the watcher task. Data from the watcher runbook is passed onto the action runbook. In this step, you configure the watcher task referencing the watcher and action runbooks defined in the preceding steps.

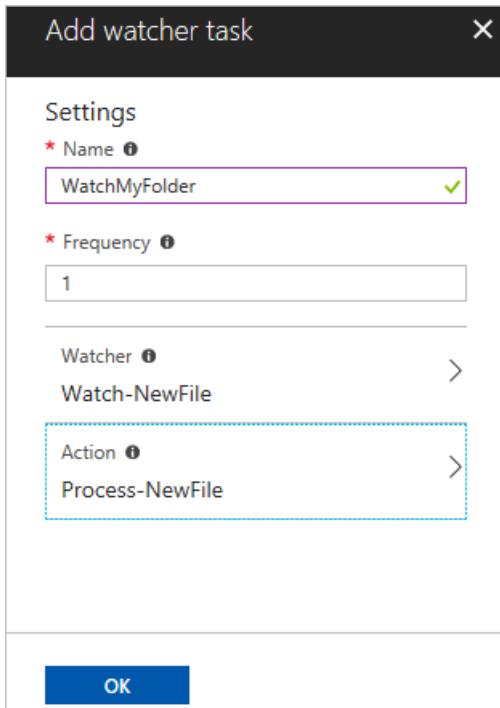
1. Navigate to your automation account and select **Watcher tasks** under the **PROCESS AUTOMATION** category.
2. Select the Watcher tasks page and click on **+ Add a watcher task** button.
3. Enter "WatchMyFolder" as the name.
4. Select **Configure watcher** and select the **Watch-NewFile** runbook.
5. Enter the following values for the parameters:
 - **FOLDERPATH** - A folder on the hybrid worker where new files get created. d:\examplefiles
 - **EXTENSION** - Leave blank to process all file extensions.
 - **RECURSE** - Leave this value as the default.
 - **RUN SETTINGS** - Pick the hybrid worker.
6. Click OK, and then Select to return to the watcher page.
7. Select **Configure action** and select "Process-NewFile" runbook.

8. Enter the following values for parameters:

- **EVENTDATA** - Leave blank. Data is passed in from the watcher runbook.
- **Run Settings** - Leave as Azure as this runbook runs in the Automation service.

9. Click **OK**, and then Select to return to the watcher page.

10. Click **OK** to create the watcher task.



Trigger a watcher

To test the watcher is working as expected, you need to create a test file.

Remote into the hybrid worker. Open **PowerShell** and create a test file in the folder.

```
New-Item -Name ExampleFile1.txt
```

The following example shows the expected output.

```
Directory: D:\examplefiles

Mode                LastWriteTime         Length Name
----                -----          -----
-a---  12/11/2017  9:05 PM           0 ExampleFile1.txt
```

Inspect the output

1. Navigate to your automation account and select **Watcher tasks** under the **PROCESS AUTOMATION** category.
2. Select the watcher task "WatchMyFolder".
3. Click on **View watcher streams** under **Streams** to see that the watcher found the new file and started the action runbook.
4. To see the action runbook jobs, click on the **View watcher action jobs**. Each job can be selected to view the details of the job.

STATUS	CREATED	LAST UPDATED
<input checked="" type="checkbox"/> ✓ Completed	11/11/2017 1:13 PM	11/11/2017 1:13 PM
✓ Completed	11/11/2017 1:11 PM	11/11/2017 1:11 PM
✓ Completed	11/11/2017 1:09 PM	11/11/2017 1:09 PM
✓ Completed	11/11/2017 12:59 PM	11/11/2017 12:59 PM

The expected output when the new file is found can be seen in the following example:

```
Message is Process new file...
Passed in data is @{FileName=D:\examplefiles\ExampleFile1.txt; Length=0}
```

Next steps

In this tutorial, you learned how to:

- Import a watcher runbook
- Create an Automation variable
- Create an action runbook
- Create a watcher task
- Trigger a watcher
- Inspect the output

Follow this link to learn more about authoring your own runbook.

[My first PowerShell runbook.](#)

Manage Windows updates by using Azure Automation

6/20/2018 • 7 minutes to read • [Edit Online](#)

You can use the Update Management solution to manage updates and patches for your virtual machines. In this tutorial, you learn how to quickly assess the status of available updates, schedule installation of required updates, review deployment results, and create an alert to verify that updates apply successfully.

For pricing information, see [Automation pricing for Update Management](#).

In this tutorial, you learn how to:

- Onboard a VM for Update Management
- View an update assessment
- Configure alerting
- Schedule an update deployment
- View the results of a deployment

Prerequisites

To complete this tutorial, you need:

- An Azure subscription. If you don't have one yet, you can [activate your monthly Azure credit for Visual Studio subscribers](#) or sign up for a [free account](#).
- An [Azure Automation account](#) to hold the watcher and action runbooks and the Watcher Task.
- A [virtual machine](#) to onboard.

Sign in to Azure

Sign in to the Azure portal at <https://portal.azure.com>.

Enable Update Management

First, enable Update Management on your VM for this tutorial:

1. In the Azure portal, in the left menu, select **Virtual machines**. Select a VM from the list.
2. On the VM page, under **OPERATIONS**, select **Update management**. The **Enable Update Management** pane opens.

Validation is performed to determine whether Update Management is enabled for this VM. This validation includes checks for an Azure Log Analytics workspace and linked Automation account, and whether the Update Management solution is in the workspace.

A [Log Analytics](#) workspace is used to collect data that's generated by features and services like Update Management. The workspace provides a single location to review and analyze data from multiple sources.

The validation process also checks to see whether the VM is provisioned with the Microsoft Monitoring Agent (MMA) and Automation Hybrid Runbook Worker. This agent is used to communicate with Azure Automation and to obtain information about the update status. The agent requires port 443 to be open to communicate with the Azure Automation service and to download updates.

If any of the following prerequisites were found to be missing during onboarding, they're automatically added:

- A [Log Analytics workspace](#)
- An [Automation account](#)
- A [Hybrid Runbook Worker](#) (enabled on the VM)

Under **Update Management**, set the location, Log Analytics workspace, and Automation account to use. Then, select **Enable**. If these options aren't available, it means that another automation solution is enabled for the VM. In that case, the same workspace and Automation account must be used.

The screenshot shows the Azure portal interface for managing updates on a virtual machine. The top navigation bar includes 'Home', 'Resource groups', 'ContosoVM', and 'ContosoVM4 - Update management'. The main title is 'ContosoVM4 - Update management' with a subtitle 'Virtual machine'. On the left, a sidebar lists 'Automation script', 'OPERATIONS' (Auto-shutdown, Backup, Disaster recovery (Preview), Update management, Inventory, Change tracking, State configuration (Preview)), and 'MONITORING' (Metrics). The 'Update management' item is selected and highlighted in blue. The main content area is titled 'Update Management' with a sub-instruction 'Enable consistent control and compliance of this VM with Update Management.' It explains that the service is included with Azure virtual machines and requires a Log Analytics workspace and Automation account. It shows dropdown menus for 'Location' (set to 'East US'), 'Log Analytics workspace' (set to 'defaultworkspace-147a22e9-2356-4e56-...'), and 'Automation account' (set to 'Automate-147a22e9-2356-4e56-b3de-1f...'). A large blue 'Enable' button is at the bottom.

Enabling the solution can take up to a few minutes. During this time, don't close the browser window. After the solution is enabled, information about missing updates on the VM flows to Log Analytics. It can take between 30 minutes and 6 hours for the data to be available for analysis.

View update assessment

After Update Management is enabled, the **Update management** pane opens. If any updates are missing, a list of missing updates is shown on the **Missing updates** tab.

Under **INFORMATION LINK**, select the update link to open the support article for the update in a new window. You can learn important information about the update in this window.

The screenshot shows the 'Update management' configuration page for 'ContosoVM1 - Update management'. The sidebar on the left has 'Update management' selected. The main area displays a summary of update status: 'Compliance' (green checkmark), 'Missing updates (2)' (0 Critical, 0 Security, 2 Others), 'Failed update deployments (0)' (0 out of 1 in the past six months), and links to 'Learn more', 'Update Management for Windows', and 'Provide feedback'. Below this, the 'Missing updates (2)' tab is selected, showing a table with two rows of update details. The first row is for '2018-02 Cumulative Update for Windows Server 2016 for x64-based System...' (Published Date: 2/21/2018, Classification: Updates, Information Link: KB4077525). The second row is for 'Windows Malicious Software Removal Tool x64 - February 2018 (KB890830)' (Published Date: 2/12/2018, Classification: Update rollups, Information Link: KB890830). The 'Information Link' for the second update is highlighted with a red box.

Click anywhere else on the update to open the **Log Search** pane for the selected update. The query for the log search is predefined for that specific update. You can modify this query or create your own query to view detailed information about the updates that are deployed or missing in your environment.

The screenshot shows the Log Search pane with the following details:

- Header:** Log Search, workspace-2fa536b3-be37-4397-bd48-b24eba9bd3ee-eus
- Top Bar:** Refresh, Analytics, Export, PowerBI
- Message Bar:** Our query editor now supports multi-line queries. [Learn more](#)
- Left Panel:**
 - Data based on last 1 day
 - 1 bar = 1hr
 - 10:00:00 AM Dec 13, 2017
 - TYPE (1)
 - Update 201
 - OSVERSION (0)
- Right Panel:**
 - Show legacy language converter
 - Update query: | where OSType=="Linux" and Optional==false and (VMUUID=~"5f8f75c5-1ac6-463a-bbee-2d22205ebea3" or VMUUID=~"c5758f5f-c61a-3a46-bbee-2d22205ebea3") | summarize hint.strategy=partitioned arg_max(TimeGenerated, *) by Computer, SourceComputerId, UpdateID
 - 1 Results
 - Table view showing one result row:

TimeGenerated	Computer	ApprovalSource	Type	Approved
12/13/2017 11:14:08.320 AM	ContosoVM1	Windows Update	Update	true
 - Advanced Analytics 00:00:12.465

Configure alerts

In this step, you set an alert to let you know when updates have been successfully deployed. The alert you create is based on a Log Analytics query. You can write a custom query for additional alerts to cover many different scenarios. In the Azure portal, go to **Monitor**, and then select **Create Alert**.

Under **Create rule**, under **1. Define alert condition**, select **Select target**. Under **Filter by resource type**, select **Log Analytics**. Select your Log Analytics workspace, and then select **Done**.

The screenshot shows the Azure portal's Create rule wizard and a separate "Select a resource" dialog.

Create rule (Left Panel):

- Step 1: Define alert condition**
 - Alert condition configuration requires 1) Target selection and 2) Alert criteria definition.
 - Alert target:** Microsoft Azure (selected)
 - Select the target(s) that you wish to monitor:** + Select target (button highlighted with a red box).
 - Alert criteria:** No criteria defined, click on 'Add criteria' to select a signal and define the alert logic.
 - Add criteria:** + Add criteria button.
- Step 2: Define alert details**
- Step 3: Define action group**
- Bottom:** Create alert rule button.

Select a resource (Right Panel):

- For metric and log based alert rules please select a specific target, for activity log alert rules you can select a subscription, a resource type or a resource group.
- Filter by subscription:** Microsoft Azure
- Filter by resource type:** Log Analytics (highlighted with a red box).
- Search to filter items...** input field.
- RESOURCE:**
 - mms-eus
 - GWTestWorkspace
 - TestAzureAuto
 - Workspace-** (highlighted with a red box)
- Selection preview:** Microsoft Azure > TestAzureAuto > Workspace-
- Available signal(s):** Log, Activity Log
- Bottom:** Done button.

Select **Add criteria**.

Under **Configure signal logic**, in the table, select **Custom log search**. Enter the following query in the **Search query** text box:

```
UpdateRunProgress  
| where InstallationStatus == 'Succeeded'  
| where TimeGenerated > now(-10m)  
| summarize by UpdateRunName, Computer
```

This query returns the computers and the update run name that completed in the specified timeframe.

Under **Alert logic**, for **Threshold**, enter **1**. When you're finished, select **Done**.

The screenshot shows the 'Alert logic' configuration interface. At the top, there is a search query editor with the following code:

```
* Search query ⓘ  
UpdateRunProgress  
| where InstallationStatus == 'Succeeded'  
| where TimeGenerated > now(-10m)  
| summarize by UpdateRunName, Computer
```

Below the query, the 'Query to be executed' section displays the generated query:

Query to be executed : `UpdateRunProgress | where InstallationStatus == 'Succeeded' | where TimeGenerated > now(-10m) | summarize by UpdateRunName, Computer` | count

For time window : 4/16/2018, 9:05:16 AM - 4/16/2018, 9:10:16 AM

The 'Alert logic' section includes fields for 'Based on' (set to 'Number of results'), 'Condition' (set to 'Greater than'), and 'Threshold' (set to '1').

The 'Condition preview' section states: "Whenever the custom log search is greater than 1 count".

The 'Evaluated based on' section includes fields for 'Period (in minutes)' (set to '5') and 'Frequency (in minutes)' (set to '5').

A blue 'Done' button is located at the bottom right of the configuration area.

Under **2. Define alert details**, enter a name and description for the alert. Set **Severity** to **Informational(Sev 2)** because the alert is for a successful run.

The screenshot shows the '2. Define alert details' configuration page. It includes fields for:

- Alert rule name**: Set to 'Update Run Succeeded'.
- Description**: A text area containing the description: "A list of computers in the Update Run that were successfully patched."
- Severity**: Set to 'Informational(Sev 2)'.
- Enable rule upon creation**: A toggle switch set to 'Yes'.
- Suppress Alerts**: A checkbox that is unchecked.

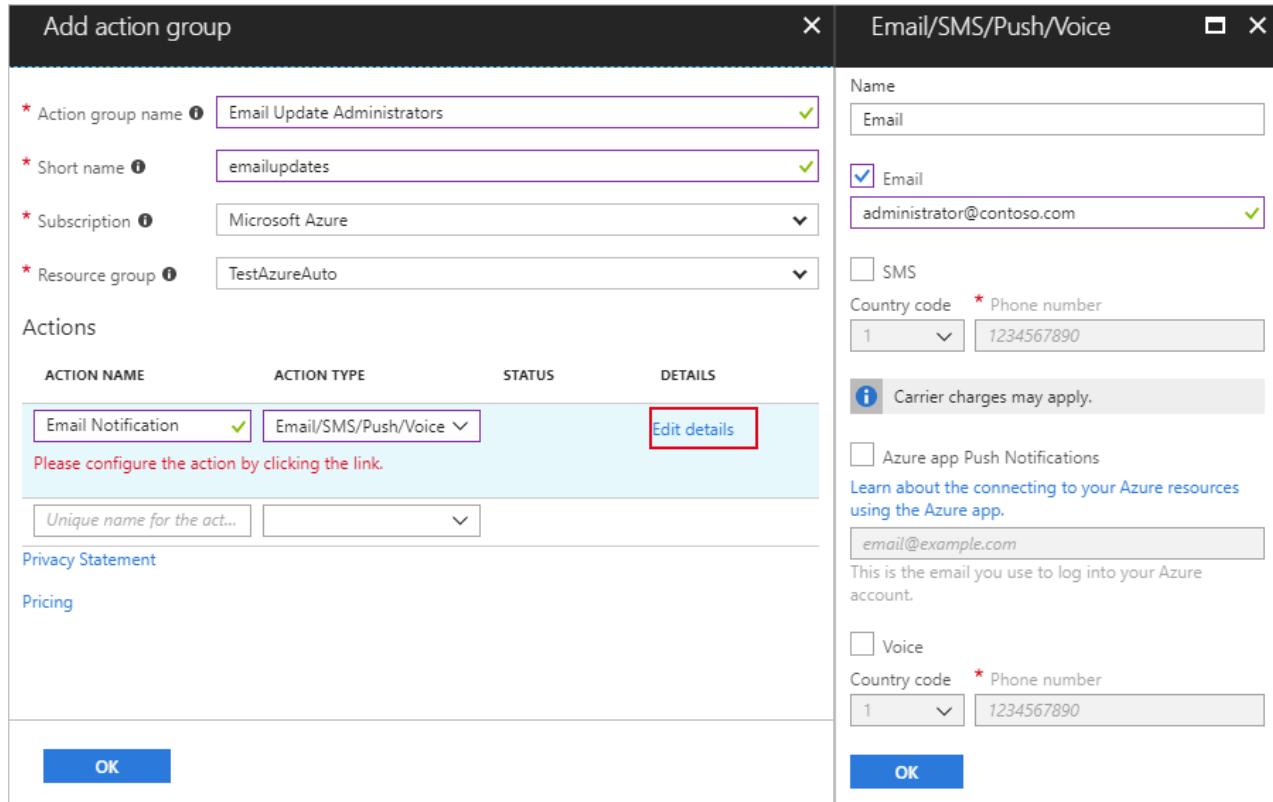
Under **3. Define action group**, select **New action group**. An action group is a group of actions that you can use

across multiple alerts. The actions can include but are not limited to email notifications, runbooks, webhooks, and many more. To learn more about action groups, see [Create and manage action groups](#).

In the **Action group name** box, enter a name for the alert and a short name. The short name is used in place of a full action group name when notifications are sent by using this group.

Under **Actions**, enter a name for the action, like **Email Notifications**. Under **ACTION TYPE**, select **Email/SMS/Push/Voice**. Under **DETAILS**, select **Edit details**.

In the **Email/SMS/Push/Voice** pane, enter a name. Select the **Email** check box, and then enter a valid email address.



In the **Email/SMS/Push/Voice** pane, select **OK**. In the **Add action group** pane, select **OK**.

To customize the subject of the alert email, under **Create rule**, under **Customize Actions**, select **Email subject**. When you're finished, select **Create alert rule**. The alert tells you when an update deployment succeeds, and which machines were part of that update deployment run.

Schedule an update deployment

Next, schedule a deployment that follows your release schedule and service window to install updates. You can choose which update types to include in the deployment. For example, you can include critical or security updates and exclude update rollups.

WARNING

When updates require a restart, the VM is restarted automatically.

To schedule a new update deployment for the VM, go to **Update management**, and then select **Schedule update deployment**.

Under **New update deployment**, specify the following information:

- **Name:** Enter a unique name for the update deployment.

- **Operating system:** Select the OS to target for the update deployment.
- **Update classification:** Select the types of software that the update deployment included in the deployment. For this tutorial, leave all types selected.

The classification types are:

OS	Type
Windows	Critical updates Security updates Update rollups Feature packs Service packs Definition updates Tools Updates
Linux	Critical and security updates Other updates

For a description of the classification types, see [update classifications](#).

- **Schedule settings:** The **Schedule Settings** pane opens. The default start time is 30 minutes after the current time. You can set the start time to any time from 10 minutes in the future.

You can also specify whether the deployment occurs once, or set up a recurring schedule. Under **Recurrence**, select **Once**. Leave the default as 1 day and select **OK**. This sets up a recurring schedule.

- **Maintenance window (minutes):** Leave the default value. You can set the window of time that you want the update deployment to occur within. This setting helps ensure that changes are performed within your defined service windows.

New update deployment □ X

Update Deployment

* Name i
April Updates ✓

Operating system
Windows Linux

* Machines to update >
3 items selected

* Update classifications
8 selected

Updates to exclude >
Click to Configure

* Schedule settings >
One time

Maintenance window (minutes) i
120

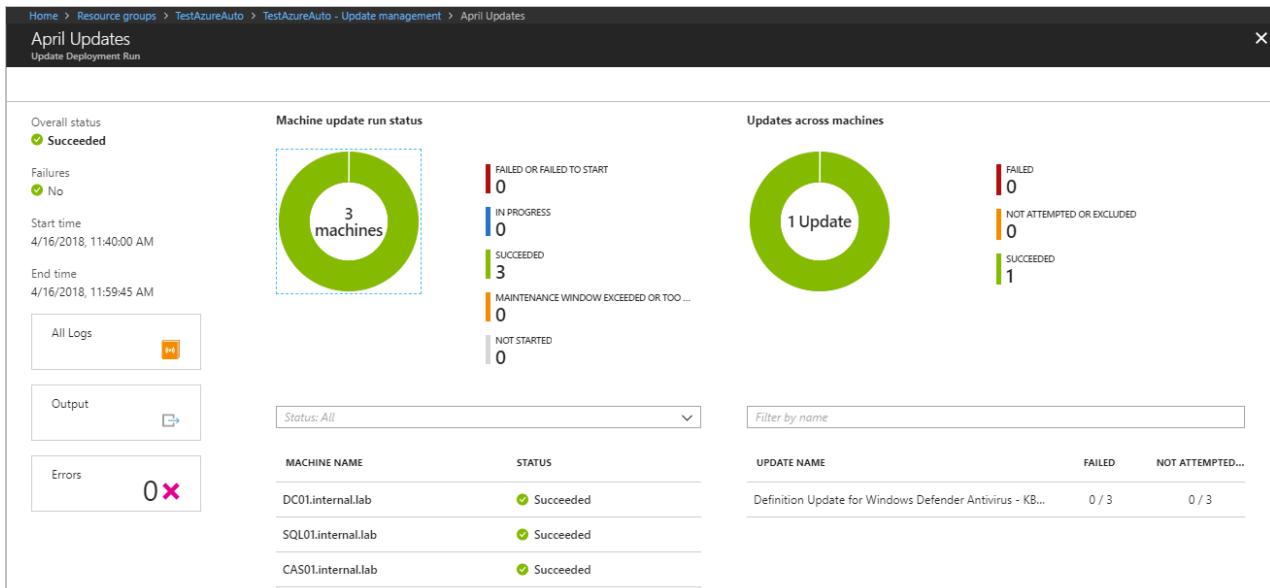
Create

When you're finished configuring the schedule, select **Create**. You're returned to the status dashboard. Select **Scheduled Update deployments** to show the deployment schedule you created.

View results of an update deployment

After the scheduled deployment starts, you can see the status for that deployment on the **Update deployments** tab under **Update management**. The status is **In progress** when the deployment is currently running. When the deployment finishes, if it's successful, the status changes to **Succeeded**. When there are failures with one or more updates in the deployment, the status is **Partially failed**.

Select the completed update deployment to see the dashboard for that update deployment.



Under **Update results**, a summary provides the total number of updates and deployment results on the VM. The

table on the right shows a detailed breakdown of each update and the installation results.

The following list shows the available values:

- **Not attempted:** The update wasn't installed because there was insufficient time available based on the maintenance window duration defined.
- **Succeeded:** The update succeeded.
- **Failed:** The update failed.

Select **All logs** to see all log entries that the deployment created.

Select **Output** to see the job stream of the runbook responsible for managing the update deployment on the target VM.

Select **Errors** to see detailed information about any errors from the deployment.

When your update deployment is successful, an email that's similar to the following example is sent to show success of the deployment:

The screenshot shows an email from Microsoft Azure. The subject is "Updates were successfully deployed". The body of the email includes a summary table and a detailed results table.

Summary Table:

We are notifying you because there are 3 counts of "Update Run Succeeded"	
NAME	Update Run Succeeded
SEVERITY	Informational
RESOURCE	Workspace
SEARCH INTERVAL START TIME	4/16/2018 6:38:27 PM (UTC)
SEARCH INTERVAL DURATION	5 min
SEARCH QUERY	UpdateRunProgress where InstallationStatus == 'Succeeded' where TimeGenerated > now(-10m) summarize by UpdateRunName, Computer
SEARCH RESULTS	3 result(s)
DESCRIPTION	A list of computers in the Update Run that were successfully patched.

Detailed Results Table:

Top 3 result(s)	
UpdateRunName	April Updates
Computer	CAS01.internal.lab
UpdateRunName	April Updates
Computer	SQL01.internal.lab
UpdateRunName	April Updates
Computer	DC01.internal.lab

Next steps

In this tutorial, you learned how to:

- Onboard a VM for Update Management
- View an update assessment
- Configure alerting
- Schedule an update deployment
- View the results of a deployment

Continue to the overview for the Update Management solution.

Update Management solution

Onboard update and change tracking solutions to Azure Automation

6/7/2018 • 4 minutes to read • [Edit Online](#)

In this tutorial, you learn how to automatically onboard Update, Change Tracking, and Inventory solutions for VMs to Azure Automation:

- Onboard an Azure VM
- Enable solutions
- Install and update modules
- Import the onboarding runbook
- Start the runbook

Prerequisites

To complete this tutorial, the following are required:

- Azure subscription. If you don't have one yet, you can [activate your MSDN subscriber benefits](#) or sign up for a [free account](#).
- [Automation account](#) to manage machines.
- A [virtual machine](#) to onboard.

Onboard an Azure VM

There are multiple ways to onboard machines, you can onboard the solution [from a virtual machine](#), [from browsing multiple machines from your Automation account](#), or by runbook. This tutorial walks through enabling Update Management through a runbook. To onboard Azure Virtual Machines at scale, an existing VM must be onboarded with the Change tracking or Update management solution. In this step, you onboard a virtual machine with Update management, and Change tracking.

Enable Change Tracking and Inventory

The Change Tracking and Inventory solution provides the ability to [track changes](#) and [inventory](#) on your virtual machines. In this step, you enable the solution on a virtual machine.

1. From the left menu, select **Automation Accounts**, and then select your automation account in the list.
2. Select **Inventory** under **CONFIGURATION MANAGEMENT**.
3. Select an existing Log Analytics workspace or create new. Click the **Enable** button.

The screenshot shows the Azure portal interface for an Automation Account named 'ExampleAutoAccount - Inventory'. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Configuration Management (with sub-options like Inventory, Change tracking, DSC nodes, DSC configurations, and DSC configurations gallery), and a search bar. The main content area is titled 'Change Tracking and Inventory' and includes a brief description: 'Enable consistent control and compliance of your VMs with Change Tracking and Inventory.' It also links to 'Learn more about pricing for these capabilities.' A large blue 'Enable' button is prominent. Below it, the 'Settings' section allows configuration of the Log Analytics workspace (set to 'East US') and the Automation account (set to 'ExampleAutoAccount').

When the change tracking and inventory solution onboarding notification completes, click on **Update Management** under **CONFIGURATION MANAGEMENT**.

Enable Update Management

The Update Management solution allows you to manage updates and patches for your Azure Windows VMs. You can assess the status of available updates, schedule installation of required updates, and review deployment results to verify updates were applied successfully to the VM. In this step, you enable the solution for your VM.

1. From your Automation Account, select **Update management** under **UPDATE MANAGEMENT**.
2. The Log analytics workspace selected is the same workspace used in the preceding step. Click **Enable** to onboard the Update management solution.

Home > ExampleAutoAccount - Update management

While the Update management solution is being installed, a blue banner is shown. When the solution is enabled select **Change tracking** under **CONFIGURATION MANAGEMENT** to go to the next step.

Select Azure VM to be managed

Now that the solutions are enabled, you can add an Azure VM to onboard to those solutions.

1. From your Automation Account, on the **Change tracking** page, select **+ Add Azure VM** to add your virtual machine.
2. Select your VM from the list and select **Enable**. This action enables the Change tracking and Inventory solution for the virtual machine.

3. When the VM onboarding notification completes, from your Automation Account select **Update management** under **UPDATE MANAGEMENT**.
4. Select **+ Add Azure VM** to add your virtual machine.
5. Select your VM from the list and select **Enable**. This action enables the Update management solution for the virtual machine.

NOTE

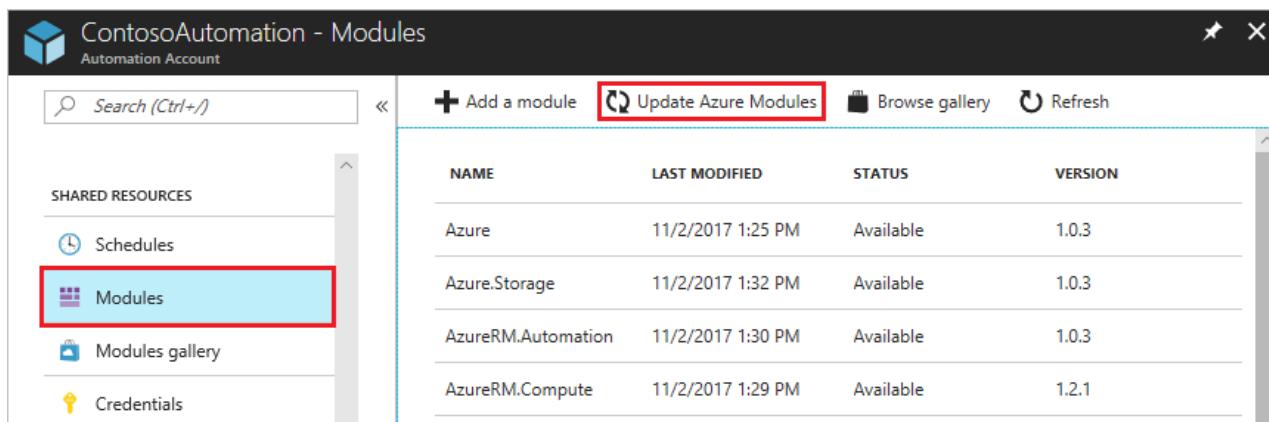
If you do not wait for the other solution to complete, when Enabling the next solution you receive a message stating:
Installation of another solution is in progress on this or a different virtual machine. When that installation completes the Enable button is enabled, and you can request installation of the solution on this virtual machine.

Install and update modules

It is required to update to the latest Azure modules and import `AzureRM.OperationalInsights` to successfully automate solution onboarding.

Update Azure Modules

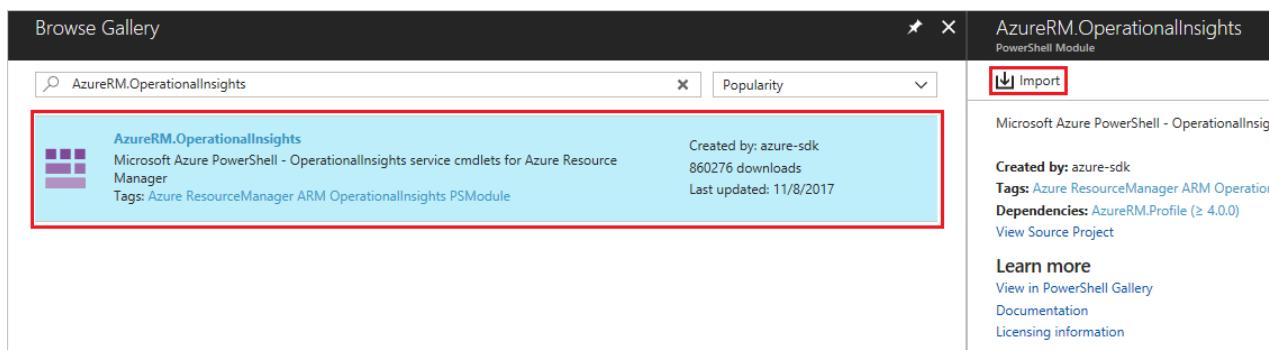
From your Automation Account, select **Modules** under **SHARED RESOURCES**. Select **Update Azure Modules** to update the Azure modules to the latest version. Select **Yes** on the prompt to update all existing Azure modules to the latest version.



NAME	LAST MODIFIED	STATUS	VERSION
Azure	11/2/2017 1:25 PM	Available	1.0.3
Azure.Storage	11/2/2017 1:32 PM	Available	1.0.3
AzureRM.Automation	11/2/2017 1:30 PM	Available	1.0.3
AzureRM.Compute	11/2/2017 1:29 PM	Available	1.2.1

Install AzureRM.OperationalInsights module

From the **Modules** page, select **Browse gallery** to open up the module gallery. Search for `AzureRM.OperationalInsights` and import this module into the Automation account.



Import the onboarding runbook

1. From your Automation Account, select on **Runbooks** under the **PROCESS AUTOMATION**.
2. Select **Browse gallery**.
3. Search for **update and change tracking**, click the runbook and select **Import** on the **View Source** page.
Select **OK** to import the runbook into the Automation account.

Browse Gallery

Filter: update and change tracking

Popularity: 29 downloads

Onboard Azure VMs to update and change tracking at scale
PowerShell Runbook
This sample automation runbook onboards Azure VMs for either the Update or ChangeTracking (which includes Inventory) solution. It requires an existing Azure VM to already be onboarded to the solution.
Tags: inventory, Windows Update, Windows Azure Virtual Machines

Created by: SC Automation Product Team
Last updated: 11/30/2017

Onboard Azure VMs to update and change tracking

This sample automation runbook onboards Azure VMs for either the Update or ChangeTracking (which includes Inventory) solution. It requires an existing Azure VM to already be onboarded to the solution. It needs to be run from the Automation account. It depends on the Log Analytics workspace.

4. On the **Runbook** page, select **Edit**, then select **Publish**. On the **Publish Runbook** dialog, select **Yes** to publish the runbook.

Start the runbook

You must have onboarded either change tracking or update solutions to an Azure VM in order to start this runbook. It requires an existing virtual machine and resource group with the solution onboarded for parameters.

1. Open the Enable-MultipleSolution runbook.

Home > ContosoFinance - Runbooks > Browse Gallery > Enable-MultipleSolution

Enable-MultipleSolution
Runbook

Search (Ctrl+ /)

Start View Edit Schedule Webhook Delete Export Refresh

Essentials
Resource group: Contoso
Account: ContosoFinance
Location: West Europe
Subscription name: Contoso

Details
Jobs

Overview Activity log Tags Diagnose and solve problems

RESOURCES Jobs Schedules Webhooks

RUNBOOK SETTINGS Properties Description Logging and tracing

2. Click the start button and enter the following values for parameters.

- **VMNAME** - Leave blank. The name of an existing VM to onboard to update or change tracking solution. By leaving this value blank, all VMs in the resource group are onboarded.
- **VMRESOURCEGROUP** - The name of the resource group for the VMs to be onboarded.
- **SUBSCRIPTIONID** - Leave blank. The subscription ID of the new VM to be onboarded. If left blank, the subscription of the workspace is used. When a different subscription ID is given, the RunAs account for this automation account should be added as a contributor for this subscription also.
- **ALREADYONBOARDEDVM** - The name of the VM that was manually onboarded to either the Updates or ChangeTracking solution.
- **ALREADYONBOARDEDVMRESOURCEGROUP** - The name of the resource group that the VM is a member of.
- **SOLUTIONTYPE** - Enter **Updates** or **ChangeTracking**

Start Runbook

Enable-MultipleSolution

Parameters

VMNAME ⓘ
No value
Optional, String

* VMRESOURCEGROUP ⓘ
finance ✓
Mandatory, String

SUBSCRIPTIONID ⓘ
No value
Optional, String

* ALREADYONBOARDEDVM ⓘ
ContosoVM1 ✓
Mandatory, String

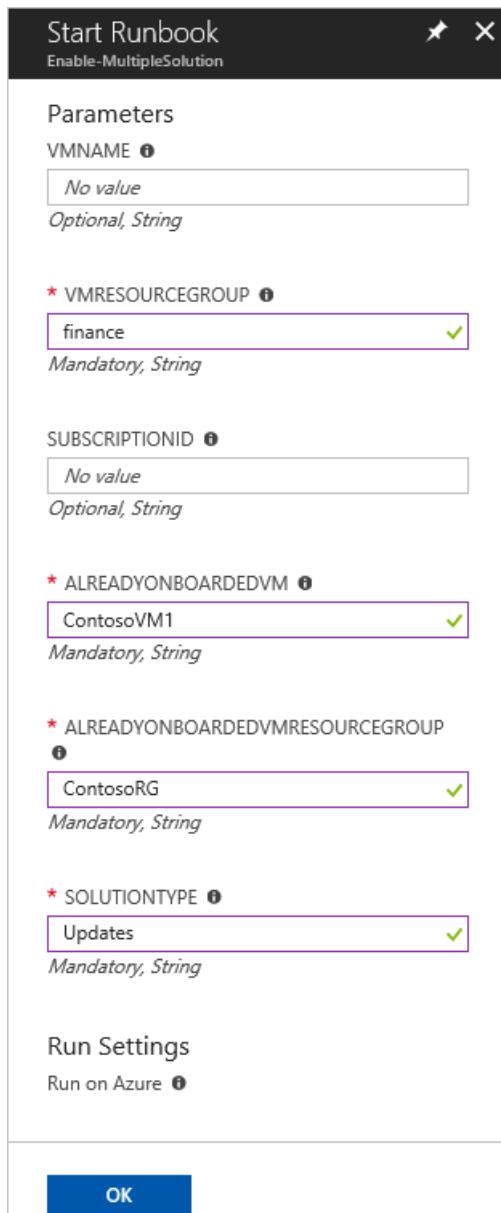
* ALREADYONBOARDEDVMRESOURCEGROUP ⓘ
ContosoRG ✓
Mandatory, String

* SOLUTIONTYPE ⓘ
Updates ✓
Mandatory, String

Run Settings

Run on Azure ⓘ

OK



3. Select **OK** to start the runbook job.
4. Monitor progress and any errors on the runbook job page.

Next steps

In this tutorial, you learned how to:

- Onboard an Azure virtual machine manually.
- Install and update required Azure modules.
- Import a runbook that onboards Azure VMs.
- Start the runbook that onboards Azure VMs automatically.

Follow this link to learn more about scheduling runbooks.

[Scheduling runbooks.](#)

Discover what software is installed on your Azure and non-Azure machines

5/10/2018 • 4 minutes to read • [Edit Online](#)

In this tutorial, you learn how to discover what software is installed in your environment. You can collect and view inventory for software, files, Linux daemons, Windows Services, and Windows Registry keys on your computers. Tracking the configurations of your machines can help you pinpoint operational issues across your environment and better understand the state of your machines.

In this tutorial you learn how to:

- Enable the solution
- Onboard an Azure VM
- Onboard a non-Azure VM
- View installed software
- Search inventory logs for installed software

Prerequisites

To complete this tutorial, you need:

- An Azure subscription. If you don't have one yet, you can [activate your MSDN subscriber benefits](#) or sign up for a [free account](#).
- An [Automation Account](#) to hold the watcher and action runbooks and the Watcher Task.
- A [virtual machine](#) to onboard.

Log in to Azure

Log in to the Azure portal at <http://portal.azure.com>.

Enable Change tracking and Inventory

First you need to enable Change tracking and Inventory for this tutorial. If you've previously enabled the **Change Tracking** solution, this step is not necessary.

Navigate to your Automation Account and select **Inventory** under **CONFIGURATION MANAGEMENT**.

Choose the Log Analytics workspace and Automation Account and click **Enable** to enable the solution. The solution takes up to 15 minutes to enable.

The screenshot shows the Azure portal interface for an Automation Account named 'ExampleAutoAccount'. The left sidebar has a search bar at the top and a navigation menu with the following items:

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems

Below this, under 'CONFIGURATION MANAGEMENT', there is a list of items:

- Inventory (highlighted in blue)
- Change tracking
- DSC nodes
- DSC configurations
- DSC configurations gallery (...)
- DSC node configurations

The main content area is titled 'Inventory' and contains the following information:

Enable consistent control and compliance of your VMs with Change Tracking and Inventory. This service is included with Azure virtual machines. You only pay for logs stored in Log Analytics. This service requires a Log Analytics workspace and this Automation account.

Location: East US

Log Analytics workspace subscription: Microsoft Azure

* Log Analytics workspace: Create New Workspace (highlighted in purple)

Automation account: ExampleAutoAccount

Enable button

To enable the solution, configure the location, Log analytics workspace, and Automation Account to use and click **Enable**. If the fields are grayed out, that means another automation solution is enabled for the VM and the same workspace and Automation Account must be used.

A [Log Analytics](#) workspace is used to collect data that is generated by features and services such as Inventory. The workspace provides a single location to review and analyze data from multiple sources.

Enabling the solution can take up to 15 minutes. During this time, you shouldn't close the browser window. After the solution is enabled, information about installed software and changes on the VM flows to Log Analytics. It can take between 30 minutes and 6 hours for the data to be available for analysis.

Onboard a VM

In your Automation Account, navigate to **Inventory** under **CONFIGURATION MANAGEMENT**.

Select **+ Add Azure VM**, this opens up the **Virtual machines** page and allows you to select an existing VM from the list. Select the VM you want to onboard. On the page that opens click **Enable** to enable the solution on the VM. The Microsoft Management Agent is deployed to the VM and configures the agent to talk to the Log Analytics workspace you configured when enabling the solution. This can take a few minutes to complete the onboarding. At this point, you can select a new VM from the list and onboard another VM.

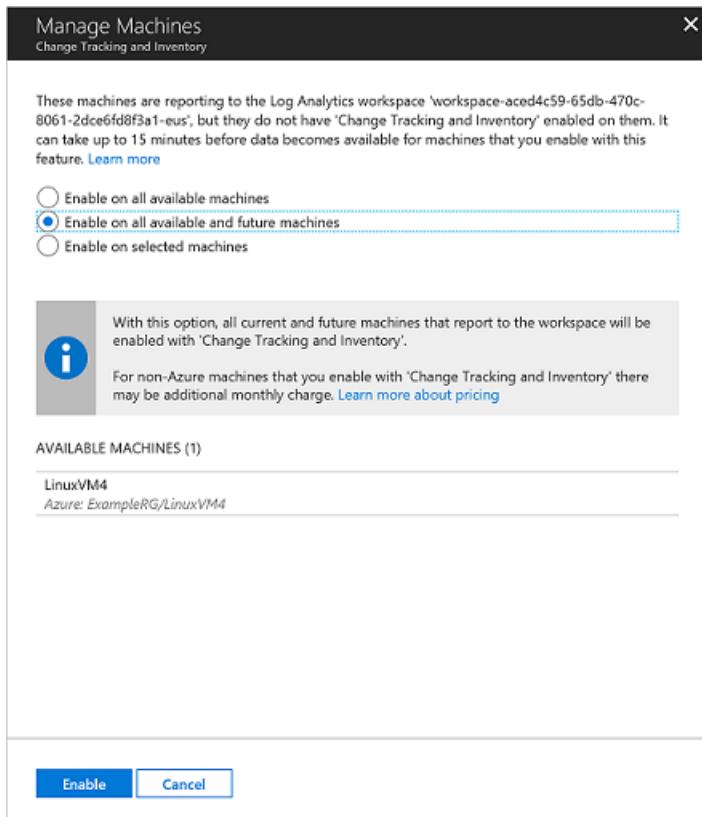
Onboard a non-Azure machine

To add non-Azure machines, install the agent for [Windows](#) or [Linux](#) depending on your operating system. Once the agent is installed, navigate to your Automation Account and go to **Inventory** under **CONFIGURATION MANAGEMENT**. When you click **Manage Machines**, you see a list of the machines reporting to your Log Analytics workspace that do not have the solution enabled. Select the appropriate option for your environment.

- **Enable on all available machines** - This option enables the solution on all the machines reporting to your Log Analytics workspace at this time.
- **Enable on all available machines and future machines** - This option enables the solution on all machines

reporting to your Log Analytics workspace and subsequently on all future machines added to the workspace.

- **Enable on selected machines** - This option enables the solution only on the machines that you have selected.



View installed software

Once the Change tracking and Inventory solution is enabled, you can view the results on the **Inventory** page.

From within your Automation Account, select **Inventory** under **CONFIGURATION MANAGEMENT**.

On the **Inventory** page, click on the **Software** tab.

On the **Software** tab, there is a table that lists the software that has been found. The software is grouped by software name and version.

The high-level details for each software record are viewable in the table. These details include the software name, version, publisher, last refreshed time (the most recent refresh time reported by a machine in the group), and machines (the count of machines with that software).

New software		Machines reporting		Learn more
0	Last 24 hours	3	Last 24 hours	Inventory
Machines Software (selected) Files Windows Registry Windows Services Linux Daemons				
<input type="text"/> Search to filter items...				
NAME	VERSION	PUBLISHER	LAST REFRESHED TIME	MACHINES
accountsservice	0.6.40-2ubuntu11.3	Ubuntu Developers <ubun...	4/11/2018 9:05 PM	1
acl	2.2.52-3	Ubuntu Developers <ubun...	4/11/2018 9:05 PM	1
acpid	1:2.0.26-1ubuntu2	Ubuntu Developers <ubun...	4/11/2018 9:05 PM	1
adduser	3.113+nmu3ubuntu4	Ubuntu Core Developers...	4/11/2018 9:05 PM	1
apparmor	2.10.95-0ubuntu2.9	Ubuntu Developers <ubun...	4/11/2018 9:05 PM	1
apport	2.20.1-0ubuntu2.15	Martin Pitt <martin.pitt@...	4/11/2018 9:05 PM	1
apport-symptoms	0.20	Ubuntu Developers <ubun...	4/11/2018 9:05 PM	1
apt	1.2.26	Ubuntu Developers <ubun...	4/11/2018 9:05 PM	1
apt-transport-https	1.2.26	Ubuntu Developers <ubun...	4/11/2018 9:05 PM	1

Click on a row to view the properties of the software record and the names of the machines with that software.

To look for a specific software or group of software, you can search in the text box directly above the software list. The filter allows you to search based off the software name, version, or publisher.

For instance, searching for "Contoso" returns all software with a name, publisher, or version containing "Contoso".

Search inventory logs for installed software

Inventory generates log data that is sent to Log Analytics. To search the logs by running queries, select **Log Analytics** at the top of the **Inventory** window.

Inventory data is stored under the type **ConfigurationData**. The following sample Log Analytics query returns the inventory results where the Publisher equals "Microsoft Corporation".

```
ConfigurationData
| where ConfigDataType == "Software"
| where Publisher == "Microsoft Corporation"
| summarize arg_max(TimeGenerated, *) by SoftwareName, Computer
```

To learn more about running and searching log files in Log Analytics, see [Azure Log Analytics](#).

Single machine inventory

To see the software inventory for a single machine, you can access Inventory from the Azure VM resource page or use Log Analytics to filter down to the corresponding machine. The following example Log Analytics query returns the list of software for a machine named ContosoVM.

```
ConfigurationData
| where ConfigDataType == "Software"
| summarize arg_max(TimeGenerated, *) by SoftwareName, CurrentVersion
| where Computer == "ContosoVM"
| render table
| summarize by Publisher, SoftwareName
```

Next steps

In this tutorial you learned how view software inventory such as how to:

- Enable the solution
- Onboard an Azure VM
- Onboard a non-Azure VM
- View installed software
- Search inventory logs for installed software

Continue to the overview for the Change tracking and Inventory solution to learn more about it.

[Change management and Inventory solution](#)

Troubleshoot changes in your environment

7/6/2018 • 6 minutes to read • [Edit Online](#)

In this tutorial, you learn how to troubleshoot changes on an Azure virtual machine. By enabling Change tracking, you can track changes to software, files, Linux daemons, Windows Services, and Windows Registry keys on your computers. Identifying these configuration changes can help you pinpoint operational issues across your environment.

In this tutorial you learn how to:

- Onboard a VM for Change tracking and Inventory
- Search change logs for stopped services
- Configure change tracking
- Enable Activity log connection
- Trigger an event
- View changes

Prerequisites

To complete this tutorial, you need:

- An Azure subscription. If you don't have one yet, you can [activate your MSDN subscriber benefits](#) or sign up for a [free account](#).
- An [Automation account](#) to hold the watcher and action runbooks and the Watcher Task.
- A [virtual machine](#) to onboard.

Log in to Azure

Log in to the Azure portal at <http://portal.azure.com>.

Enable Change tracking and Inventory

First you need to enable Change tracking and Inventory for your VM for this tutorial. If you have previously enabled another automation solution for a VM, this step is not necessary.

1. On the left menu, select **Virtual machines** and select a VM from the list
2. On the left menu, under the **OPERATIONS** section, click **Inventory**. The **Change tracking** page opens.

The screenshot shows the Azure portal interface for a virtual machine named "ContosoVM4 - Inventory". On the left, there's a sidebar with a search bar and several service icons: Automation script, Auto-shutdown, Backup, Disaster recovery (Preview), Update management, Inventory (which is selected and highlighted in blue), Change tracking, and State configuration (Preview). Below these are sections for Operations and Monitoring, each with a Metrics icon. The main content area is titled "Inventory" and contains a summary: "Enable consistent control and compliance of this VM with Change Tracking and Inventory. This service is included with Azure virtual machines. You only pay for logs stored in Log Analytics. This service requires a Log Analytics workspace and an Automation account. You can use your existing workspace and account or let us configure the nearest workspace and account for use." It includes dropdowns for "Location" (set to "East US"), "Log Analytics workspace" (set to "defaultworkspace-147a22e9-2356-4e56-..."), and "Automation account" (set to "Automate-147a22e9-2356-4e56-b3de-1f..."). A large blue "Enable" button is at the bottom.

The **Change Tracking** screen opens. Configure the location, Log analytics workspace, and Automation account to use and click **Enable**. If the fields are grayed out, that means another automation solution is enabled for the VM and the same workspace and Automation account must be used.

A [Log Analytics](#) workspace is used to collect data that is generated by features and services such as Inventory. The workspace provides a single location to review and analyze data from multiple sources.

During onboarding the VM is provisioned with the Microsoft Monitoring Agent (MMA) and hybrid worker. This agent is used to communicate with the VM and obtain information about installed software.

Enabling the solution can take up to 15 minutes. During this time, you shouldn't close the browser window. After the solution is enabled, information about installed software and changes on the VM flows to Log Analytics. It can take between 30 minutes and 6 hours for the data to be available for analysis.

Using Change tracking in Log Analytics

Change tracking generates log data that is sent to Log Analytics. To search the logs by running queries, select **Log Analytics** at the top of the **Change tracking** window. Change tracking data is stored under the type **ConfigurationChange**. The following sample Log Analytics query returns all the Windows Services that have been stopped.

```
ConfigurationChange  
| where ConfigChangeType == "WindowsServices" and SvcState == "Stopped"
```

To learn more about running and searching log files in Log Analytics, see [Azure Log Analytics](#).

Configure Change tracking

Change tracking gives you the ability to track configuration changes on your VM. The following steps show you how to configure tracking of registry keys and files.

To choose which files and Registry keys to collect and track, select **Edit settings** at the top of the **Change tracking** page.

NOTE

Inventory and Change tracking use the same collection settings, and settings are configured on a workspace level.

In the **Workspace Configuration** window, add the Windows Registry keys, Windows files, or Linux files to be tracked, as outlined in the next three sections.

Add a Windows Registry key

1. On the **Windows Registry** tab, select **Add**. The **Add Windows Registry for Change Tracking** window opens.
2. On the **Add Windows Registry for Change Tracking**, enter the information for the key to track and click **Save**

PROPERTY	DESCRIPTION
Enabled	Determines if the setting is applied
Item Name	Friendly name of the file to be tracked
Group	A group name for logically grouping files
Windows Registry Key	The path to check for the file For example: "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders\Common Startup"

Add a Windows file

1. On the **Windows Files** tab, select **Add**. The **Add Windows File for Change Tracking** window opens.
2. On the **Add Windows File for Change Tracking**, enter the information for the file or directory to track and click **Save**

PROPERTY	DESCRIPTION
Enabled	Determines if the setting is applied
Item Name	Friendly name of the file to be tracked
Group	A group name for logically grouping files
Enter Path	The path to check for the file For example: "c:\temp\myfile.txt"
Upload file content for all settings	Turns on or off file content upload on tracked changes. Available options: True or False .

Add a Linux file

1. On the **Linux Files** tab, select **Add**. The **Add Linux File for Change Tracking** window opens.
2. On the **Add Linux File for Change Tracking**, enter the information for the file or directory to track and click **Save**

PROPERTY	DESCRIPTION
Enabled	Determines if the setting is applied
Item Name	Friendly name of the file to be tracked
Group	A group name for logically grouping files
Enter Path	The path to check for the file For example: "/etc/*.conf"
Path Type	Type of item to be tracked, possible values are File and Directory
Recursion	Determines if recursion is used when looking for the item to be tracked.
Use Sudo	This setting determines if sudo is used when checking for the item.
Links	This setting determines how symbolic links dealt with when traversing directories. Ignore - Ignores symbolic links and does not include the files/directories referenced Follow - Follows the symbolic links during recursion and also includes the files/directories referenced Manage - Follows the symbolic links and allows alter the treatment of returned content
Upload file content for all settings	Turns on or off file content upload on tracked changes. Available options: True or False .

NOTE

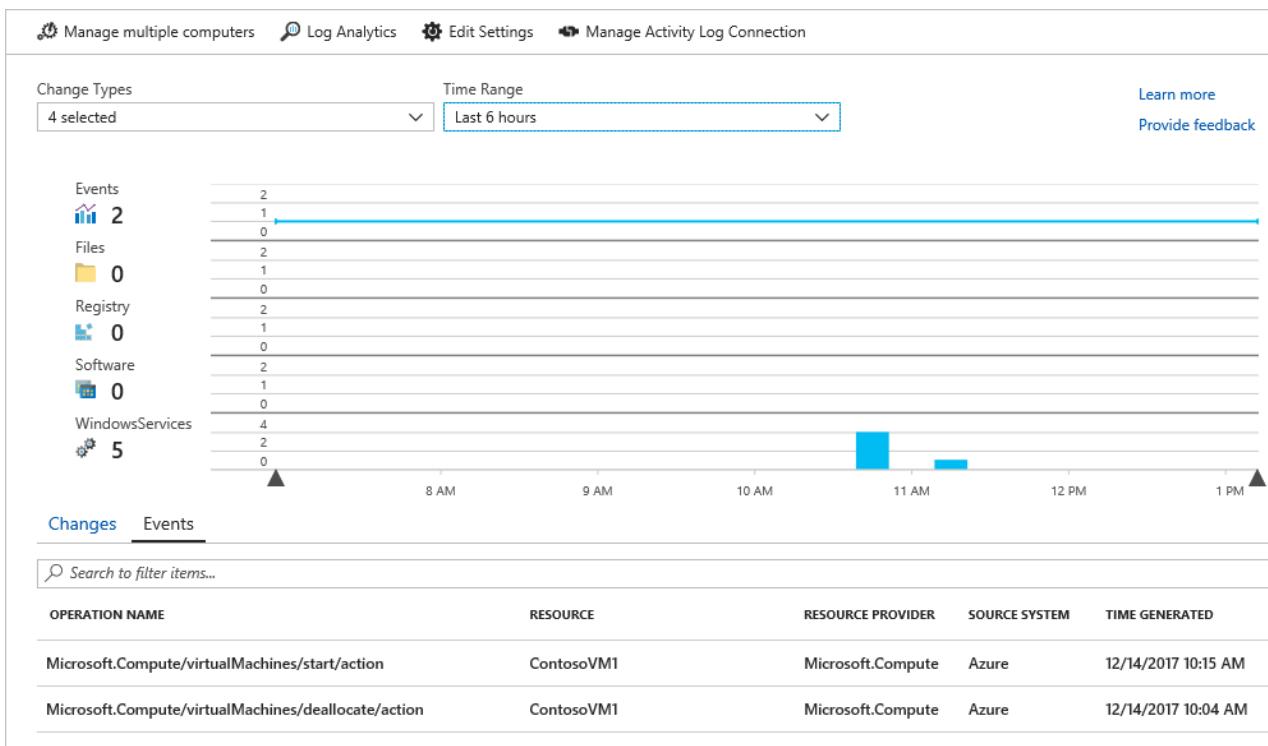
The "Manage" links option is not recommended. File content retrieval is not supported.

Enable Activity log connection

From the **Change tracking** page on your VM, select **Manage Activity Log Connection**. This task opens the **Azure Activity log** page. Select **Connect** to connect Change tracking to the Azure activity log for your VM.

With this setting enabled, navigate to the **Overview** page for your VM and select **Stop** to stop your VM. When prompted, select **Yes** to stop the VM. When it is deallocated, select **Start** to restart your VM.

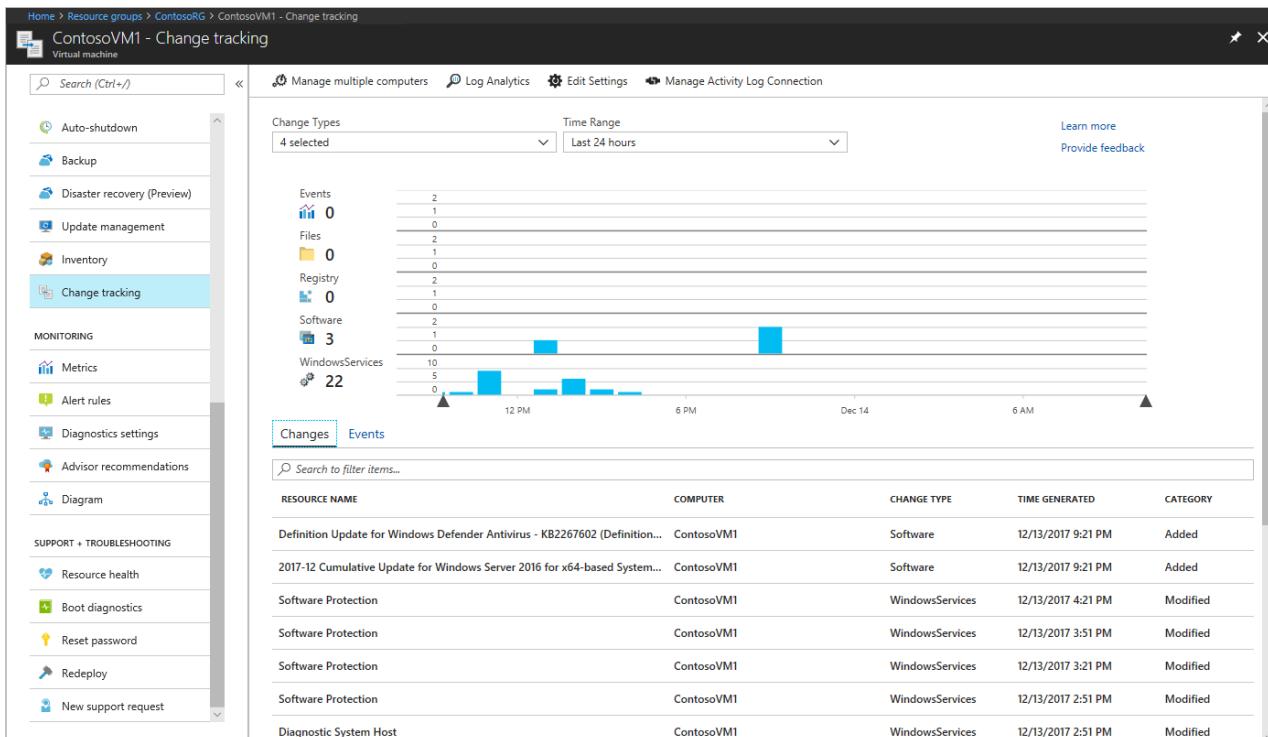
Stopping and starting a VM logs an event in its activity log. Navigate back to the **Change tracking** page. Select the **Events** tab at the bottom of the page. After a while, the events shown in the chart and the table. Like in the preceding step, each event can be selected to view detailed information on the event.



View changes

Once the Change tracking and Inventory solution is enabled, you can view the results on the **Change tracking** page.

From within your VM, select **Change tracking** under **OPERATIONS**.



The chart shows changes that have occurred over time. After you have added an Activity Log connection, the line graph at the top displays Azure Activity Log events. Each row of bar graphs represents a different trackable Change type. These types are Linux daemons, files, Windows Registry keys, software, and Windows services. The change tab shows the details for the changes shown in the visualization in descending order of time that the change occurred (most recent first). The **Events** tab, the table displays the connected Activity Log events and their corresponding details with the most recent first.

You can see in the results, that there were multiple changes to the system, including changes to services and

software. You can use the filters at the top of the page to filter the results by **Change type** or by a time range.

Select a **WindowsServices** change, this opens the **Change Details** window. The change details window shows details about the change and the values before and after the change. In this instance, the Software Protection service was stopped.

Software Protection		
Change Details		
 Computers with this resource		
PROPERTY	VALUE BEFORE	VALUE AFTER
SvcState	Running	Stopped
▼ UNCHANGED PROPERTY...		
SourceComputerId		No Change
SvcDisplayName	Software Protection	No Change
SvcName	sppsvc	No Change
SvcStartupType	Auto	No Change
SvcAccount	NT AUTHORITY\NetworkService	No Change
SvcPath	C:\Windows\system32\sppsvc.exe	No Change
SourceSystem	OpsManager	No Change
MG	00000000-0000-0000-0000-000000000001	No Change
ManagementGroup...	AOI-30862e74	No Change
TenantId		No Change
VMUUID		No Change

Next Steps

In this tutorial you learned how to:

- Onboard a VM for Change tracking and Inventory
- Search change logs for stopped services
- Configure change tracking
- Enable Activity log connection
- Trigger an event
- View changes

Continue to the overview for the Change tracking and Inventory solution to learn more about it.

[Change management and Inventory solution](#)

Enable Update Management, Change Tracking, and Inventory solutions on multiple VMs

6/8/2018 • 3 minutes to read • [Edit Online](#)

Azure Automation provides solutions to manage operating system security updates, track changes, and inventory what is installed on your computers. There are multiple ways to onboard machines, you can onboard the solution from a virtual machine, from your [Automation account](#), when browsing virtual machines, or by [runbook](#). This article covers onboarding these solutions when browsing virtual machines in Azure.

Log in to Azure

Log in to Azure at <https://portal.azure.com>

Enable solutions

In the Azure portal, navigate to **Virtual machines**.

Using the checkboxes, select the virtual machines you wish to onboard with Change Tracking and Inventory or Update Management. Onboarding is available for up to three different resource groups at a time.

NAME	TYPE	STATUS	RESOURCE GROUP	LOCATION	MAINTENANCE	SUBSCRIPTION
LinuxVM1	Virtual machine	Running	myResourceGroup	East US	Not scheduled	Microsoft Azure
LinuxVM2	Virtual machine	Running	myresourcegroup	East US	Not scheduled	Microsoft Azure
LinuxVM3	Virtual machine	Running	myResourceGroup	East US	Not scheduled	Microsoft Azure
LinuxVM4	Virtual machine	Running	myResourceGroup2	East US	Not scheduled	Microsoft Azure
LinuxVM5	Virtual machine	Running	myResourceGroup2	East US	Not scheduled	Microsoft Azure
LinuxVM6	Virtual machine	Running	myresourcegroup2	East US	Not scheduled	Microsoft Azure
WinVM1	Virtual machine	Running	myResourceGroup	East US	Not scheduled	Microsoft Azure
WinVM2	Virtual machine	Running	myresourcegroup	East US	Not scheduled	Microsoft Azure
WinVM3	Virtual machine	Running	myResourceGroup	East US	Not scheduled	Microsoft Azure
WinVM4	Virtual machine	Running	myResourceGroup	East US	Not scheduled	Microsoft Azure
WinVM5	Virtual machine	Running	myResourceGroup	East US	Not scheduled	Microsoft Azure
WinVM6	Virtual machine	Running	myresourcegroup2	East US	Not scheduled	Microsoft Azure

TIP

Use the filter controls to modify the list of virtual machines and then click the top checkbox to select all virtual machines in the list.

From the command bar, click **Services** and select either **Change tracking**, **Inventory**, or **Update Management**.

NOTE

Change tracking and **Inventory** use the same solution, when one is enabled the other is enabled as well.

The following image is for Update Management. Change tracking and Inventory have the same layout and behavior.

The list of virtual machines is filtered to show only the virtual machines that are in the same subscription and location. If your virtual machines are in more than three resource groups, the first three resource groups are selected.

Use the filter controls to select virtual machines from different subscriptions, locations, and resource groups.

Enable Update Management

Update Management

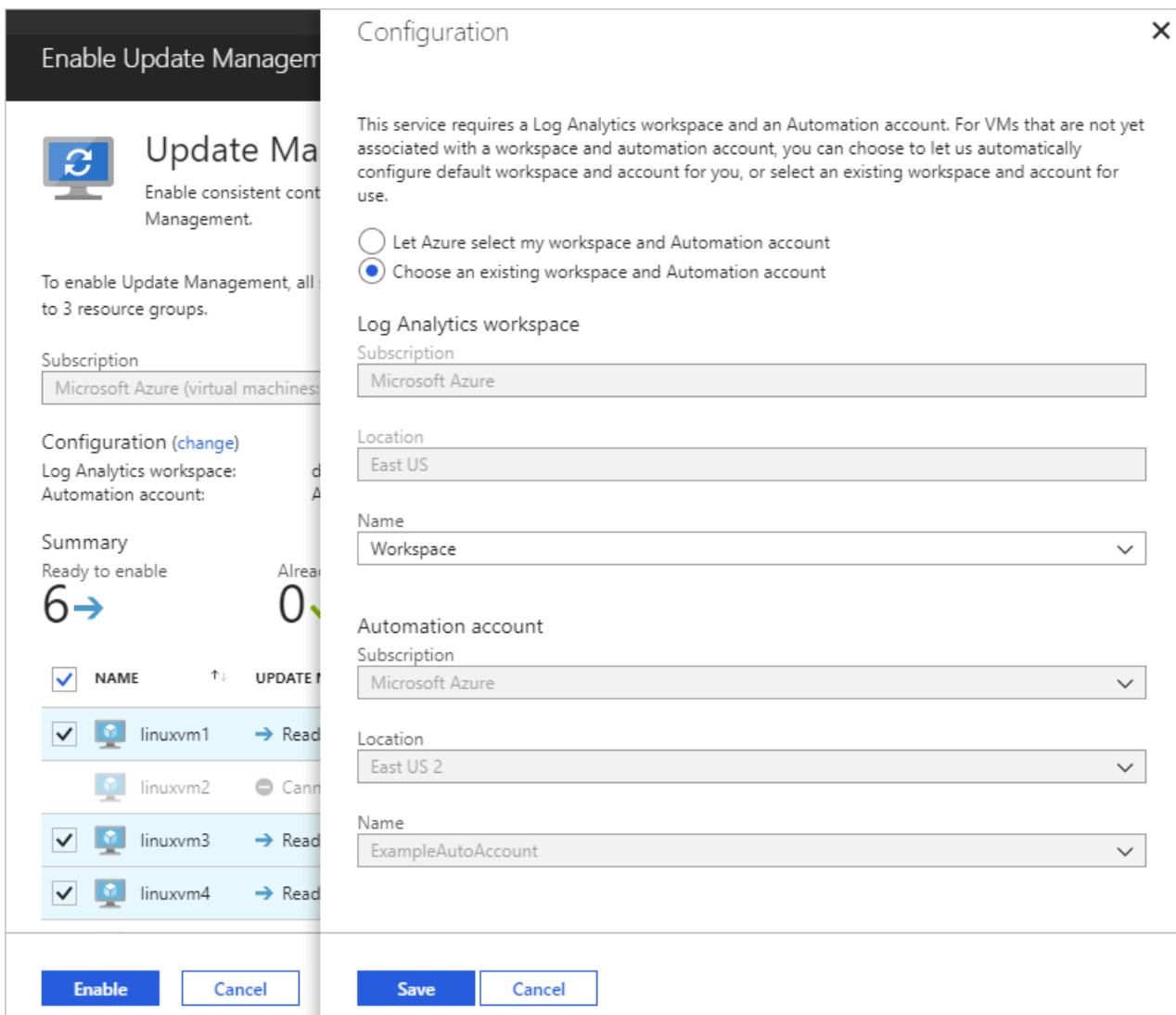
Enable consistent control and compliance of these virtual machines with Update Management.

Learn more
[Update Management](#)
[Update Management pricing](#)
[Troubleshooting](#)

VM Name	Status	Details
aks-agentpool-4260...	✓ Already enabled	Go to Update Management (Account: Automate-147a22e9-2356-4e56-b3de-1f5842ae4a3b-EUS)
aks-agentpool-4260...	ⓘ Cannot enable	VM is stopped. Start the VM
linuxvm1	ⓘ Cannot enable	VM reports to workspace: 'defaultworkspace-147a22e9-2356-4e56-b3de-1f5842ae4a3b-eus'. Use as configuration
linuxvm2	✓ Already enabled	Go to Update Management (Account: TestAzureAuto)
linuxvm3	ⓘ Cannot enable	VM reports to workspace: 'defaultworkspace-147a22e9-2356-4e56-b3de-1f5842ae4a3b-eus'. Use as configuration
linuxvm4	ⓘ Cannot enable	VM reports to workspace: 'defaultworkspace-147a22e9-2356-4e56-b3de-1f5842ae4a3b-eus'. Use as configuration
linuxvm5	✓ Already enabled	Go to Update Management (Account: ExampleAutoAccount)
linuxvm6	✓ Already enabled	Go to Update Management (Account: TestAzureAuto)
winvm1	ⓘ Cannot enable	VM reports to workspace: 'defaultworkspace-147a22e9-2356-4e56-b3de-1f5842ae4a3b-eus'. Use as configuration
winvm2	ⓘ Cannot enable	VM agent is missing or not responding.
<input checked="" type="checkbox"/>  winvm3	➔ Ready to enable	
<input checked="" type="checkbox"/>  winvm4	➔ Ready to enable	
 winvm5	✓ Already enabled	Go to Update Management (Account: ExampleAutoAccount)
 winvm6	✓ Already enabled	Go to Update Management (Account: TestAzureAuto)

Enable **Cancel** Number of virtual machines to enable Update Management: 2

Review the choices for the Log analytics workspace and Automation account. A new workspace and Automation Account are selected by default. If you have an existing Log Analytics workspace and Automation Account you want to use, click **change** to select them from the **Configuration** page. When done, click **Save**.



Deselect the checkbox next to any virtual machine that you don't want to enable. Virtual machines that can't be enabled are already deselected.

Click **Enable** to enable the solution. The solution takes up to 15 minutes to enable.

Troubleshooting

When onboarding multiple machines, there may be machines that show as **Cannot enable**. There are different reasons why some machines may not be enabled. The following sections show possible reasons for the **Cannot enable** state on a VM when attempting to onboard.

VM reports to a different workspace: '<workspaceName>'. Change configuration to use it for enabling

Cause: This error shows that the VM that you are trying to onboard reports to another workspace.

Solution: Click **Use as configuration** to change the targeted Automation Account and Log Analytics workspace.

VM reports to a workspace that is not available in this subscription

Cause: The workspace that the virtual machine reports to:

- Is in a different subscription, or
- No longer exists, or
- Is in a resource group you don't have access permissions to

Solution: Find the automation account associated with the workspace that the VM reports to and onboard the virtual machine by changing the scope configuration.

VM operating system version or distribution is not supported

Cause: The solution is not supported for all Linux distributions or all versions of Windows.

Solution: Refer to the [list of supported clients](#) for the solution.

Classic VMs cannot be enabled

Cause: Virtual machines that use the classic deployment model are not supported.

Solution: Migrate the virtual machine to the resource manager deployment model. To learn how to do this, see [Migrate classic deployment model resources](#).

VM is stopped. (deallocated)

Cause: The virtual machine is not in a **Running** state.

Solution: In order to onboard a VM to a solution the VM must be running. Click the **Start VM** inline link to start the VM without navigating away from the page.

Next steps

Now that the solution is enabled for your virtual machines, visit the Update Management overview article to learn how to view the update assessment for your machines.

[Update Management - View update assessment](#)

Addition tutorials on the solutions and how to use them:

- [Tutorial - Manage Updates for your VM](#)
- [Tutorial - Identify software on a VM](#)
- [Tutorial - Troubleshoot changes on a VM](#)

Onboard Update Management, Change Tracking, and Inventory solutions from an Azure virtual machine

6/20/2018 • 2 minutes to read • [Edit Online](#)

Azure Automation provides solutions to help you manage operating system security updates, track changes, and inventory what's installed on your computers. There are multiple ways to onboard machines. You can onboard the solution from a virtual machine, [from your Automation account](#), [from browsing multiple machines](#), or by using a [runbook](#). This article covers onboarding these solutions from an Azure virtual machine.

Sign in to Azure

Sign in to the Azure portal at <https://portal.azure.com>.

Enable the solutions

Go to an existing virtual machine. Under **OPERATIONS**, select **Update management**, **Inventory**, or **Change tracking**.

To enable the solution for the VM only, ensure that **Enable for this VM** is selected. To onboard multiple machines to the solution, select **Enable for VMs in this subscription**, and then select **Click to select machines to enable**. To learn how to onboard multiple machines at once, see [Onboard Update Management, Change Tracking, and Inventory solutions](#).

Select the Azure Log Analytics workspace and Automation account, and then select **Enable** to enable the solution. The solution takes up to 15 minutes to enable.

Home > Virtual machines > WinVM1 - Update management

WinVM1 - Update management

Virtual machine

Search (Ctrl+ /) <<

- Availability set
- Configuration
- Properties
- Locks
- Automation script

OPERATIONS

- Auto-shutdown
- Backup
- Disaster recovery
- Update management**
- Inventory
- Change tracking
- Run command

MONITORING

Update Management

Enable consistent control and compliance of this VM with Update Management.

This service is included with Azure virtual machines. You only pay for logs stored in Log Analytics.

This service requires a Log Analytics workspace and an Automation account. You can use your existing workspace and account or let us configure the nearest workspace and account for use.

Enable for this VM Enable for VMs in this subscription

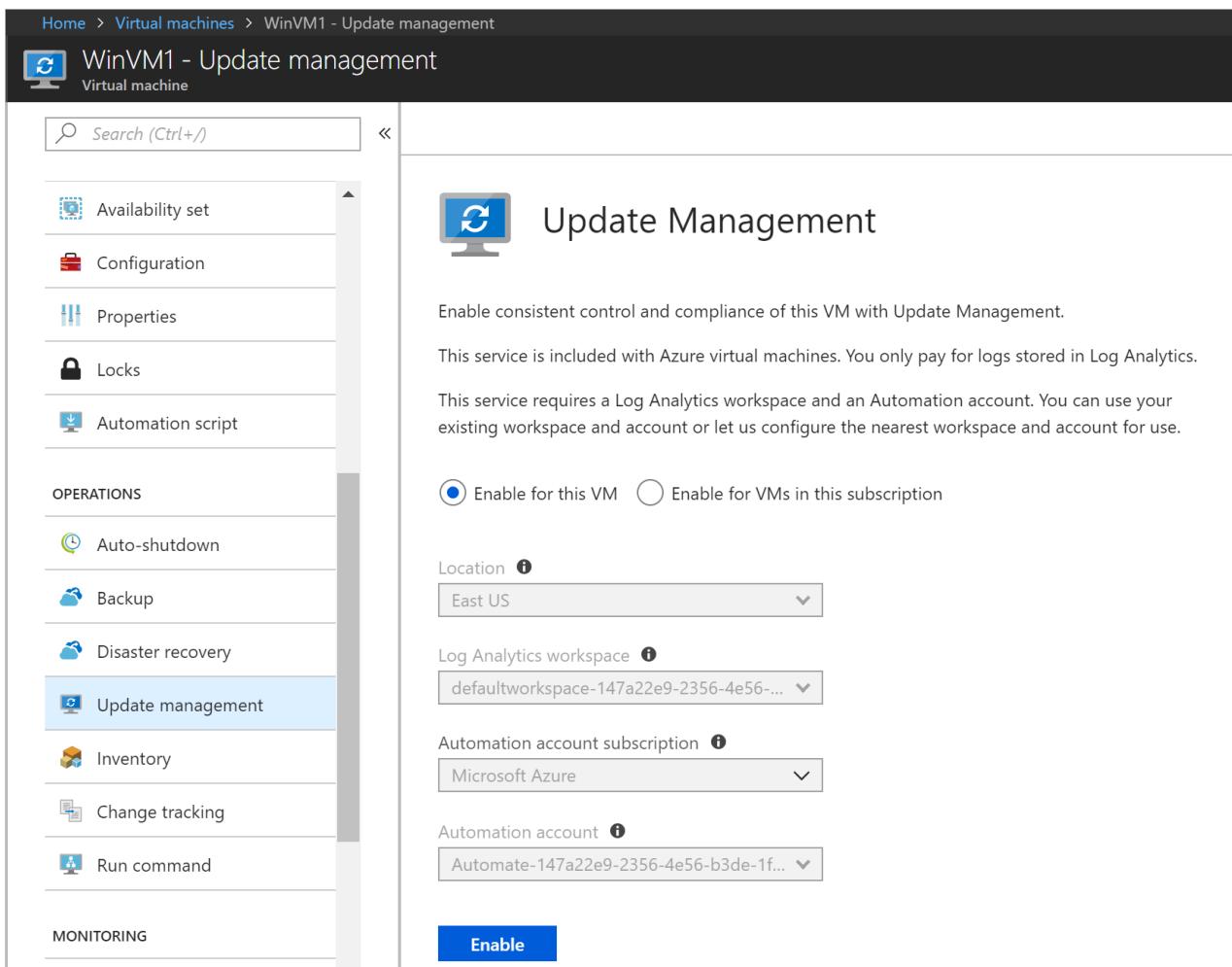
Location i
East US

Log Analytics workspace i
defaultworkspace-147a22e9-2356-4e56-... v

Automation account subscription i
Microsoft Azure v

Automation account i
Automate-147a22e9-2356-4e56-b3de-1f... v

Enable



Go to the other solutions, and then select **Enable**. The Log Analytics and Automation account drop-down lists are disabled because these solutions use the same workspace and Automation account as the previously enabled solution.

NOTE

Change tracking and **Inventory** use the same solution. When one of these solutions is enabled, the other is also enabled.

Scope configuration

Each solution uses a scope configuration in the workspace to target the computers that get the solution. The scope configuration is a group of one or more saved searches that are used to limit the scope of the solution to specific computers. To access the scope configurations, in your Automation account, under **RELATED RESOURCES**, select **Workspace**. In the workspace, under **WORKSPACE DATA SOURCES**, select **Scope Configurations**.

If the selected workspace doesn't already have the Update Management or Change Tracking solutions, the following scope configurations are created:

- **MicrosoftDefaultScopeConfig-ChangeTracking**
- **MicrosoftDefaultScopeConfig-Updates**

If the selected workspace already has the solution, the solution isn't redeployed and the scope configuration isn't added.

Select the ellipses (...) on any of the configurations, and then select **Edit**. In the **Edit scope configuration** pane, select **Select Computer Groups**. The **Computer Groups** pane shows the saved searches that are used to create the scope configuration.

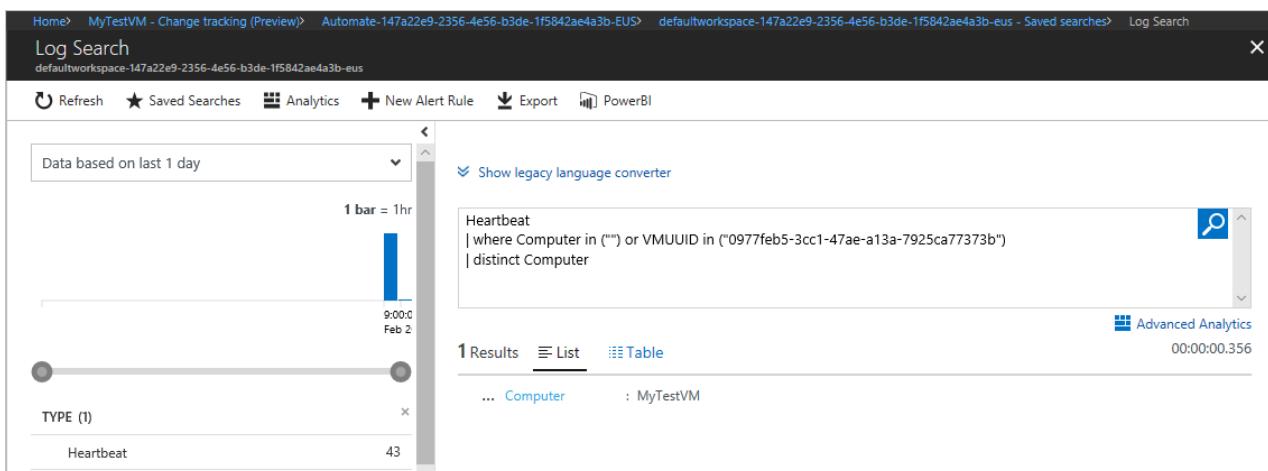
Saved searches

When a computer is added to the Update Management, Change Tracking, or Inventory solutions, the computer is added to one of two saved searches in your workspace. The saved searches are queries that contain the computers that are targeted for these solutions.

Go to your workspace. Under **General**, select **Saved searches**. The two saved searches that are used by these solutions are shown in the following table:

NAME	CATEGORY	ALIAS
MicrosoftDefaultComputerGroup	ChangeTracking	ChangeTracking__MicrosoftDefaultComputerGroup
MicrosoftDefaultComputerGroup	Updates	Updates__MicrosoftDefaultComputerGroup

Select either of the saved searches to view the query that's used to populate the group. The following image shows the query and its results:



Next steps

Continue to the tutorials for the solutions to learn how to use them:

- [Tutorial - Manage updates for your VM](#)
- [Tutorial - Identify software on a VM](#)
- [Tutorial - Troubleshoot changes on a VM](#)

Onboard Update Management, Change Tracking, and Inventory solutions

7/6/2018 • 4 minutes to read • [Edit Online](#)

Azure Automation provides solutions to manage operating system security updates, track changes, and inventory what is installed on your computers. There are multiple ways to onboard machines, you can onboard the solution [from a virtual machine](#), [from browsing multiple machines](#), from your Automation account, or by [runbook](#). This article covers onboarding these solutions from your Automation account.

Log in to Azure

Log in to Azure at <https://portal.azure.com>

Enable solutions

Navigate to your Automation account and select either **Inventory** or **Change tracking** under **CONFIGURATION MANAGEMENT**.

Choose the Log analytics workspace and Automation account and click **Enable** to enable the solution. The solution takes up to 15 minutes to enable.

The screenshot shows the Azure portal interface for enabling the Inventory solution. The top navigation bar includes 'Home', 'Resource groups', 'ExampleAutoAccount', and 'ExampleAutoAccount - Inventory'. The main content area has a sidebar with links like 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', and 'Diagnose and solve problems'. Under 'CONFIGURATION MANAGEMENT', 'Inventory' is selected and highlighted in blue. The main panel displays the 'Inventory' service details, including a description of enabling consistent control and compliance for VMs, mentioning it's included with Azure virtual machines and requires a Log Analytics workspace and Automation account. It shows the location as 'East US', the Log Analytics workspace subscription as 'Microsoft Azure', and a dropdown for 'Log Analytics workspace' currently set to 'Create New Workspace'. The 'Automation account' dropdown is set to 'ExampleAutoAccount'. A large blue 'Enable' button is at the bottom.

The Change Tracking and Inventory solution provides the ability to [track changes](#) and [inventory](#) on your virtual machines. In this step, you enable the solution on a virtual machine.

When the change tracking and inventory solution onboarding notification completes, click on **Update Management** under **CONFIGURATION MANAGEMENT**.

The Update Management solution allows you to manage updates and patches for your Azure Windows VMs. You can assess the status of available updates, schedule installation of required updates, and review deployment results to verify updates were applied successfully to the VM. This action enabled the solution for your VM.

Select **Update management** under **UPDATE MANAGEMENT**. The Log analytics workspace selected is the same workspace used in the preceding step. Click **Enable** to onboard the Update management solution. The solution takes up to 15 minutes to enable.

The screenshot shows the Azure portal interface for an 'Automation Account'. The left sidebar has a search bar and navigation links for 'CONFIGURATION MANAGEMENT' (Inventory, Change tracking, DSC nodes, DSC configurations, DSC configurations gallery, DSC node configurations), 'UPDATE MANAGEMENT' (the 'Update management' item is selected and highlighted in blue), and 'PROCESS AUTOMATION' (Runbooks, Jobs). The main content area is titled 'Update Management' and contains a sub-section titled 'Enable consistent control and compliance of your VMs with Update Management.' It states that this service is included with Azure virtual machines and requires a Log Analytics workspace and Automation account. It includes dropdown menus for 'Location' (set to 'East US'), 'Log Analytics workspace subscription' (set to 'Microsoft Azure'), 'Log Analytics workspace' (set to 'Workspace-f3aee704-2a73-4ce3-b4fb-499...'), and 'Automation account' (set to 'ExampleAutoAccount'). A large blue 'Enable' button is at the bottom.

Scope Configuration

Each solution uses a Scope Configuration within the workspace to target the computers that get the solution. The Scope Configuration is a group of one or more saved searches that is used to limit the scope of the solution to specific computers. To access the Scope Configurations, in your Automation account under **RELATED RESOURCES**, select **Workspace**. Then in the workspace under **WORKSPACE DATA SOURCES**, select **Scope Configurations**.

If the selected workspace does not have the Update Management or Change Tracking solutions yet, The following scope configurations are created:

- **MicrosoftDefaultScopeConfig-ChangeTracking**
- **MicrosoftDefaultScopeConfig-Updates**

If the selected workspace already has the solution. The solution is not re-deployed, and the scope configuration is not added to it.

Saved searches

When a computer is added to the Update Management or the Change Tracking and Inventory solutions, they are added to one of two saved searches in your workspace. These saved searches are queries that contain the computers that are targeted for these solutions.

Navigate to your Automation account and select **Saved searches** under **General**. The two saved searches used by these solutions can be seen in the following table:

NAME	CATEGORY	ALIAS
MicrosoftDefaultComputerGroup	ChangeTracking	ChangeTracking__MicrosoftDefaultComputerGroup
MicrosoftDefaultComputerGroup	Updates	Updates__MicrosoftDefaultComputerGroup

Select either saved search to view the query used to populate the group. The following image shows the query and its results:

The screenshot shows the Azure Log Search interface. The top navigation bar includes 'Home', 'ExampleAzureAuto', 'workspace-e860d6b6-5b5a-4c90-af04-8157be89a1d6-eus - Saved searches', and 'Log Search'. Below the navigation is a toolbar with 'Refresh', 'Saved Searches', 'Analytics', 'New Alert Rule', 'Export', and 'PowerBI'. A dropdown menu shows 'Data based on last 1 day' and '1 bar = 1hr'. The main search area contains the query 'Heartbeat | where Computer in ("") or VMUUID in ("") | distinct Computer'. The results section shows '0 Results' and 'List' and 'Table' options. A note at the bottom states 'The search returned no results for the specified criteria.' On the right side, there's an 'Advanced Analytics' section with a timestamp '00:00:00'.

Onboard Azure VMs

From your Automation account select **Inventory** or **Change tracking** under **CONFIGURATION MANAGEMENT**, or **Update management** under **UPDATE MANAGEMENT**.

Click **+ Add Azure VMs**, select one or more VMs from the list. Virtual machines that can't be enabled are greyed out and unable to be selected. On the **Enable Update Management** page, click **Enable**. This adds the selected VMs to the computer group saved search for the solution.

Update Management

Enable consistent control and compliance of these virtual machines with Update Management.

To enable Update Management, all selected virtual machines must be in the same subscription and location. Virtual machines can be in up to 3 resource groups.

Subscription	Location	Resource group
Microsoft Azure (virtual machines: 12)	East US (virtual machines: 12)	2 selected

Configuration

Log Analytics workspace: workspace-aced4c59-65db-470c-8061-2dce6fd8f3a1-eus
Automation account: ExampleAutoAccount

Summary

Ready to enable	Already enabled	Cannot enable
6 →	2 ✓	4 -

NAME	UPDATE MANAGEMENT	DETAILS
linuxvm1	Ready to enable	
linuxvm2	Cannot enable	VM reports to a different workspace: 'workspace-e7e3582f-a695-46cb-bc8a-b63f08f57046-eus'.
linuxvm3	Ready to enable	
linuxvm4	Ready to enable	
linuxvm5	Already enabled	
linuxvm6	Cannot enable	VM reports to a different workspace: 'workspace-e7e3582f-a695-46cb-bc8a-b63f08f57046-eus'.
winvm1	Ready to enable	
winvm2	Cannot enable	VM reports to a different workspace: 'workspace-e7e3582f-a695-46cb-bc8a-b63f08f57046-eus'.
winvm3	Ready to enable	
winvm4	Ready to enable	
winvm5	Already enabled	
winvm6	Cannot enable	VM reports to a different workspace: 'workspace-e7e3582f-a695-46cb-bc8a-b63f08f57046-eus'.

Enable **Cancel** Number of virtual machines to enable Update Management: 6

Onboard a non-Azure machine

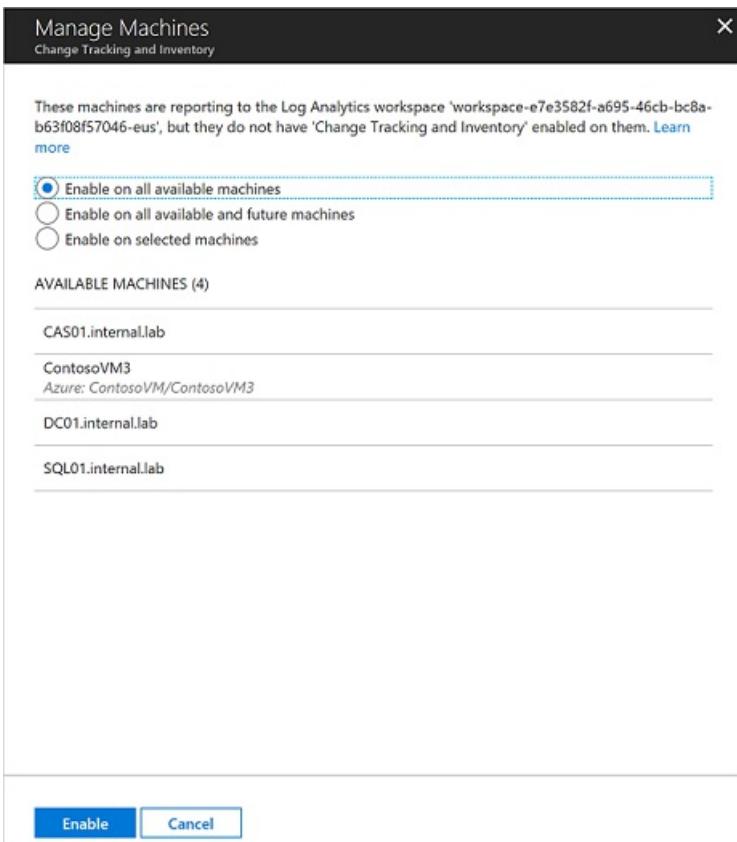
Machines not in Azure need to be added manually. From your Automation account select **Inventory** or **Change tracking** under **CONFIGURATION MANAGEMENT**, or **Update management** under **UPDATE MANAGEMENT**.

Click **Add non-Azure machine**. This opens up a new browser window with the [instructions on how to install and configure the Microsoft Monitoring Agent on the machine](#) so the machine can begin reporting to the solution. If you are onboarding a machine that currently managed by System Center Operations Manager, a new agent is not required, the workspace information is entered into the existing agent.

Onboard machines in the workspace

Manually installed machines or machines already reporting to your workspace need to be added to Azure Automation for the solution to be enabled. From your Automation account select **Inventory** or **Change tracking** under **CONFIGURATION MANAGEMENT**, or **Update management** under **UPDATE MANAGEMENT**.

Select **Manage machines**. This opens up the **Manage Machines** page. This page allows you to enable the solution on a select set of machines, all available machines, or enable the solution for all current machines and enable it on all future machines.



All available machines

To enable the solution for all available machines, select **Enable on all available machines**. This disables the control to add machines individually. This task adds all the names of the machines reporting to the workspace to the computer group saved search query.

All available and future machines

To enable the solution for all available machines and all future machines, select **Enable on all available and future machines**. This option deletes the saved searches and Scope Configurations from the workspace. This opens the solution to all Azure and non-Azure machines that are reporting to the workspace.

Selected machines

To enable the solution for one or more machines, select **Enable on selected machines** and click **add** next to each machine you want to add to the solution. This task adds the selected machine names to the computer group saved search query for the solution.

Next steps

Continue to the tutorials on the solutions to learn how to use them.

- [Tutorial - Manage Updates for your VM](#)
- [Tutorial - Identify software on a VM](#)
- [Tutorial - Troubleshoot changes on a VM](#)

Update Management solution in Azure

7/2/2018 • 22 minutes to read • [Edit Online](#)

You can use the Update Management solution in Azure Automation to manage operating system updates for your Windows and Linux computers that are deployed in Azure, in on-premises environments, or in other cloud providers. You can quickly assess the status of available updates on all agent computers and manage the process of installing required updates for servers.

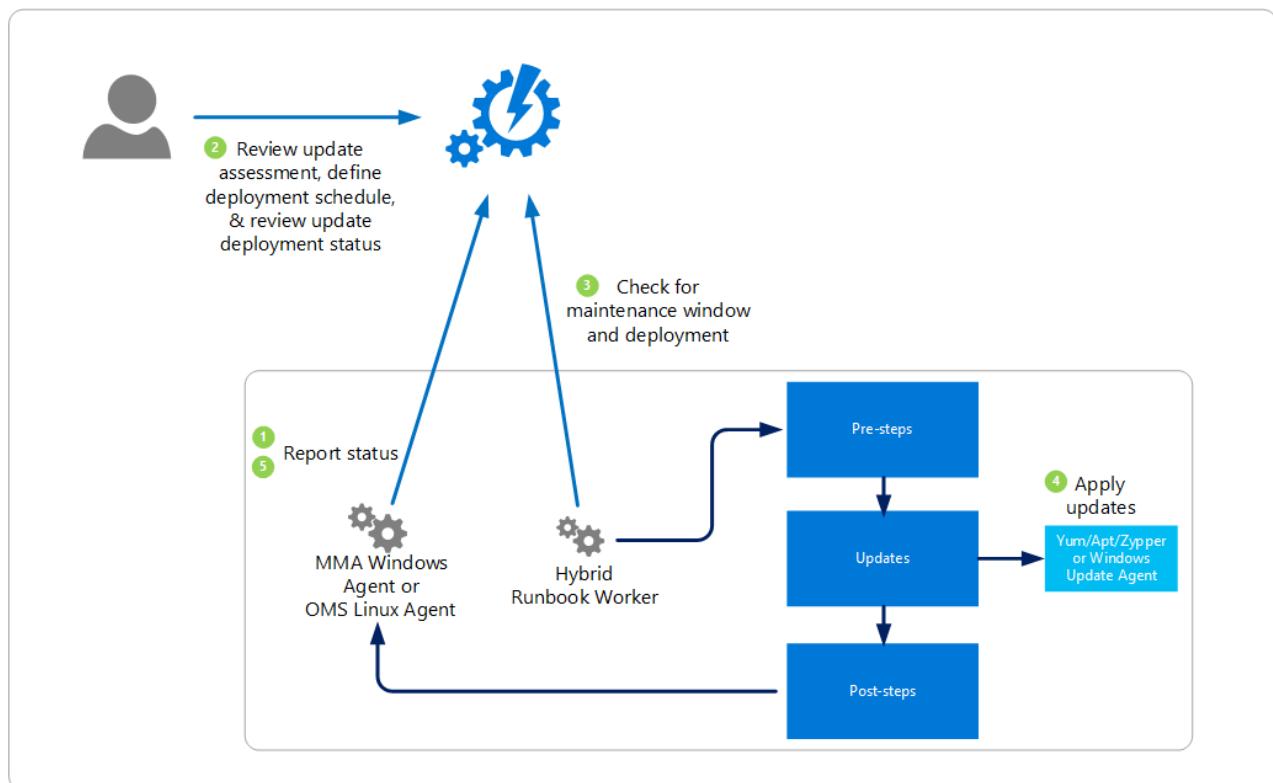
You can enable Update Management for virtual machines directly from your Azure Automation account. To learn how to enable Update Management for virtual machines from your Automation account, see [Manage updates for multiple virtual machines](#). You can also enable Update Management for a single virtual machine from the virtual machine pane in the Azure portal. This scenario is available for [Linux](#) and [Windows](#) virtual machines.

Solution overview

Computers that are managed by Update Management use the following configurations to perform assessment and update deployments:

- Microsoft Monitoring Agent (MMA) for Windows or Linux
- PowerShell Desired State Configuration (DSC) for Linux
- Automation Hybrid Runbook Worker
- Microsoft Update or Windows Server Update Services (WSUS) for Windows computers

The following diagram shows a conceptual view of the behavior and data flow with how the solution assesses and applies security updates to all connected Windows Server and Linux computers in a workspace:



After a computer performs a scan for update compliance, the agent forwards the information in bulk to Azure Log Analytics. On a Windows computer, the compliance scan is performed every 12 hours by default.

In addition to the scan schedule, the scan for update compliance is initiated within 15 minutes if the MMA is

restarted, before update installation, and after update installation.

For a Linux computer, the compliance scan is performed every 3 hours by default. If the MMA agent is restarted, a compliance scan is initiated within 15 minutes.

The solution reports how up-to-date the computer is based on what source you're configured to sync with. If the Windows computer is configured to report to WSUS, depending on when WSUS last synced with Microsoft Update, the results might differ from what Microsoft Updates shows. This is the same for Linux computers that are configured to report to a local repo instead of to a public repo.

NOTE

To properly report to the service, Update Management requires certain URLs and ports to be enabled. To learn more about these requirements, see [Network planning for Hybrid Workers](#).

You can deploy and install software updates on computers that require the updates by creating a scheduled deployment. Updates classified as *Optional* aren't included in the deployment scope for Windows computers. Only required updates are included in the deployment scope.

The scheduled deployment defines what target computers receive the applicable updates, either by explicitly specifying computers or by selecting a [computer group](#) that's based on log searches of a specific set of computers. You also specify a schedule to approve and designate a period of time during which updates can be installed.

Updates are installed by runbooks in Azure Automation. You can't view these runbooks, and the runbooks don't require any configuration. When an update deployment is created, the update deployment creates a schedule that starts a master update runbook at the specified time for the included computers. The master runbook starts a child runbook on each agent to perform installation of required updates.

At the date and time specified in the update deployment, the target computers execute the deployment in parallel. Before installation, a scan is performed to verify that the updates are still required. For WSUS client computers, if the updates aren't approved in WSUS, the update deployment fails.

Clients

Supported client types

The following table shows a list of supported operating systems:

OPERATING SYSTEM	NOTES
Windows Server 2008, Windows Server 2008 R2 RTM	Supports only update assessments.
Windows Server 2008 R2 SP1 and later	.NET Framework 4.5 or later is required. (Download .NET Framework) Windows PowerShell 4.0 or later is required. (Download WMF 4.0) Windows PowerShell 5.1 is recommended for increased reliability. (Download WMF 5.1)
CentOS 6 (x86/x64) and 7 (x64)	Linux agents must have access to an update repository. Classification-based patching requires 'yum' to return security data which CentOS does not have out of the box.
Red Hat Enterprise 6 (x86/x64) and 7 (x64)	Linux agents must have access to an update repository.
SUSE Linux Enterprise Server 11 (x86/x64) and 12 (x64)	Linux agents must have access to an update repository.

OPERATING SYSTEM	NOTES
Ubuntu 14.04 LTS and 16.04 LTS (x86/x64)	Linux agents must have access to an update repository.

Unsupported client types

The following table lists operating systems that aren't supported:

OPERATING SYSTEM	NOTES
Windows client	Client operating systems (such as Windows 7 and Windows 10) aren't supported.
Windows Server 2016 Nano Server	Not supported.

Client requirements

Windows

Windows agents must be configured to communicate with a WSUS server or they must have access to Microsoft Update. You can use Update Management with System Center Configuration Manager. To learn more about integration scenarios, see [Integrate System Center Configuration Manager with Update Management](#). The [Windows agent](#) is required. The agent is installed automatically if you're onboarding an Azure virtual machine.

Linux

For Linux, the machine must have access to an update repository. The update repository can be private or public. TLS 1.1 or TLS 1.2 is required to interact with Update Management. An Operations Management Suite (OMS) Agent for Linux that's configured to report to multiple Log Analytics workspaces isn't supported with this solution.

For information about how to install the OMS Agent for Linux and to download the latest version, see [Operations Management Suite Agent for Linux](#). For information about how to install the OMS Agent for Windows, see [Operations Management Suite Agent for Windows](#).

Permissions

To create and manage update deployments, you need specific permissions. To learn about these permissions, see [Role-based access - Update Management](#).

Solution components

The solution consists of the following resources. The resources are added to your Automation account. They're either directly connected agents or in an Operations Manager-connected management group.

Hybrid Worker groups

After you enable this solution, any Windows computer that's directly connected to your Log Analytics workspace is automatically configured as a Hybrid Runbook Worker to support the runbooks that are included in this solution.

Each Windows computer that's managed by the solution is listed in the **Hybrid worker groups** pane as a **System hybrid worker group** for the Automation account. The solutions use the naming convention *Hostname FQDN_GUID*. You can't target these groups with runbooks in your account. They fail if you try. These groups are intended to support only the management solution.

You can add the Windows computers to a Hybrid Runbook Worker group in your Automation account to support Automation runbooks if you use the same account for both the solution and the Hybrid Runbook Worker group membership. This functionality was added in version 7.2.12024.0 of the Hybrid Runbook Worker.

Management packs

If your System Center Operations Manager management group is connected to a Log Analytics workspace, the

following management packs are installed in Operations Manager. These management packs are also installed on directly connected Windows computers after you add the solution. You don't need to configure or manage these management packs.

- Microsoft System Center Advisor Update Assessment Intelligence Pack
(Microsoft.IntelligencePacks.UpdateAssessment)
- Microsoft.IntelligencePack.UpdateAssessment.Configuration
(Microsoft.IntelligencePack.UpdateAssessment.Configuration)
- Update Deployment MP

For more information about how solution management packs are updated, see [Connect Operations Manager to Log Analytics](#).

NOTE

For systems with the Operations Manger Agent, to be able to be fully managed by Update Management, the agent needs to be updated to the Microsoft Monitoring Agent. To learn how to update the agent, see [How to upgrade an Operations Manager agent](#).

Confirm that non-Azure machines are onboarded

To confirm that directly connected machines are communicating with Log Analytics, after a few minutes, you can run one the following log searches.

Linux

```
Heartbeat
| where OSType == "Linux" | summarize arg_max(TimeGenerated, *) by SourceComputerId | top 500000 by Computer
asc | render table
```

Windows

```
Heartbeat
| where OSType == "Windows" | summarize arg_max(TimeGenerated, *) by SourceComputerId | top 500000 by Computer
asc | render table
```

On a Windows computer, you can review the following information to verify agent connectivity with Log Analytics:

1. In Control Panel, open **Microsoft Monitoring Agent**. On the **Azure Log Analytics** tab, the agent displays the following message: **The Microsoft Monitoring Agent has successfully connected to Log Analytics**.
2. Open the Windows Event Log. Go to **Application and Services Logs\Operations Manager** and search for Event ID 3000 and Event ID 5002 from the source **Service Connector**. These events indicate that the computer has registered with the Log Analytics workspace and is receiving configuration.

If the agent can't communicate with Log Analytics and the agent is configured to communicate with the internet through a firewall or proxy server, confirm that the firewall or proxy server is properly configured. To learn how to verify that the firewall or proxy server is properly configured, see [Network configuration for Windows agent](#) or [Network configuration for Linux agent](#).

NOTE

If your Linux systems are configured to communicate with a proxy or OMS Gateway and you're onboarding this solution, update the *proxy.conf* permissions to grant the *omiuser* group read permission on the file by using the following commands:

```
sudo chown omsagent:omiusers /etc/opt/microsoft/omsagent/proxy.conf
sudo chmod 644 /etc/opt/microsoft/omsagent/proxy.conf
```

Newly added Linux agents show a status of **Updated** after an assessment has been performed. This process can take up to 6 hours.

To confirm that an Operations Manager management group is communicating with Log Analytics, see [Validate Operations Manager integration with Log Analytics](#).

Data collection

Supported agents

The following table describes the connected sources that are supported by this solution:

CONNECTED SOURCE	SUPPORTED	DESCRIPTION
Windows agents	Yes	The solution collects information about system updates from Windows agents and then initiates installation of required updates.
Linux agents	Yes	The solution collects information about system updates from Linux agents and then initiates installation of required updates on supported distributions.
Operations Manager management group	Yes	The solution collects information about system updates from agents in a connected management group. A direct connection from the Operations Manager agent to Log Analytics isn't required. Data is forwarded from the management group to the Log Analytics workspace.

Collection frequency

A scan is performed twice per day for each managed Windows computer. Every 15 minutes, the Windows API is called to query for the last update time to determine whether the status has changed. If the status has changed, a compliance scan is initiated.

A scan is performed every 3 hours for each managed Linux computer.

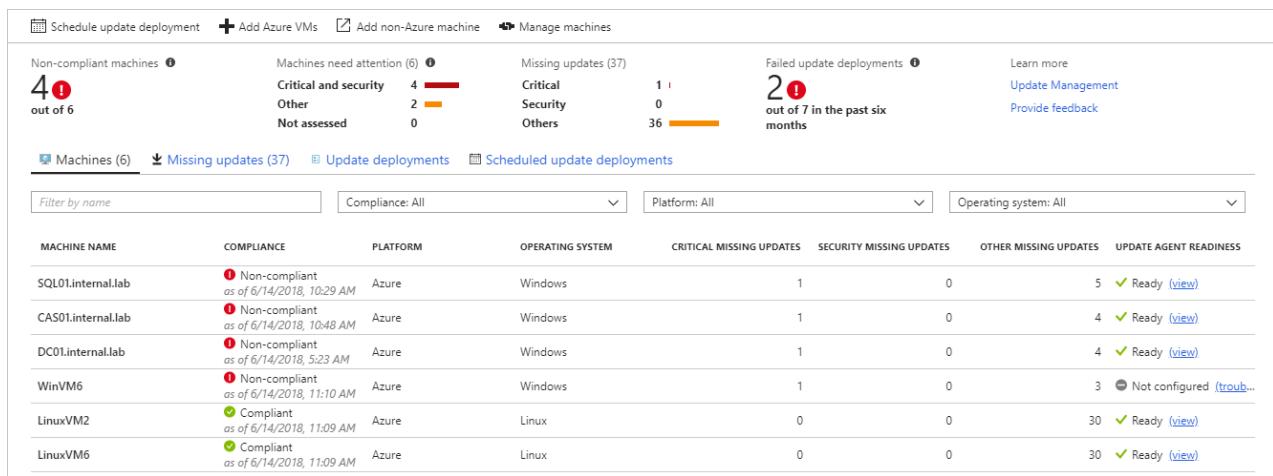
It can take between 30 minutes and 6 hours for the dashboard to display updated data from managed computers.

View update assessments

In your Automation account, select **Update Management** to view the status of your machines.

This view provides information about your machines, missing updates, update deployments, and scheduled update deployments. In the **COMPLIANCE COLUMN**, you can see the last time the machine was assessed. In the **UPDATE AGENT READINESS** column, you can see if the health of the update agent. If there's an issue, select the link to go to troubleshooting documentation that can help you learn what steps to take to correct the problem.

To run a log search that returns information about the machine, update, or deployment, select the item in the list. The **Log Search** pane opens with a query for the item selected:



Install updates

After updates are assessed for all the Linux and Windows computers in your workspace, you can install required updates by creating an *update deployment*. An update deployment is a scheduled installation of required updates for one or more computers. You specify the date and time for the deployment and a computer or group of computers to include in the scope of a deployment. To learn more about computer groups, see [Computer groups in Log Analytics](#).

When you include computer groups in your update deployment, group membership is evaluated only once, at the time of schedule creation. Subsequent changes to a group aren't reflected. To work around this, delete the scheduled update deployment and re-create it.

NOTE

Windows virtual machines that are deployed from the Azure Marketplace by default are set to receive automatic updates from Windows Update Service. This behavior doesn't change when you add this solution or add Windows virtual machines to your workspace. If you don't actively manage updates by using this solution, the default behavior (to automatically apply updates) applies.

To avoid updates being applied outside of a maintenance window on Ubuntu, reconfigure the Unattended-Upgrade package to disable automatic updates. For information about how to configure the package, see [Automatic Updates topic in the Ubuntu Server Guide](#).

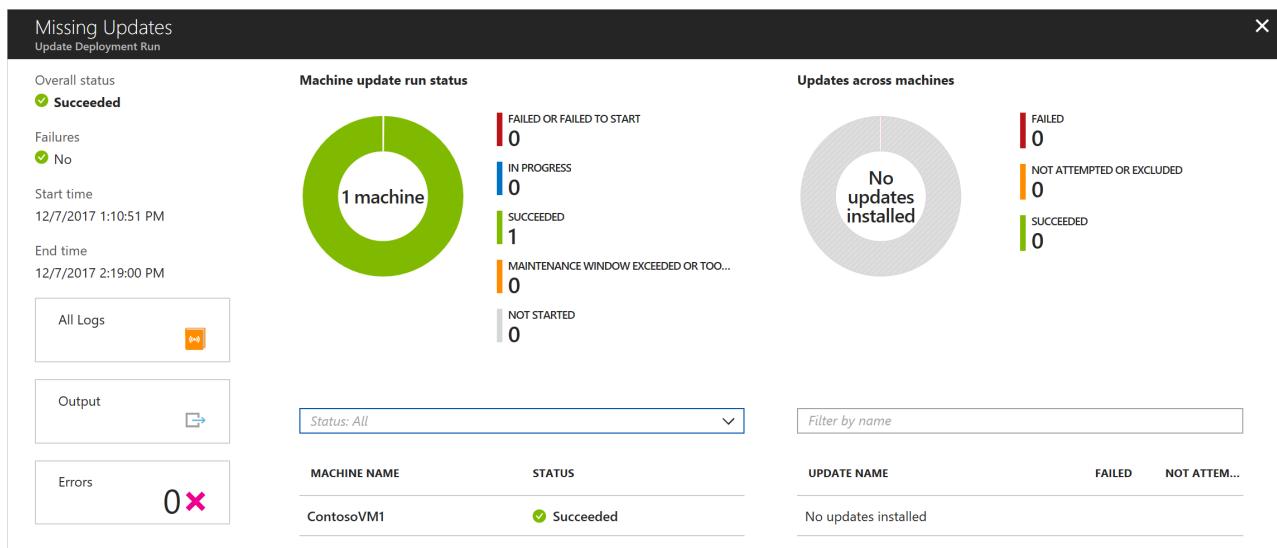
Virtual machines that were created from the on-demand Red Hat Enterprise Linux (RHEL) images that are available in the Azure Marketplace are registered to access the [Red Hat Update Infrastructure \(RHUI\)](#) that's deployed in Azure. Any other Linux distribution must be updated from the distribution's online file repository by following the distribution's supported methods.

View missing updates

Select **Missing updates** to view the list of updates that are missing from your machines. Each update is listed and can be selected. Information about the number of machines that require the update, the operating system, and a link for more information is shown. The **Log search** pane shows more details about the updates.

View update deployments

Select the **Update Deployments** tab to view the list of existing update deployments. Select any of the update deployments in the table to open the **Update Deployment Run** pane for that update deployment.



Create or edit an update deployment

To create a new update deployment, select **Schedule update deployment**. The **New Update Deployment** pane opens. Enter values for the properties described in the following table:

PROPERTY	DESCRIPTION
Name	Unique name to identify the update deployment.
Operating System	Select Linux or Windows .
Machines to update	Select a saved search, or select Machine from the drop-down list, and then select individual machines.
Update classifications	Select all the update classifications that you need. CentOS does not support this out of the box.
Updates to exclude	Enter the updates to exclude. For Windows, enter the KB article without the KB prefix. For Linux, enter the package name or use a wildcard character.
Schedule settings	Select the time to start, and then select either Once or Recurring for the recurrence.

Update classifications

The following tables list the update classifications in Update Management, with a definition for each classification.

Windows

CLASSIFICATION	DESCRIPTION
Critical updates	An update for a specific problem that addresses a critical, non-security-related bug.
Security updates	An update for a product-specific, security-related issue.
Update rollups	A cumulative set of hotfixes that are packaged together for easy deployment.

CLASSIFICATION	DESCRIPTION
Feature packs	New product features that are distributed outside a product release.
Service packs	A cumulative set of hotfixes that are applied to an application.
Definition updates	An update to virus or other definition files.
Tools	A utility or feature that helps complete one or more tasks.
Updates	An update to an application or file that currently is installed.

Linux

CLASSIFICATION	DESCRIPTION
Critical and security updates	Updates for a specific problem or a product-specific, security-related issue.
Other updates	All other updates that aren't critical in nature or aren't security updates.

For Linux, Update Management can distinguish between critical and security updates in the cloud while displaying assessment data due to data enrichment in the cloud. For patching, Update Management relies on classification data available on the machine. Unlike other distributions, CentOS does not have this information available out of the box. If you have CentOS machines configured in a way to return security data for the following command, Update Management will be able to patch based on classifications.

```
sudo yum -q --security check-update
```

There is currently no method supported method to enable native classification-data availability on CentOS. At this time, only best-effort support is provided to customers who may have enabled this on their own.

Ports

The following addresses are required specifically for Update Management. Communication to these addresses occurs over port 443.

AZURE PUBLIC	AZURE GOVERNMENT
*.ods.opinsights.azure.com	*.ods.opinsights.azure.us
*.oms.opinsights.azure.com	*.oms.opinsights.azure.us
*.blob.core.windows.net	*.blob.core.usgovcloudapi.net

For more information about ports that the Hybrid Runbook Worker requires, see [Hybrid Worker role ports](#).

It is recommended to use the addresses listed when defining exceptions. For IP addresses you can download the [Microsoft Azure Datacenter IP Ranges](#). This file is updated weekly, and reflects the currently deployed ranges and any upcoming changes to the IP ranges.

Search logs

In addition to the details that are provided in the Azure portal, you can do searches against the logs. On the solution pages, select **Log Analytics**. The **Log Search** pane opens.

You can also learn how to customize the queries or use them from different clients and more by visiting: [Log Analytics search API documentation](#).

Sample queries

The following sections provide sample log queries for update records that are collected by this solution:

Single Azure VM Assessment queries (Windows)

Replace the VMUUID value with the VM GUID of the virtual machine you are querying. You can find the VMUUID that should be used by running the following query in Log Analytics:

```
Update | where Computer == "<machine name>" | summarize by Computer, VMUUID
```

Missing updates summary

```
Update
| where TimeGenerated>ago(14h) and OSType!="Linux" and (Optional==false or Classification has "Critical" or
Classification has "Security") and VMUUID=~"b08d5afa-1471-4b52-bd95-a44fea6e4ca8"
| summarize hint.strategy=partitioned arg_max(TimeGenerated, UpdateState, Classification, Approved) by
Computer, SourceComputerId, UpdateID
| where UpdateState=~"Needed" and Approved!=false
| summarize by UpdateID, Classification
| summarize allUpdatesCount=count(), criticalUpdatesCount=countif(Classification has "Critical"),
securityUpdatesCount=countif(Classification has "Security"), otherUpdatesCount=countif(Classification !has
"Critical" and Classification !has "Security")
```

Missing updates list

```
Update
| where TimeGenerated>ago(14h) and OSType!="Linux" and (Optional==false or Classification has "Critical" or
Classification has "Security") and VMUUID=~"8bf1cccc6-b6d3-4a0b-a643-23f346dfdf82"
| summarize hint.strategy=partitioned arg_max(TimeGenerated, UpdateState, Classification, Title, KBID,
PublishedDate, Approved) by Computer, SourceComputerId, UpdateID
| where UpdateState=~"Needed" and Approved!=false
| project-away UpdateState, Approved, TimeGenerated
| summarize computersCount=dcount(SourceComputerId, 2), displayName=any(Title),
publishedDate=min(PublishedDate), ClassificationWeight=max(if(Classification has "Critical", 4,
if(Classification has "Security", 2, 1))) by id=strcat(UpdateID, "_", KBID), classification=Classification,
InformationId=strcat("KB", KBID), InformationUrl=if(isnotempty(KBID),
strcat("https://support.microsoft.com/kb/", KBID), ""), osType=2
| sort by ClassificationWeight desc, computersCount desc, displayName asc
| extend informationLink=(if(isnotempty(InformationId) and isnotempty(InformationUrl), toobject(strcat('{",
"uri": "", InformationUrl, '", "text": "", InformationId, '", "target": "blank" }'))), toobject(''))
| project-away ClassificationWeight, InformationId, InformationUrl
```

Single Azure VM assessment queries (Linux)

For some Linux distros there is a [endianness](#) mismatch with the VMUUID value that comes from Azure Resource Manager and what is stored in Log Analytics. The following query checks for a match on either endianness. Replace the VMUUID values with the big-endian and little-endian format of the GUID to properly return the results. You can find the VMUUID that should be used by running the following query in Log Analytics:

```
Update | where Computer == "<machine name>" | summarize by Computer, VMUUID
```

Missing updates summary

```

Update
| where TimeGenerated>ago(5h) and OSType=="Linux" and (VMUUID=~"625686a0-6d08-4810-aae9-a089e68d4911" or
VMUUID=~"a0865662-086d-1048-aae9-a089e68d4911")
| summarize hint.strategy=partitioned arg_max(TimeGenerated, UpdateState, Classification) by Computer,
SourceComputerId, Product, ProductArch
| where UpdateState=~"Needed"
| summarize by Product, ProductArch, Classification
| summarize allUpdatesCount=count(), criticalUpdatesCount=countif(Classification has "Critical"),
securityUpdatesCount=countif(Classification has "Security"), otherUpdatesCount=countif(Classification !has
"Critical" and Classification !has "Security")

```

Missing updates list

```

Update
| where TimeGenerated>ago(5h) and OSType=="Linux" and (VMUUID=~"625686a0-6d08-4810-aae9-a089e68d4911" or
VMUUID=~"a0865662-086d-1048-aae9-a089e68d4911")
| summarize hint.strategy=partitioned arg_max(TimeGenerated, UpdateState, Classification, BulletinUrl,
BulletinID) by Computer, SourceComputerId, Product, ProductArch
| where UpdateState=~"Needed"
| project-away UpdateState, TimeGenerated
| summarize computersCount=dcount(SourceComputerId, 2), ClassificationWeight=max(if(Classification has
"Critical", 4, iff(Classification has "Security", 2, 1))) by id=strcat(Product, "_", ProductArch),
displayName=Product, productArch=ProductArch, classification=Classification, InformationId=BulletinID,
InformationUrl=tostring(split(BulletinUrl, ";", 0)[0]), osType=1
| sort by ClassificationWeight desc, computersCount desc, displayName asc
| extend informationLink=(iff(isnotempty(InformationId) and isnotempty(InformationUrl), toobject(strcat('{',
"uri": "", InformationUrl, '", "text": "", InformationId, '", "target": "blank" }')), toobject(''))
| project-away ClassificationWeight, InformationId, InformationUrl

```

Multi-VM assessment queries

Computers summary

```

Heartbeat
| where TimeGenerated>ago(12h) and OSType=~"Windows" and notempty(Computer)
| summarize arg_max(TimeGenerated, Solutions) by SourceComputerId
| where Solutions has "updates"
| distinct SourceComputerId
| join kind=leftouter
(
    Update
    | where TimeGenerated>ago(14h) and OSType!="Linux"
    | summarize hint.strategy=partitioned arg_max(TimeGenerated, UpdateState, Approved, Optional,
Classification) by SourceComputerId, UpdateID
        | distinct SourceComputerId, Classification, UpdateState, Approved, Optional
        | summarize WorstMissingUpdateSeverity=max(if(UpdateState=~"Needed" and (Optional==false or
Classification has "Critical" or Classification has "Security") and Approved!=false, iff(Classification has
"Critical", 4, iff(Classification has "Security", 2, 1)), 0)) by SourceComputerId
    )
on SourceComputerId
| extend WorstMissingUpdateSeverity=coalesce(WorstMissingUpdateSeverity, -1)
| summarize computersBySeverity=count() by WorstMissingUpdateSeverity
| union (Heartbeat
| where TimeGenerated>ago(12h) and OSType=="Linux" and notempty(Computer)
| summarize arg_max(TimeGenerated, Solutions) by SourceComputerId
| where Solutions has "updates"
| distinct SourceComputerId
| join kind=leftouter
(
    Update
    | where TimeGenerated>ago(5h) and OSType=="Linux"
    | summarize hint.strategy=partitioned arg_max(TimeGenerated, UpdateState, Classification) by
SourceComputerId, Product, ProductArch
        | distinct SourceComputerId, Classification, UpdateState
        | summarize WorstMissingUpdateSeverity=max(if(UpdateState=~"Needed", iff(Classification has "Critical",
4, iff(Classification has "Security", 2, 1)), 0)) by SourceComputerId
    )
on SourceComputerId
| extend WorstMissingUpdateSeverity=coalesce(WorstMissingUpdateSeverity, -1)
| summarize computersBySeverity=count() by WorstMissingUpdateSeverity
| summarize assessedComputersCount=sumif(computersBySeverity, WorstMissingUpdateSeverity>-1),
notAssessedComputersCount=sumif(computersBySeverity, WorstMissingUpdateSeverity== -1),
computersNeedCriticalUpdatesCount=sumif(computersBySeverity, WorstMissingUpdateSeverity==4),
computersNeedSecurityUpdatesCount=sumif(computersBySeverity, WorstMissingUpdateSeverity==2),
computersNeededOtherUpdatesCount=sumif(computersBySeverity, WorstMissingUpdateSeverity==1),
upToDateComputersCount=sumif(computersBySeverity, WorstMissingUpdateSeverity==0)
| summarize assessedComputersCount=sum(assessedComputersCount),
computersNeedCriticalUpdatesCount=sum(computersNeedCriticalUpdatesCount),
computersNeedSecurityUpdatesCount=sum(computersNeedSecurityUpdatesCount),
computersNeededOtherUpdatesCount=sum(computersNeededOtherUpdatesCount),
upToDateComputersCount=sum(upToDateComputersCount), notAssessedComputersCount=sum(notAssessedComputersCount)
| extend allComputersCount=assessedComputersCount+notAssessedComputersCount

```

Missing updates summary

```

Update
| where TimeGenerated>ago(5h) and OSType=="Linux" and SourceComputerId in ((Heartbeat
| where TimeGenerated>ago(12h) and OSType=="Linux" and notempty(Computer)
| summarize arg_max(TimeGenerated, Solutions) by SourceComputerId
| where Solutions has "updates"
| distinct SourceComputerId))
| summarize hint.strategy=partitioned arg_max(TimeGenerated, UpdateState, Classification) by Computer,
SourceComputerId, Product, ProductArch
| where UpdateState=~"Needed"
| summarize by Product, ProductArch, Classification
| union (Update
| where TimeGenerated>ago(14h) and OSType!="Linux" and (Optional==false or Classification has "Critical" or
Classification has "Security") and SourceComputerId in ((Heartbeat
| where TimeGenerated>ago(12h) and OSType=~"Windows" and notempty(Computer)
| summarize arg_max(TimeGenerated, Solutions) by SourceComputerId
| where Solutions has "updates"
| distinct SourceComputerId))
| summarize hint.strategy=partitioned arg_max(TimeGenerated, UpdateState, Classification, Approved) by
Computer, SourceComputerId, UpdateID
| where UpdateState=~"Needed" and Approved!=false
| summarize by UpdateID, Classification )
| summarize allUpdatesCount=count(), criticalUpdatesCount=countif(Classification has "Critical"),
securityUpdatesCount=countif(Classification has "Security"), otherUpdatesCount=countif(Classification !has
"Critical" and Classification !has "Security")

```

Computers list

```

Heartbeat
| where TimeGenerated>ago(12h) and OSType=="Linux" and notempty(Computer)
| summarize arg_max(TimeGenerated, Solutions, Computer, ResourceId, ComputerEnvironment, VMUUID) by
SourceComputerId
| where Solutions has "updates"
| extend vmuiId=VMUUID, azureResourceId=ResourceId, osType=1, environment=iff(ComputerEnvironment=~"Azure", 1,
2), scopedToUpdatesSolution=true, lastUpdateAgentSeenTime=""
| join kind=leftouter
(
    Update
    | where TimeGenerated>ago(5h) and OSType=="Linux" and SourceComputerId in ((Heartbeat
    | where TimeGenerated>ago(12h) and OSType=="Linux" and notempty(Computer)
    | summarize arg_max(TimeGenerated, Solutions) by SourceComputerId
    | where Solutions has "updates"
    | distinct SourceComputerId))
    | summarize hint.strategy=partitioned arg_max(TimeGenerated, UpdateState, Classification, Product,
Computer, ComputerEnvironment) by SourceComputerId, Product, ProductArch
        | summarize Computer=any(Computer), ComputerEnvironment=any(ComputerEnvironment),
missingCriticalUpdatesCount=countif(Classification has "Critical" and UpdateState=~"Needed"),
missingSecurityUpdatesCount=countif(Classification has "Security" and UpdateState=~"Needed"),
missingOtherUpdatesCount=countif(Classification !has "Critical" and Classification !has "Security" and
UpdateState=~"Needed"), lastAssessedTime=max(TimeGenerated), lastUpdateAgentSeenTime="" by SourceComputerId
            | extend compliance=iff(missingCriticalUpdatesCount > 0 or missingSecurityUpdatesCount > 0, 2, 1)
            | extend ComplianceOrder=iff(missingCriticalUpdatesCount > 0 or missingSecurityUpdatesCount > 0 or
missingOtherUpdatesCount > 0, 1, 3)
)
on SourceComputerId
| project id=SourceComputerId, displayName=Computer, sourceComputerId=SourceComputerId,
scopedToUpdatesSolution=true, missingCriticalUpdatesCount=coalesce(missingCriticalUpdatesCount, -1),
missingSecurityUpdatesCount=coalesce(missingSecurityUpdatesCount, -1),
missingOtherUpdatesCount=coalesce(missingOtherUpdatesCount, -1), compliance=coalesce(compliance, 4),
lastAssessedTime, lastUpdateAgentSeenTime, osType=1, environment=iff(ComputerEnvironment=~"Azure", 1, 2),
ComplianceOrder=coalesce(ComplianceOrder, 2)
| union(Heartbeat
| where TimeGenerated>ago(12h) and OSType=~"Windows" and notempty(Computer)
| summarize arg_max(TimeGenerated, Solutions, Computer, ResourceId, ComputerEnvironment, VMUUID) by
SourceComputerId
| where Solutions has "updates"
| extend vmuiId=VMUUID, azureResourceId=ResourceId, osType=2, environment=iff(ComputerEnvironment=~"Azure", 1,
2), scopedToUpdatesSolution=true, lastUpdateAgentSeenTime=""

```

```

| join kind=leftouter
(
    Update
    | where TimeGenerated>ago(14h) and OSType!="Linux" and SourceComputerId in ((Heartbeat
    | where TimeGenerated>ago(12h) and OSType=~"Windows" and notempty(Computer)
    | summarize arg_max(TimeGenerated, Solutions) by SourceComputerId
    | where Solutions has "updates"
    | distinct SourceComputerId))
    | summarize hint.strategy=partitioned arg_max(TimeGenerated, UpdateState, Classification, Title, Optional,
Approved, Computer, ComputerEnvironment) by Computer, SourceComputerId, UpdateID
    | summarize Computer=any(Computer), ComputerEnvironment=any(ComputerEnvironment),
missingCriticalUpdatesCount=countif(Classification has "Critical" and UpdateState=~"Needed" and
Approved!=false), missingSecurityUpdatesCount=countif(Classification has "Security" and UpdateState=~"Needed"
and Approved!=false), missingOtherUpdatesCount=countif(Classification !has "Critical" and Classification !has
"Security" and UpdateState=~"Needed" and Optional==false and Approved!=false),
lastAssessedTime=max(TimeGenerated), lastUpdateAgentSeenTime="" by SourceComputerId
    | extend compliance=iff(missingCriticalUpdatesCount > 0 or missingSecurityUpdatesCount > 0, 2, 1)
    | extend ComplianceOrder=iff(missingCriticalUpdatesCount > 0 or missingSecurityUpdatesCount > 0 or
missingOtherUpdatesCount > 0, 1, 3)
)
on SourceComputerId
| project id=SourceComputerId, displayName=Computer, sourceComputerId=SourceComputerId,
scopedToUpdatesSolution=true, missingCriticalUpdatesCount=coalesce(missingCriticalUpdatesCount, -1),
missingSecurityUpdatesCount=coalesce(missingSecurityUpdatesCount, -1),
missingOtherUpdatesCount=coalesce(missingOtherUpdatesCount, -1), compliance=coalesce(compliance, 4),
lastAssessedTime, lastUpdateAgentSeenTime, osType=2, environment=iff(ComputerEnvironment=~"Azure", 1, 2),
ComplianceOrder=coalesce(ComplianceOrder, 2) )
| order by ComplianceOrder asc, missingCriticalUpdatesCount desc, missingSecurityUpdatesCount desc,
missingOtherUpdatesCount desc, displayName asc
| project-away ComplianceOrder

```

Missing updates list

```

Update
| where TimeGenerated>ago(5h) and OSType=="Linux" and SourceComputerId in ((Heartbeat
| where TimeGenerated>ago(12h) and OSType=="Linux" and notempty(Computer)
| summarize arg_max(TimeGenerated, Solutions) by SourceComputerId
| where Solutions has "updates"
| distinct SourceComputerId))
| summarize hint.strategy=partitioned arg_max(TimeGenerated, UpdateState, Classification, BulletinUrl,
BulletinID) by SourceComputerId, Product, ProductArch
| where UpdateState=~"Needed"
| project-away UpdateState, TimeGenerated
| summarize computersCount=dcount(SourceComputerId, 2), ClassificationWeight=max(ifClassification has
"Critical", 4, iff(Classification has "Security", 2, 1)) by id=strcat(Product, "_", ProductArch),
displayName=Product, productArch=ProductArch, classification=Classification, InformationId=BulletinID,
InformationUrl=toString(split(BulletinUrl, ";", 0)[0]), osType=1
| union(Update
| where TimeGenerated>ago(14h) and OSType!="Linux" and (Optional==false or Classification has "Critical" or
Classification has "Security") and SourceComputerId in ((Heartbeat
| where TimeGenerated>ago(12h) and OSType=~"Windows" and notempty(Computer)
| summarize arg_max(TimeGenerated, Solutions) by SourceComputerId
| where Solutions has "updates"
| distinct SourceComputerId))
| summarize hint.strategy=partitioned arg_max(TimeGenerated, UpdateState, Classification, Title, KBID,
PublishedDate, Approved) by Computer, SourceComputerId, UpdateID
| where UpdateState=~"Needed" and Approved!=false
| project-away UpdateState, Approved, TimeGenerated
| summarize computersCount=dcount(SourceComputerId, 2), displayName=any(Title),
publishedDate=min(PublishedDate), ClassificationWeight=max(ifClassification has "Critical", 4,
iff(Classification has "Security", 2, 1)) by id=strcat(UpdateID, "_", KBID), classification=Classification,
InformationId=strcat("KB", KBID), InformationUrl=iff(isnotempty(KBID),
strcat("https://support.microsoft.com/kb/", KBID), ""), osType=2
| sort by ClassificationWeight desc, computersCount desc, displayName asc
| extend informationLink=(iff(isnotempty(InformationId) and isnotempty(InformationUrl), toobject(strcat('{
"uri": "", InformationUrl, '", "text": "", InformationId, '", "target": "blank" }')), toobject(')))
| project-away ClassificationWeight, InformationId, InformationUrl

```

Integrate with System Center Configuration Manager

Customers who have invested in System Center Configuration Manager for managing PCs, servers, and mobile devices also rely on the strength and maturity of Configuration Manager to help them manage software updates. Configuration Manager is part of their software update management (SUM) cycle.

To learn how to integrate the management solution with System Center Configuration Manager, see [Integrate System Center Configuration Manager with Update Management](#).

Patch Linux machines

The following sections explain potential issues with Linux patching.

Unexpected OS-level upgrades

On some Linux variants, such as Red Hat Enterprise Linux, OS-level upgrades might occur via packages. This might lead to Update Management runs where the OS version number changes. Because Update Management uses the same methods to update packages that an administrator would use locally on the Linux computer, this behavior is intentional.

To avoid updating the OS version via Update Management runs, use the **Exclusion** feature.

In Red Hat Enterprise Linux, the package name to exclude is `redhat-release-server.x86_64`.

<p>New update deployment</p> <p>Update Deployment</p> <p>* Name ⓘ Updates</p> <p>Operating system <input checked="" type="radio"/> Windows <input type="radio"/> Linux</p> <hr/> <p>* Machines to update > <i>Click to Configure</i></p> <hr/> <p>* Update classifications 2 selected</p> <hr/> <p>Updates to exclude > <i>Click to Configure</i></p> <hr/> <p>* Schedule settings > <i>Click to Configure</i></p> <hr/> <p>Maintenance window (minutes) ⓘ 120</p>	<p>Exclude Updates</p> <p>Provide a list of package names, that need to be skipped during the updates deployment. Wildcards syntax is supported.</p> <p>PACKAGE NAMES</p> <p>redhat-release-server.x86_64 ...</p> <p>Enter package name, e.g. *kernel* ...</p>
<input type="button" value="Create"/>	<input type="button" value="OK"/>

Critical / security patches aren't applied

When you deploy updates to a Linux machine, you can select update classifications. This filters the updates that are applied to those that meet the specified criteria. This filter is applied locally on the machine when the update is deployed.

Because Update Management performs update enrichment in the cloud, some updates might be flagged in Update Management as having security impact, even though the local machine doesn't have that information. As a result, if you apply critical updates to a Linux machine, there might be updates that aren't marked as having security impact on that machine and the updates aren't applied.

However, Update Management might still report that machine as being non-compliant because it has additional information about the relevant update.

Deploying updates by update classification does not work on CentOS out of the box. For SUSE, selecting *only* 'Other updates' as the classification may result in some security updates also being installed if security updates related to zypper (package manager) or its dependencies are required first. This is a limitation of zypper. In some cases, you may be required to re-run the update deployment, to verify check the update log.

Troubleshoot

To learn how to troubleshoot your Update Management, see [Troubleshooting Update Management](#)

Next steps

Continue to the tutorial to learn how to manage updates for your Windows virtual machines.

Manage updates and patches for your Azure Windows VMs

- Use log searches in [Log Analytics](#) to view detailed update data.
- [Create alerts](#) when critical updates are detected as missing from computers or if a computer has automatic updates disabled.

Manage updates for multiple machines

6/26/2018 • 6 minutes to read • [Edit Online](#)

You can use the Update Management solution to manage updates and patches for your Windows and Linux virtual machines. From your [Azure Automation](#) account, you can:

- Onboard virtual machines
- Assess the status of available updates
- Schedule installation of required updates
- Review deployment results to verify that updates were applied successfully to all virtual machines for which Update Management is enabled

Prerequisites

To use Update Management, you need:

- An Azure Automation Run As account. To learn how to create one, see [Getting started with Azure Automation](#).
- A virtual machine or computer with one of the supported operating systems installed.

Supported operating systems

Update Management is supported on the following operating systems:

OPERATING SYSTEM	NOTES
Windows Server 2008, Windows Server 2008 R2 RTM	Supports only update assessments.
Windows Server 2008 R2 SP1 and later	Windows PowerShell 4.0 or later is required. (Download WMF 4.0) Windows PowerShell 5.1 is recommended for increased reliability. (Download WMF 5.1)
CentOS 6 (x86/x64) and 7 (x64)	Linux agents must have access to an update repository.
Red Hat Enterprise 6 (x86/x64) and 7 (x64)	Linux agents must have access to an update repository.
SUSE Linux Enterprise Server 11 (x86/x64) and 12 (x64)	Linux agents must have access to an update repository.
Ubuntu 12.04 LTS, 14.04 LTS, and 16.04 LTS (x86/x64)	Linux agents must have access to an update repository.

NOTE

To prevent updates from being applied outside a maintenance window on Ubuntu, reconfigure the Unattended-Upgrade package to disable automatic updates. For more information, see the [Automatic Updates topic in the Ubuntu Server Guide](#).

Linux agents must have access to an update repository.

This solution doesn't support an Operations Management Suite (OMS) Agent for Linux that's configured to report to multiple Azure Log Analytics workspaces.

Enable Update Management for Azure virtual machines

In the Azure portal, open your Automation account, and then select **Update management**.

Select **Add Azure VMs**.

MACHINE NAME	COMPLIANCE	OPERATING SYSTEM	CRITICAL MISSING UPD...	SECURITY MISSING U...	OTHER MISSING UPD...	UPDATE AGENT READI...
CAS01.internal.lab	Non-compliant as of 6/6/2018, 5:33 PM	Windows	1	0	4	✓ Ready (view)
DC01.internal.lab	Non-compliant as of 6/6/2018, 3:33 PM	Windows	1	0	4	✓ Ready (view)
SQL01.internal.lab	Non-compliant as of 6/6/2018, 2:59 PM	Windows	1	0	4	✓ Ready (view)
LinuxVM2	Compliant as of 6/6/2018, 5:16 PM	Linux	0	0	24	✓ Ready (view)

Select a virtual machine to onboard.

Under **Enable Update Management**, select **Enable** to onboard the virtual machine.

NAME	UPDATE MANAGEMENT	DETAILS
winvm6	✓ Already enabled	
winvm5	✓ Already enabled	
winvm4	➡ Ready to enable	
winvm3	➡ Ready to enable	
winvm2	⌚ Cannot enable	VM agent is missing or not responding.
winvm1	⌚ Cannot enable	VM reports to workspace: 'defaultworkspace-147a22e9-2356-4e56-b3de-1f5842ae4a3b-eus'.
linuxvm6	✓ Already enabled	

When onboarding is finished, Update Management is enabled for your virtual machine.

Enable Update Management for non-Azure virtual machines and computers

To learn how to enable Update Management for non-Azure Windows virtual machines and computers, see [Connect Windows computers to the Log Analytics service in Azure](#).

To learn how to enable Update Management for non-Azure Linux virtual machines and computers, see [Connect your Linux computers to Log Analytics](#).

View computers attached to your Automation account

After you enable Update Management for your machines, you can view machine information by selecting **Computers**. You can see information about *machine name, compliance status, environment, OS type, critical and security updates installed, other updates installed, and update agent readiness* for your computers.

MACHINE NAME	COMPLIANCE	OPERATING SYSTEM	CRITICAL MISSING UPD...	SECURITY MISSING U...	OTHER MISSING UPD...	UPDATE AGENT READI...
CAS01.internal.lab	Non-compliant as of 6/6/2018, 5:33 PM	Windows	1	0	4	Ready (view)
DC01.internal.lab	Non-compliant as of 6/6/2018, 3:33 PM	Windows	1	0	4	Ready (view)
SQL01.internal.lab	Non-compliant as of 6/6/2018, 2:59 PM	Windows	1	0	4	Ready (view)
LinuxVM2	Compliant as of 6/6/2018, 5:16 PM	Linux	0	0	24	Ready (view)

Computers that have recently been enabled for Update Management might not have been assessed yet. The compliance state status for those computers is **Not assessed**. Here's a list of possible values for compliance state:

- **Compliant:** Computers that are not missing critical or security updates.
- **Non-compliant:** Computers that are missing at least one critical or security update.
- **Not assessed:** The update assessment data hasn't been received from the computer within the expected timeframe. For Linux computers, the expect timeframe is in the last 3 hours. For Windows computers, the expected timeframe is in the last 12 hours.

To view the status of the agent, select the link in the **UPDATE AGENT READINESS** column. Selecting this option opens the **Hybrid Worker** pane, and shows the status of the Hybrid Worker. The following image shows an example of an agent that hasn't been connected to Update Management for an extended period of time:

ContosoVM1_ec30e6e0-889e-4794-abe8-0ff1d8ab0021

HybridWorkerGroup

Search (Ctrl+ /)

Overview

SETTINGS

Properties

HYBRID WORKER GROUPS

Hybrid worker group settings

Hybrid Workers

Delete

Hybrid runbook workers of the group have been out of reach since 4/7/2018, 8:38 AM. Click here for the troubleshooting information.

Essentials

Resource group: TestAzureAuto

Location: eastus2

Last registration time: 3/15/2018, 6:44 PM

Group type: System

Account: TestAzureAuto

Subscription name: Microsoft Azure

Last seen time: 4/7/2018, 8:38 AM

Run As: --

Details

Hybrid Workers

1

View an update assessment

After Update Management is enabled, the **Update management** pane opens. You can see a list of missing

updates on the **Missing updates** tab.

Collect data

Agents that are installed on virtual machines and computers collect data about updates. The agents send the data to Azure Update Management.

Supported agents

The following table describes the connected sources that this solution supports:

CONNECTED SOURCE	SUPPORTED	DESCRIPTION
Windows agents	Yes	Update Management collects information about system updates from Windows agents and then initiates installation of required updates.
Linux agents	Yes	Update Management collects information about system updates from Linux agents and then initiates installation of required updates on supported distributions.
Operations Manager management group	Yes	Update Management collects information about system updates from agents in a connected management group.
Azure Storage account	No	Azure Storage doesn't include information about system updates.

Collection frequency

A scan runs twice a day for each managed Windows computer. Every 15 minutes, the Windows API is called to query for the last update time to determine whether the status has changed. If the status changed, a compliance scan starts. A scan runs every 3 hours for each managed Linux computer.

It can take between 30 minutes and 6 hours for the dashboard to display updated data from managed computers.

Schedule an update deployment

To install updates, schedule a deployment that aligns with your release schedule and service window. You can choose which update types to include in the deployment. For example, you can include critical or security updates and exclude update rollups.

To schedule a new update deployment for one or more virtual machines, under **Update management**, select **Schedule update deployment**.

In the **New update deployment** pane, specify the following information:

- **Name:** Enter a unique name to identify the update deployment.
- **Operating system:** Select **Windows** or **Linux**.
- **Machines to update:** Select the virtual machines that you want to update. The readiness of the machine is shown in the **UPDATE AGENT READINESS** column. You can see the health state of the machine before you schedule the update deployment.

The screenshot shows two overlapping windows. The left window is titled 'New update deployment' and contains fields for 'Name' (April Updates), 'Operating system' (Windows selected), and sections for 'Machines to update', 'Update classifications', 'Updates to exclude', and 'Schedule settings'. The right window is titled 'Select machines' and shows a list of available items (CAS01.internal.lab, ContosoVM1, DC01.internal.lab, SQL01.internal.lab) with their update agent readiness status (Ready, Disconnected). A 'Selected items' section below shows 'None'.

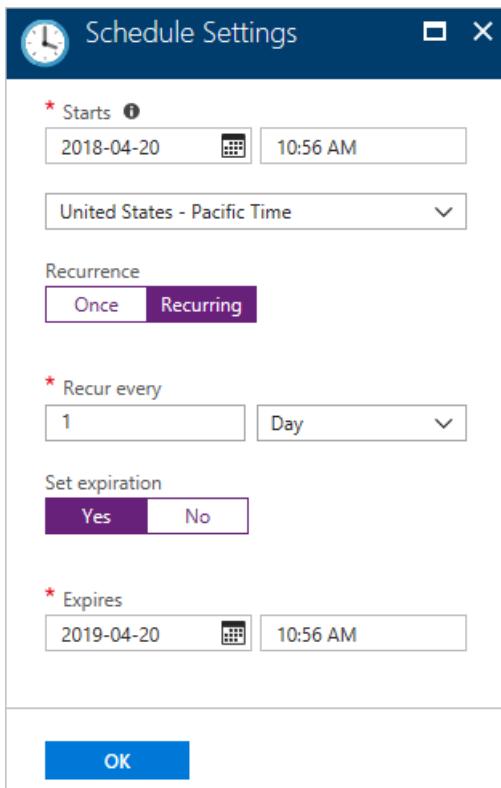
- **Update classification:** Select the types of software to include in the update deployment. For a description of the classification types, see [Update classifications](#). The classification types are:

- Critical updates
- Security updates
- Update rollups
- Feature packs
- Service packs
- Definition updates
- Tools
- Updates

- **Updates to exclude:** Selecting this option opens the **Exclude** page. Enter the KB articles or package names to exclude.

- **Schedule settings:** You can accept the default date and time, which is 30 minutes after the current time. You can also specify a different time.

You can also specify whether the deployment occurs once or on a recurring schedule. To set up a recurring schedule, under **Recurrence**, select **Recurring**.



- **Maintenance window (minutes):** Specify the period of time that you want the update deployment to occur. This setting helps ensure that changes are performed within your defined service windows.

When you're finished configuring the schedule, select the **Create** button to return to the status dashboard. The **Scheduled** table shows the deployment schedule that you created.

WARNING

For updates that require a restart, the virtual machine restarts automatically.

View results of an update deployment

After the scheduled deployment starts, you can see the status for that deployment on the **Update deployments** tab under **Update management**.

If the deployment is currently running, its status is **In progress**. After the deployment finishes successfully, the status changes to **Succeeded**.

If one or more updates fail in the deployment, the status is **Partially failed**.

The screenshot shows the Azure Update Management dashboard. At the top, there are four tabs: 'Schedule update deployment' (with a calendar icon), 'Add Azure VM' (with a plus icon), 'Add non-Azure machine' (with a gear icon), and 'Manage machines' (with a wrench icon). Below these are four summary sections: 'Non-compliant machines' (0 out of 5), 'Machines need attention' (5), 'Missing updates' (3), and 'Failed update deployments' (2 out of 8 in the past six months). A 'Learn more' link and 'Update Management' and 'Provide feedback' buttons are also present. Below these sections are three navigation links: 'Machines (5)', 'Missing updates (3)', and 'Update deployments' (which is selected and highlighted with a blue border). A dropdown menu labeled 'Status: All' is shown. The main content area is a table titled 'Update deployments' with columns: NAME, STATUS, MAINTENANCE WINDOW UTILIZATION, and START TIME. The table lists eight completed deployments:

NAME	STATUS	MAINTENANCE WINDOW UTILIZATION	START TIME
April Updates	✓ Succeeded	19 / 120 minutes	4/16/2018 11:40 AM
MonthlyAllUpdates	✓ Succeeded	34 / 120 minutes	4/13/2018 2:29 PM
HRAppWebServer-Monthly	✓ Succeeded	0 / 120 minutes	4/13/2018 11:47 AM
Example Deployment	✓ Succeeded	43 / 120 minutes	4/12/2018 12:22 PM
PayrollDB-Criticalupdates	✓ Succeeded	46 / 120 minutes	4/4/2018 8:50 PM
PayRollMonthly	! Failed	1 / 120 minutes	4/4/2018 8:02 PM
Test Updates	! Failed	73 / 120 minutes	4/4/2018 5:42 PM
Missing Updates	✓ Succeeded	68 / 120 minutes	12/7/2017 1:10 PM

To see the dashboard for an update deployment, select the completed deployment.

The **Update results** pane shows the total number of updates and the deployment results for the virtual machine. The table on the right gives a detailed breakdown of each update and the installation results. Installation results can be one of the following values:

- **Not attempted:** The update was not installed because insufficient time was available based on the defined maintenance window.
- **Succeeded:** The update succeeded.
- **Failed:** The update failed.

To see all log entries that the deployment created, select **All logs**.

To see the job stream of the runbook that manages the update deployment on the target virtual machine, select the output tile.

To see detailed information about any errors from the deployment, select **Errors**.

Next steps

- To learn more about Update Management, including logs, output, and errors, see [Update Management solution in Azure](#).

Integrate System Center Configuration Manager with Update Management

5/21/2018 • 3 minutes to read • [Edit Online](#)

Customers who have invested in System Center Configuration Manager to manage PCs, servers, and mobile devices also rely on its strength and maturity in managing software updates as part of their software update management (SUM) cycle.

You can report and update managed Windows servers by creating and pre-staging software update deployments in Configuration Manager, and get detailed status of completed update deployments using the [Update Management solution](#). If you use Configuration Manager for update compliance reporting but not for managing update deployments with your Windows servers, you can continue reporting to Configuration Manager while security updates are managed with the Update Management solution.

Prerequisites

- You must have the [Update Management solution](#) added to your Automation account.
- Windows servers currently managed by your System Center Configuration Manager environment also need to report to the Log Analytics workspace that also has the Update Management solution enabled.
- This feature is enabled in System Center Configuration Manager current branch version 1606 and higher. To integrate your Configuration Manager central administration site or a stand-alone primary site with Log Analytics and import collections, review [Connect Configuration Manager to Log Analytics](#).
- Windows agents must either be configured to communicate with a Windows Server Update Services (WSUS) server or have access to Microsoft Update if they don't receive security updates from Configuration Manager.

How you manage clients hosted in Azure IaaS with your existing Configuration Manager environment primarily depends on the connection you have between Azure datacenters and your infrastructure. This connection affects any design changes you may need to make to your Configuration Manager infrastructure and related cost to support those necessary changes. To understand what planning considerations you need to evaluate before proceeding, review [Configuration Manager on Azure - Frequently Asked Questions](#).

Configuration

Manage software updates from Configuration Manager

Perform the following steps if you are going to continue managing update deployments from Configuration Manager. Azure Automation connects to Configuration Manager to apply updates to the client computers connected to your Log Analytics workspace. Update content is available from the client computer cache as if the deployment were managed by Configuration Manager.

1. Create a software update deployment from the top-level site in your Configuration Manager hierarchy using the process described in [deploy software update process](#). The only setting that must be configured differently from a standard deployment is the option **Do not install software updates** to control the download behavior of the deployment package. This behavior is managed by the Update Management solution by creating a scheduled update deployment in the next step.
2. In Azure Automation, select **Update Management**. Create a new deployment following the steps described in [Creating an Update Deployment](#) and select **Imported groups** on the **Type** drop-down to select the appropriate Configuration Manager collection. Keep in mind the following important points: a. If a maintenance window is defined on the selected Configuration Manager device collection, members of the

collection honor it instead of the **Duration** setting defined in the scheduled deployment. b. Members of the target collection must have a connection to the Internet (either direct, through a proxy server or through the OMS Gateway).

After completing the update deployment through Azure Automation, the target computers that are members of the selected computer group will install updates at the scheduled time from their local client cache. You can [view update deployment status](#) to monitor the results of your deployment.

Manage software updates from Azure Automation

To manage updates for Windows Server VMs that are Configuration Manager clients, you need to configure client policy to disable the Software Update Management feature for all clients managed by this solution. By default, client settings target all devices in the hierarchy. For more information about this policy setting and how to configure it, review [how to configure client settings in System Center Configuration Manager](#).

After performing this configuration change, you create a new deployment following the steps described in [Creating an Update Deployment](#) and select **Imported groups** on the **Type** drop-down to select the appropriate Configuration Manager collection.

Next steps

Track changes in your environment with the Change Tracking solution

7/6/2018 • 9 minutes to read • [Edit Online](#)

This article helps you use the Change Tracking solution to easily identify changes in your environment. The solution tracks changes to Windows and Linux software, Windows and Linux files, Windows registry keys, Windows services, and Linux daemons. Identifying configuration changes can help you pinpoint operational issues.

Changes to installed software, Windows services, Windows registry and files, and Linux daemons on the monitored servers are sent to the Log Analytics service in the cloud for processing. Logic is applied to the received data and the cloud service records the data. By using the information on the Change Tracking dashboard, you can easily see the changes that were made in your server infrastructure.

Enable Change Tracking and Inventory

To begin tracking changes, you need to enable the Change Tracking and Inventory solution for your Automation Account.

1. In the Azure portal, navigate to your Automation Account
2. Select **Change Tracking** under **CONFIGURATION**.
3. Select an existing Log analytics workspace or [Create New Workspace](#) and click **Enable**.

This enables the solution for your automation account. The solution can take up to 15 minutes to enable. The blue banner notifies you when the solution is enabled. Navigate back to the **Change Tracking** page to manage the solution.

Configuring Change Tracking and Inventory

To learn how to onboard computers to the solution visit: [Onboarding Automation solutions](#). Once you have a machine onboarding with the Change Tracking and Inventory solution you can configure the items to track. When you enable a new file or registry key to track, it is enabled for both Change Tracking and Inventory.

For tracking changes in files on both Windows and Linux, MD5 hashes of the files are used. These hashes are then used to detect if a change has been made since the last inventory.

Configure Linux files to track

Use the following steps to configure file tracking on Linux computers:

1. In your Automation Account, select **Change tracking** under **CONFIGURATION MANAGEMENT**. Click **Edit Settings** (the gear symbol).
2. On the **Change Tracking** page, select **Linux Files**, then click **+ Add** to add a new file to track.
3. On the **Add Linux File for Change Tracking**, enter the information for the file or directory to track and click **Save**.

PROPERTY	DESCRIPTION
Enabled	Determines if the setting is applied.

PROPERTY	DESCRIPTION
Item Name	Friendly name of the file to be tracked.
Group	A group name for logically grouping files.
Enter Path	The path to check for the file. For example: "/etc/*.conf"
Path Type	Type of item to be tracked, possible values are File and Directory.
Recursion	Determines if recursion is used when looking for the item to be tracked.
Use Sudo	This setting determines if sudo is used when checking for the item.
Links	<p>This setting determines how symbolic links dealt with when traversing directories.</p> <p>Ignore - Ignores symbolic links and does not include the files/directories referenced.</p> <p>Follow - Follows the symbolic links during recursion and also includes the files/directories referenced.</p> <p>Manage - Follows the symbolic links and allows altering of returned content.</p>
Upload file content for all settings	<p>Turns on or off file content upload on tracked changes.</p> <p>Available options: True or False.</p>

NOTE

The "Manage" links option is not recommended. File content retrieval is not supported.

Configure Windows files to track

Use the following steps to configure files tracking on Windows computers:

1. In your Automation Account, select **Change tracking** under **CONFIGURATION MANAGEMENT**. Click **Edit Settings** (the gear symbol).
2. On the **Change Tracking** page, select **Windows Files**, then click **+ Add** to add a new file to track.
3. On the **Add Windows File for Change Tracking**, enter the information for the file to track and click **Save**.

PROPERTY	DESCRIPTION
Enabled	Determines if the setting is applied.
Item Name	Friendly name of the file to be tracked.
Group	A group name for logically grouping files.
Enter Path	The path to check for the file For example: "c:\temp\myfile.txt"
Upload file content for all settings	<p>Turns on or off file content upload on tracked changes.</p> <p>Available options: True or False.</p>

Configure File Content tracking

You can view the the contents before and after a change of a file with File Content Change Tracking. This is available for Windows and Linux files, for each change to the file, the contents of the file is stored in a storage account, and shows the file before and after the change, inline or side by side. To learn more, see [View the contents of a tracked file](#).

The screenshot shows a comparison of file content changes. At the top, there's a navigation bar: Home > Automation Accounts > TestAzureAuto - Change tracking > View File Content Changes. Below that is a header: View File Content Changes /etc/hosts. There are two tabs: "Side by side" (selected) and "Inline". The main area displays two versions of the /etc/hosts file. The left version (version 2) has several lines crossed out with diagonal hatching. The right version (version 4+) shows the updated content. Both versions have line numbers from 1 to 21.

1 127.0.0.1 localhost	1 127.0.0.1 localhost
2	2
3	3 + 5.6.7.8 badhost
4	4 +
5	5
6	6
7 # The following lines are desirable for IPv6 capable hosts	7 # The following lines are desirable for IPv6 capable hosts
8 ::1 ip6-localhost ip6-loopback	8 ::1 ip6-localhost ip6-loopback
9 fe00::0 ip6-localnet	9 fe00::0 ip6-localnet
10 ff00::0 ip6-mcastprefix	10 ff00::0 ip6-mcastprefix
11 ff02::1 ip6-allnodes	11 ff02::1 ip6-allnodes
12 ff02::2 ip6-allrouters	12 ff02::2 ip6-allrouters
13 ff02::3 ip6-allhosts	13 ff02::3 ip6-allhosts
14	14
15	15
16	16
17	17
18	18
19	19
20	20
21	21

Configure Windows registry keys to track

Use the following steps to configure registry key tracking on Windows computers:

1. In your Automation Account, select **Change tracking** under **CONFIGURATION MANAGEMENT**. Click **Edit Settings** (the gear symbol).
2. On the **Change Tracking** page, select **Windows Registry**, then click **+ Add** to add a new registry key to track.
3. On the **Add Windows Registry for Change Tracking**, enter the information for the key to track and click **Save**.

PROPERTY	DESCRIPTION
Enabled	Determines if the setting is applied.
Item Name	Friendly name of the file to be tracked.
Group	A group name for logically grouping files.
Windows Registry Key	The path to check for the file. For example: "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders\Common Startup"

Limitations

The Change Tracking solution does not currently support the following items:

- Folders (directories) for Windows file tracking

- Recursion for Windows file tracking
- Wild cards for Windows file tracking
- Recursion for Windows registry tracking
- Path variables
- Network file systems
- File Content

Other limitations:

- The **Max File Size** column and values are unused in the current implementation.
- If you collect more than 2500 files in the 30-minute collection cycle, solution performance might be degraded.
- When network traffic is high, change records may take up to six hours to display.
- If you modify the configuration while a computer is shut down, the computer might post changes that belonged to the previous configuration.

Known Issues

The Change Tracking solution is currently experiencing the following issues:

- Hotfix updates are not collected for Windows 10 Creators Update and Windows Server 2016 Core RS3 machines.

Change Tracking data collection details

The following table shows the data collection frequency for the types of changes. For every type the data snapshot of the current state is also refreshed at least every 24 hours:

CHANGE TYPE	FREQUENCY
Windows registry	50 minutes
Windows file	30 minutes
Linux file	15 minutes
Windows services	10 seconds to 30 minutes Default: 30 minutes
Linux daemons	5 minutes
Windows software	30 minutes
Linux software	5 minutes

Windows service tracking

The default collection frequency for Windows services is 30 minutes. To configure the frequency go to **Change Tracking**. Under **Edit Settings** on the **Windows Services** tab, there is a slider that allows you to change the collection frequency for Windows services from as quickly as 10 seconds to as long as 30 minutes. Move the slider bar to the frequency you want and it automatically saves it.

Workspace Configuration

Change Tracking

 Documentation

[Windows Registry](#) [Windows Files](#) [Linux Files](#) [File Content](#) [Windows Services](#)

Adjust the collection frequency for Windows Services changes.

Collection Frequency 



The agent only tracks changes, this optimizes the performance of the agent. By setting too high of a threshold changes may be missed if the service reverted to their original state. Setting the frequency to a smaller value allows you to catch changes that may be missed otherwise.

NOTE

While the agent can track changes down to a 10 second interval, the data still takes a few minutes to be displayed in the portal. Changes during the time to display in the portal are still tracked and logged.

Registry key change tracking

The purpose of monitoring changes to registry keys is to pinpoint extensibility points where third-party code and malware can activate. The following list shows the list of pre-configured registry keys. These keys are configured but not enabled. To track these registry keys, you must enable each one.

HKEY_LOCAL_MACHINE\Software\Classes\Directory\ShellEx\ContextMenuHandlers

Monitors common autostart entries that hook directly into Windows Explorer and usually run in-process with Explorer.exe.

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Group Policy\Scripts\Startup

Monitors scripts that run at startup.

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Group Policy\Scripts\Shutdown

Monitors scripts that run at shutdown.

HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Run

Monitors keys that are loaded before the user signs in to their Windows account. The key is used for 32-bit programs running on 64-bit computers.

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Active Setup\Installed Components

Monitors changes to application settings.

HKEY_LOCAL_MACHINE\Software\Classes\Directory\ShellEx\ContextMenuHandlers

Monitors common autostart entries that hook directly into Windows Explorer and usually run in-process with Explorer.exe.

HKEY_LOCAL_MACHINE\Software\Classes\Directory\Shellex\CopyHookHandlers

Monitors common autostart entries that hook directly into Windows Explorer and usually run in-process with Explorer.exe.

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Explorer\ShellIconOverlayIdentifiers

Monitors for icon overlay handler registration.

HKEY_LOCAL_MACHINE\Software\Wow6432Node\Microsoft\Windows\CurrentVersion\Explorer\ShellIconOverlayIdentifiers

Monitors for icon overlay handler registration for 32-bit programs running on 64-bit computers.

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects

Monitors for new browser helper object plugins for Internet Explorer. Used to access the Document Object Model (DOM) of the current page and to control navigation.

HKEY_LOCAL_MACHINE\Software\Wow6432Node\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects

Monitors for new browser helper object plugins for Internet Explorer. Used to access the Document Object Model (DOM) of the current page and to control navigation for 32-bit programs running on 64-bit computers.

HKEY_LOCAL_MACHINE\Software\Microsoft\Internet Explorer\Extensions

Monitors for new Internet Explorer extensions, such as custom tool menus and custom toolbar buttons.

HKEY_LOCAL_MACHINE\Software\Wow6432Node\Microsoft\Internet Explorer\Extensions

Monitors for new Internet Explorer extensions, such as custom tool menus and custom toolbar buttons for 32-bit programs running on 64-bit computers.

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Drivers32

Monitors the 32-bit drivers associated with wavemapper, wave1 and wave2, msacm.imaadpcm, .msadpcm, .msgsm610, and vidc. Similar to the [drivers] section in the SYSTEM.INI file.

HKEY_LOCAL_MACHINE\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Drivers32

Monitors the 32-bit drivers associated with wavemapper, wave1 and wave2, msacm.imaadpcm, .msadpcm, .msgsm610, and vidc for 32-bit programs running on 64-bit computers. Similar to the [drivers] section in the SYSTEM.INI file.

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager\KnownDLLs

Monitors the list of known or commonly used system DLLs; this system prevents people from exploiting weak application directory permissions by dropping in Trojan horse versions of system DLLs.

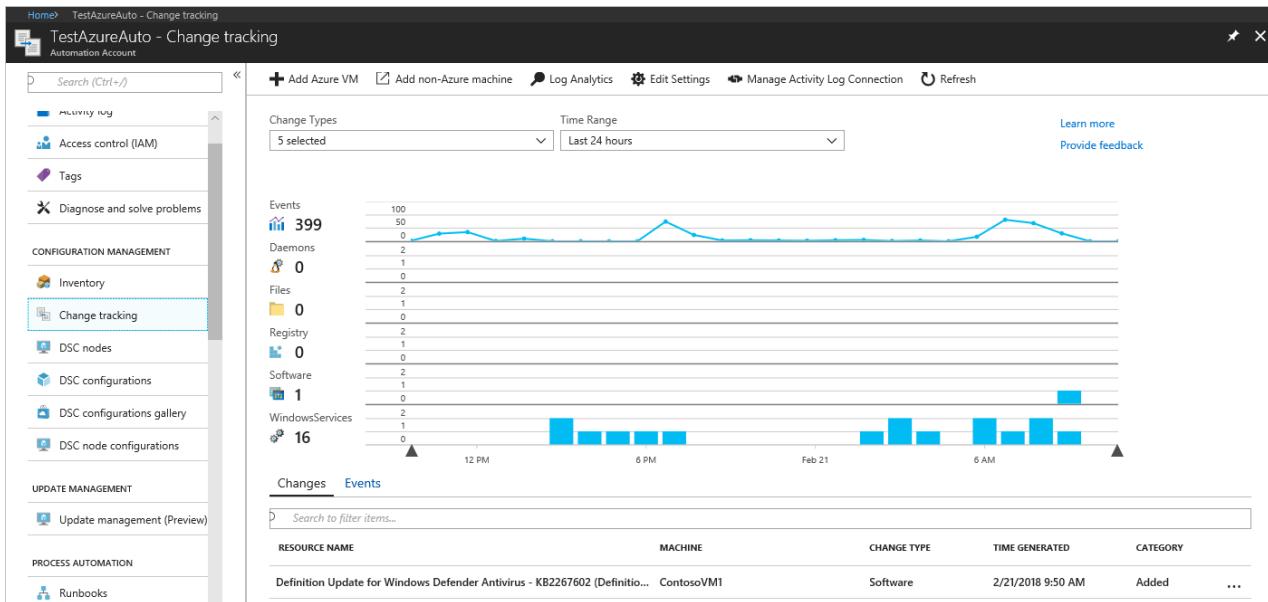
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Notify

Monitors the list of packages able to receive event notifications from Winlogon, the interactive logon support model for the Windows operating system.

Use Change Tracking

After the solution is enabled, you can view the summary of changes for your monitored computers by selecting **Change Tracking** under **CONFIGURATION MANAGEMENT** in your Automation account.

You can view changes to your computers and then drill-into details for each event. Drop downs are available at the top of the chart to limit the chart and detailed information based on change type and time ranges. You can also click and drag on the chart to select a custom time range.



Clicking on a change or event brings up the detailed information about that change. As you can see from the example, the startup type of the service was changed from Manual to Auto.

Background Intelligent Transfer Service		
Change details		
PROPERTY	VALUE BEFORE	VALUE AFTER
SvcStartupType	Manual	Auto
▼ Unchanged properties		
SourceComputerId	4f1e434c	No Change
SvcDisplayName	Background Intelligent Transfer Service	No Change
SvcName	BITS	No Change
SvcState	Running	No Change
SvcAccount	LocalSystem	No Change
SvcPath	C:\Windows\System32\svchost.exe -k netsvcs	No Change
SourceSystem	OpsManager	No Change
MG	00000000-0000-0000-0000-000000000001	No Change
ManagementGroupNa...	AOI-68993555	No Change
TenantId		No Change
VMUUID		No Change

Search logs

In addition to the details that are provided in the portal, searches can be done against the logs. With the **Change**

Tracking page open, click **Log Analytics**, this opens the **Log Search** page.

Sample queries

The following table provides sample log searches for change records collected by this solution:

QUERY	DESCRIPTION
<pre>ConfigurationData where ConfigDataType == "WindowsServices" and SvcStartupType == "Auto" where SvcState == "Stopped" summarize arg_max(TimeGenerated, *) by SoftwareName, Computer</pre>	Shows the most recent inventory records for Windows Services that were set to Auto but were reported as being Stopped Results are limited to the most recent record for that SoftwareName and Computer
<pre>ConfigurationChange where ConfigChangeType == "Software" and ChangeCategory == "Removed" order by TimeGenerated desc</pre>	Shows the change records for removed software

Next steps

Visit the tutorial on Change Tracking to learn more about using the solution:

[Troubleshoot changes in your environment](#)

- Use [Log searches in Log Analytics](#) to view detailed change tracking data.

View contents of a file that is being tracked with Change Tracking

7/6/2018 • 2 minutes to read • [Edit Online](#)

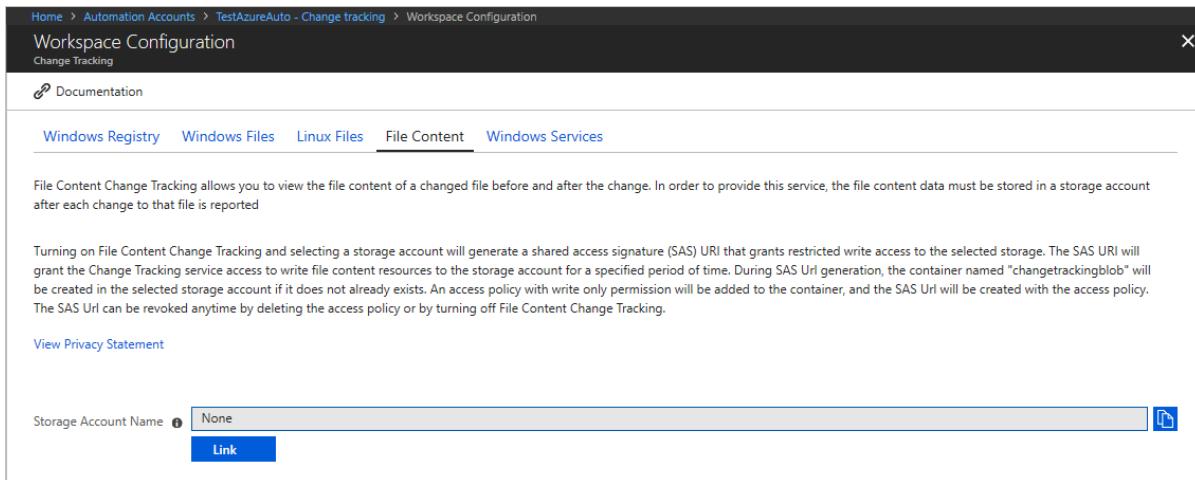
File content tracking allows you to view the contents of a file before and after a change that is being tracked with Change Tracking. To do this, it saves the file contents to a storage account after each change occurs.

Requirements

- A standard storage account using the Resource Manager deployment model is required for storing file content. Premium and classic deployment model storage accounts should not be used. For more information on storage accounts, see [About Azure storage accounts](#)
- The storage account used can only have 1 Automation Account connected.
- [Change Tracking](#) is enabled in your Automation Account.

Enable file content tracking

1. In the Azure portal, open your Automation account, and then select **Change tracking**.
2. On the top menu, select **Edit Settings**.
3. Select **File Content** and click **Link**. This opens the **Add Content Location for Change Tracking** pane.



4. Select the subscription and storage account to use to store the file contents to. If you want to enable file content tracking for all existing tracked files, select **On** for **Upload file content for all settings**. You can change this for each file path afterwards.

Add Content Location for Chang...

Save Delete Discard

Select Subscription [?](#)
Microsoft Azure

Select Storage [?](#)
changeblobexample

Upload file content for all settings [?](#)
 On Off

- Once enabled, the storage account and the SAS UrIs are shown. The SAS UrIs expire after 365 days, and can be recreated by clicking the **Regenerate** button.

Storage Account Name [?](#) changeblobexample [D](#)
 Unlink

Primary Write Only SAS Uri [?](#) https://changeblobexample.blob.core.windows.net/changetrackingblob?sv=2015-04-05&sr=c&si=changetrackingpolicykey1_1530629144955&sig=MQi%2Buc0l¹ [D](#)
 Regenerate

Secondary Write Only SAS Uri [?](#) None [D](#)
 Regenerate

Add a file

The following steps walk you through turning on change tracking for a file:

- On the **Edit Settings** page of **Change Tracking**, select either **Windows Files** or **Linux Files** tab, and click **Add**
- Fill out the information for the file path and select **True** under **Upload file content for all settings**. This setting enables file content tracking for that file path only.

Add Linux File for Change Tracking X

Save Delete X Discard

Enabled
 True False

*** Item Name**
 ✓

Group

*** Enter Path**
 ✓

Path Type

Recursion
 On Off

Use Sudo
 On Off

Links

Upload file content for all settings
 True False

Viewing the contents of a tracked file

- Once a change has been detected for the file or a file in the path, it shows in the portal. Select the file change from the list of changes. The **Change details** pane is displayed.

RESOURCE NAME	CHANGE TYPE	MACHINE	TIME GENERATED	CATEGORY
/etc/hosts	Files	LinuxVM2	7/3/2018, 9:00 AM	Modified
re-local	Daemons	LinuxVM2	7/3/2018, 8:40 AM	Removed
systemd-modules-load kmod	Daemons	LinuxVM2	7/3/2018, 8:40 AM	Removed

- On the **Change details** page, you see the standard before and after file information, in the top left, click **View File Content Changes** to see the contents of the file.

/etc/hosts

Change details

[View File Content Changes](#)

PROPERTY	VALUE BEFORE	VALUE AFTER
DateCreated	4/18/2018, 1:29:47 AM	7/3/2018, 8:56:20 AM
DateModified	4/16/2018, 2:02:20 PM	7/3/2018, 8:56:20 AM
FileContentCheck...	1523912540	1530633380
Size	221	237
▼ Unchanged properties		
SourceComputerName		No Change
FileSystemPath	/etc/hosts	No Change
SourceSystem	OpsManager	No Change
MG	00000000-0000-0000-0000-000000000002	No Change
TenantId		No Change
VMUUID		No Change

3. The new page shows you the file contents in a side-by-side view. You can also select **Inline** to see an inline view of the changes.

Home > Automation Accounts > TestAzureAuto - Change tracking > View File Content Changes

/etc/hosts

[View File Content Changes](#)

[Side by side](#) [Inline](#)

```

1 127.0.0.1 localhost
2
3
4
5 # The following lines are desirable for IPv6 capable hosts
6
7 ::1 ip6-localhost ip6-loopback
8
9 fe00::0 ip6-localnet
10
11 ff00::0 ip6-mcastprefix
12
13 ff02::1 ip6-allnodes
14
15 ff02::2 ip6-allrouters
16
17 ff02::3 ip6-allhosts
18
19
20
21

```

[Side by side](#) [Inline](#)

```

1 127.0.0.1 localhost
2
3 + 5.6.7.8 badhost
4 +
5
6
7 # The following lines are desirable for IPv6 capable hosts
8
9 ::1 ip6-localhost ip6-loopback
10
11 fe00::0 ip6-localnet
12
13 ff00::0 ip6-mcastprefix
14
15 ff02::1 ip6-allnodes
16
17 ff02::2 ip6-allrouters
18
19 ff02::3 ip6-allhosts
20
21

```

Next steps

Visit the tutorial on Change Tracking to learn more about using the solution:

[Troubleshoot changes in your environment](#)

- Use [Log searches in Log Analytics](#) to view detailed change tracking data.

Manage an Azure virtual machine with inventory collection

6/13/2018 • 4 minutes to read • [Edit Online](#)

You can enable inventory tracking for an Azure virtual machine from the virtual machine's resource page. This method provides a browser-based user interface for setting up and configuring inventory collection.

Before you begin

If you don't have an Azure subscription, [create a free account](#).

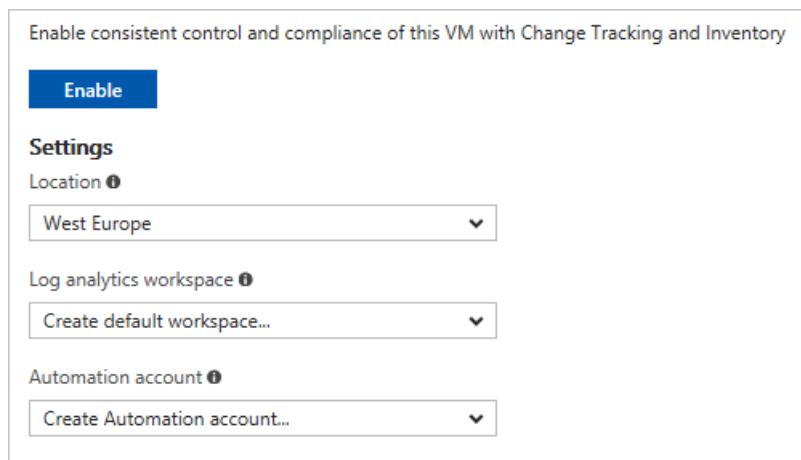
This article assumes you have a VM to configure the solution on. If you don't have an Azure virtual machine, [create a virtual machine](#).

Sign in to the Azure portal

Sign in to the [Azure portal](#).

Enable inventory collection from the virtual machine resource page

1. In the left pane of the Azure portal, select **Virtual machines**.
2. In the list of virtual machines, select a virtual machine.
3. On the **Resource** menu, under **Operations**, select **Inventory**.
4. Select a log analytics workspace for storing your data logs. If no workspace is available to you for that region, you are prompted to create a default workspace and automation account.
5. To start onboarding your computer, select **Enable**.



A status bar notifies you that the solution is being enabled. This process can take up to 15 minutes. During this time, you can close the window, or you can keep it open and it notifies you when the solution is enabled. You can monitor the deployment status from the notifications pane.

The screenshot shows the System Center Configuration Manager software interface. At the top, there are navigation links: 'Manage multiple computers', 'Edit Settings', and 'Log Analytics'. Below this, a status bar says 'New software last 24 hrs' with a refresh icon. A large blue '0' icon is displayed. On the right, there are links to 'Learn more' and 'Provide feedback'. The main area has tabs for 'Software', 'Files', 'Windows Registry', and 'Windows Services'. A search bar says 'Search to filter items...'. A table below shows inventory data with columns for 'NAME', 'VERSION', 'PUBLISHER', and 'LAST REFRESHED TIME'. The table displays 'No data'.

When the deployment is complete, the status bar disappears. The system is still collecting inventory data, and the data might not be visible yet. A full collection of data can take 24 hours.

Configure your inventory settings

By default, software, Windows services, and Linux daemons are configured for collection. To collect Windows registry and file inventory, configure the inventory collection settings.

1. In the **Inventory** view, select the **Edit Settings** button at the top of the window.
2. To add a new collection setting, go to the setting category that you want to add by selecting the **Windows Registry**, **Windows Files**, and **Linux Files** tabs.
3. Select the appropriate category and click **Add** at the top of the window.

The following tables provide information about each property that can be configured for the various categories.

Windows Registry

PROPERTY	DESCRIPTION
Enabled	Determines if the setting is applied
Item Name	Friendly name of the file to be tracked
Group	A group name for logically grouping files
Windows Registry Key	The path to check for the file For example: "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders\Common Startup"

Windows Files

PROPERTY	DESCRIPTION
Enabled	Determines if the setting is applied
Item Name	Friendly name of the file to be tracked
Group	A group name for logically grouping files
Enter Path	The path to check for the file For example: "c:\temp\myfile.txt"

Linux Files

PROPERTY	DESCRIPTION
Enabled	Determines if the setting is applied
Item Name	Friendly name of the file to be tracked
Group	A group name for logically grouping files
Enter Path	The path to check for the file For example: "/etc/*.conf"
Path Type	Type of item to be tracked, possible values are File and Directory
Recursion	Determines if recursion is used when looking for the item to be tracked.
Use Sudo	This setting determines if sudo is used when checking for the item.
Links	<p>This setting determines how symbolic links dealt with when traversing directories.</p> <p>Ignore - Ignores symbolic links and does not include the files/directories referenced</p> <p>Follow - Follows the symbolic links during recursion and also includes the files/directories referenced</p> <p>Manage - Follows the symbolic links and allows alter the treatment of returned content</p>

Manage machine groups

Inventory allows you to create and view machine groups in Log Analytics. Machine groups are collections of machines defined by a query in Log Analytics.

To view your machine groups select the **Machine groups** tab on the Inventory page.

The screenshot shows the Azure portal's 'Virtual machines > LinuxVM3 - Inventory' page. The 'Machine Groups' tab is active. At the top, there are buttons for 'Add Azure VMs', 'Add non-Azure machine', 'Manage machines', 'Log Analytics', 'Edit Settings', and 'Create a machine group'. Below this, a message says '5 machines do not have 'Change Tracking and Inventory' enabled. Click to manage machines'. The main area shows a summary: 'New software' (0) and 'Machines reporting' (3). A link to 'Learn more' leads to the 'Inventory' page. Below the summary, a navigation bar includes 'Machines(3)', 'Software(531)', 'Files(0)', 'Windows Registry(0)', 'Windows Services(0)', 'Linux Daemons(193)', and 'Machine Groups(3)', with 'Machine Groups(3)' highlighted. A search bar at the bottom is labeled 'Search to filter items...'. The table below lists machine groups:

NAME	CATEGORY	FUNCTION ALIAS
MicrosoftDefaultComputerGroup *	ChangeTracking	ChangeTracking__MicrosoftDefaultComputerGroup
Security Center Free Tier Machines	Security Center	Security_Center__Security_Center_Free_Tier_Machines
MicrosoftDefaultComputerGroup *	Updates	Updates__MicrosoftDefaultComputerGroup

* Learn more about default machine groups

Selecting a machine group from the list opens the Machine groups page. This page shows details about the machine group. These details include the log analytics query that is used to define the group. At the bottom of the page, is a paged list of the machines that are part of that group.

Security_Center_Security_Center_Free_Tier_Machines
Security Center Free Tier Machines

Clone **Delete**

Group name	securitycenterfreetiermachines
Category	Security Center
Function Alias	Security_Center_Security_Center_Free_Tier_Machines

Definition

```
Heartbeat | where SubscriptionId =~ "147a22e9-2356-4e56-b3de-1f5842ae4a3b" and ComputerEnvironment == "Azure" | distinct Computer | limit 200000
// Oql: Type=Heartbeat SubscriptionId=147a22e9-2356-4e56-b3de-1f5842ae4a3b and ComputerEnvironment=Azure | distinct Computer
```

MACHINE COUNT: 6

MACHINE	OPERATING SYSTEM	VERSION	PLATFORM
LinuxVM1	Linux	Ubuntu 16.04	Azure
aks-agentpool-42602657-0	Linux	Ubuntu 16.04	Azure
WinVM1	Windows	10.0	Azure
aks-agentpool-42602657-2	Linux	Ubuntu 16.04	Azure
LinuxVM3	Linux	Ubuntu 16.04	Azure
aks-agentpool-42602657-1	Linux	Ubuntu 16.04	Azure

Click the **+ Clone** button to clone the machine group. Here you must give the group a new name and alias for the group. The definition can be altered at this time. After changing the query press **Validate query** to preview the machines that would be selected. When you are happy with the group click **Create** to create the machine group

If you want to create a new machine group, select **+ Create a machine group**. This button opens the **Create a machine group page** where you can define your new group. Click **Create** to create the group.

Create a machine group X

* Group name !
Computers_Needing_updates ✓

Category !
 Create new Use existing

Updates ▼

* Function Alias !
Computers_Needing_updates ✓

Definition !

```
Update
| where UpdateState == "Needed" and Optional == false
| summarize by Computer
```

Validate query

COMPUTER

WinVM1

Create

Disconnect your virtual machine from management

To remove your virtual machine from inventory management:

1. In the left pane of the Azure portal, select **Log Analytics**, and then select the workspace that you used when you onboarded your virtual machine.
2. In the **Log Analytics** window, on the **Resource** menu, under the **Workspace Data Sources** category, select **Virtual machines**.
3. In the list, select the virtual machine that you want to disconnect. The virtual machine has a green check mark next to **This workspace** in the **OMS Connection** column.
4. At the top of the next page, select **Disconnect**.
5. In the confirmation window, select **Yes**. This action disconnects the machine from management.

Next steps

- To learn about managing changes in files and registry settings on your virtual machines, see [Track software changes in your environment with the Change Tracking solution](#).
- To learn about managing Windows and package updates on your virtual machines, see [The Update Management solution in Azure](#).

Create a standalone Azure Automation account

5/21/2018 • 5 minutes to read • [Edit Online](#)

This article shows you how to create an Azure Automation account in the Azure portal. You can use the portal Automation account to evaluate and learn about Automation without using additional management solutions or integration with Azure Log Analytics. You can add those management solutions or integrate with Log Analytics for advanced monitoring of runbook jobs at any point in the future.

With an Automation account, you can authenticate runbooks by managing resources in either Azure Resource Manager or the classic deployment model.

When you create an Automation account in the Azure portal, these accounts are automatically created:

- **Run As account.** This account does the following tasks:
 - Creates a service principal in Azure Active Directory (Azure AD).
 - Creates a certificate.
 - Assigns the Contributor Role-Based Access Control (RBAC), which manages Azure Resource Manager resources by using runbooks.
- **Classic Run As account.** This account uploads a management certificate. The certificate manages classic resources by using runbooks.

With these accounts created for you, you can quickly start building and deploying runbooks to support your automation needs.

Permissions required to create an Automation account

To create or update an Automation account, and to complete the tasks described in this article, you must have the following privileges and permissions:

- To create an Automation account, your Azure AD user account must be added to a role with permissions equivalent to the Owner role for **Microsoft. Automation** resources. For more information, see [Role-Based Access Control in Azure Automation](#).
- In the Azure portal, under **Azure Active Directory > MANAGE > App registrations**, if **App registrations** is set to **Yes**, non-admin users in your Azure AD tenant can [register Active Directory applications](#). If **App registrations** is set to **No**, the user who performs this action must be a global administrator in Azure AD.

If you aren't a member of the subscription's Active Directory instance before you are added to the subscription's global administrator/coadministrator role, you are added to Active Directory as a guest. In this scenario, you see this message on the **Add Automation Account** page: "You do not have permissions to create."

If a user is added to the global administrator/coadministrator role first, you can remove them from the subscription's Active Directory instance, and then re-add them to the full User role in Active Directory.

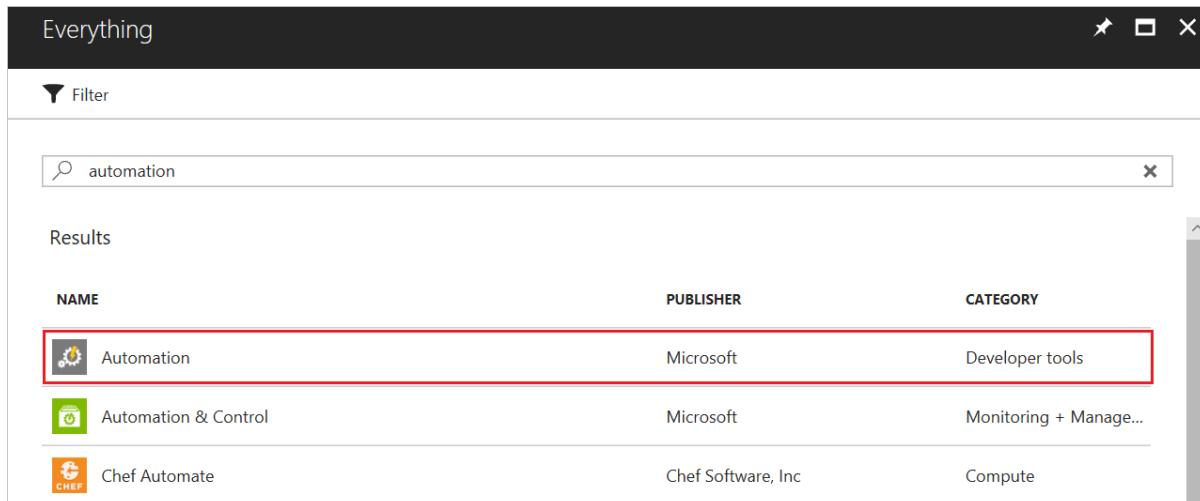
To verify user roles:

1. In the Azure portal, go to the **Azure Active Directory** pane.
2. Select **Users and groups**.
3. Select **All users**.
4. After you select a specific user, select **Profile**. The value of the **User type** attribute under the user's profile should not be **Guest**.

Create a new Automation account in the Azure portal

To create an Azure Automation account in the Azure portal, complete the following steps:

1. Sign in to the Azure portal with an account that's a member of the subscription Administrators role and a coadministrator of the subscription.
2. Select **+ Create a Resource**.
3. Search for **Automation**. In the search results, select **Automation**.



The screenshot shows the Azure portal search interface. At the top, there is a search bar with the placeholder 'Search' and a magnifying glass icon. Below the search bar, the word 'Everything' is displayed. A 'Filter' button is located next to the search bar. The main area is titled 'Results' and contains a table with three columns: 'NAME', 'PUBLISHER', and 'CATEGORY'. There are three items listed:

NAME	PUBLISHER	CATEGORY
Automation	Microsoft	Developer tools
Automation & Control	Microsoft	Monitoring + Manage...
Chef Automate	Chef Software, Inc	Compute

The first item, 'Automation' by Microsoft, is highlighted with a red border around its entire row.

Add Automation Account



* Name i

Enter the account name...

* Subscription

Microsoft Azure



* Resource group

Create new Use existing

* Location

Japan East



* Create Azure Run As account i



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.](#)



Learn more about Automation
pricing.



Pin to dashboard

Create

4. On the next screen select **Create**.

NOTE

If you see the following message in the **Add Automation Account** pane, your account is not a member of the subscription Administrators role and a coadministrator of the subscription.



You do not have permissions to create a Run As account in Azure Active directory. Please follow the directions in the documentation to learn how to create a Run As account.[Click here to learn more about Run As accounts.](#)

5. In the **Add Automation Account** pane, in the **Name** box, enter a name for your new Automation account.
6. If you have more than one subscription, in the **Subscription** box, specify the subscription you want to use for the new account.
7. For **Resource group**, enter or select a new or existing resource group.
8. For **Location**, select an Azure datacenter location.
9. For the **Create Azure Run As account** option, ensure that **Yes** is selected, and then select **Create**.

NOTE

If you choose not to create the Run As account by selecting **No** for **Create Azure Run As account**, a message appears in the **Add Automation Account** pane. Although the account is created in the Azure portal, the account doesn't have a corresponding authentication identity in your classic deployment model subscription or in the Azure Resource Manager subscription directory service. Therefore, the Automation account doesn't have access to resources in your subscription. This prevents any runbooks that reference this account from being able to authenticate and perform tasks against resources in those deployment models.



You have chosen not to create a Run As Account. Doing so might block the execution of some runbooks due to lack of access to required resources.[Click here to learn more about Run As accounts.](#)

When the service principal is not created, the Contributor role is not assigned.

10. To track the progress of the Automation account creation, in the menu, select **Notifications**.

Resources included

When the Automation account is successfully created, several resources are automatically created for you. After creation these the runbooks can be safely deleted if you do not wish to keep them. The Run As Accounts, can be used to authenticate to your account in a runbook, and should be left unless you create another one or do not require them. The following table summarizes resources for the Run As account.

RESOURCE	DESCRIPTION
AzureAutomationTutorial Runbook	An example graphical runbook that demonstrates how to authenticate by using the Run As account. The runbook gets all Resource Manager resources.

RESOURCE	DESCRIPTION
AzureAutomationTutorialScript Runbook	An example PowerShell runbook that demonstrates how to authenticate by using the Run As account. The runbook gets all Resource Manager resources.
AzureAutomationTutorialPython2 Runbook	An example Python runbook that demonstrates how to authenticate by using the Run As account. The runbook lists all resource groups present in the subscription.
AzureRunAsCertificate	A certificate asset that's automatically created when the Automation account is created, or by using a PowerShell script for an existing account. The certificate authenticates with Azure so you can manage Azure Resource Manager resources from runbooks. This certificate has a one-year lifespan.
AzureRunAsConnection	A connection asset that's automatically created when the Automation account is created, or by using a PowerShell script for an existing account.

The following table summarizes resources for the Classic Run As account.

RESOURCE	DESCRIPTION
AzureClassicAutomationTutorial Runbook	An example graphical runbook. The runbook gets all classic VMs in a subscription by using the Classic Run As Account (certificate). Then, it displays the VM names and status.
AzureClassicAutomationTutorial Script Runbook	An example PowerShell runbook. The runbook gets all classic VMs in a subscription by using the Classic Run As Account (certificate). Then, it displays the VM names and status.
AzureClassicRunAsCertificate	A certificate asset that's automatically created. The certificate authenticates with Azure so you can manage Azure classic resources from runbooks. This certificate has a one-year lifespan.
AzureClassicRunAsConnection	A connection asset that's automatically created. The asset authenticates with Azure so you can manage Azure classic resources from runbooks.

Next steps

- To learn more about graphical authoring, see [Graphical authoring in Azure Automation](#).
- To get started with PowerShell runbooks, see [My first PowerShell runbook](#).
- To get started with PowerShell workflow runbooks, see [My first PowerShell workflow runbook](#).
- To get started with Python2 runbooks, see [My first Python2 runbook](#).

Authenticate Runbooks with Azure classic deployment and Resource Manager

5/21/2018 • 2 minutes to read • [Edit Online](#)

This article describes the steps you must perform to configure an Azure AD User account for Azure Automation runbooks running against Azure classic deployment model or Azure Resource Manager resources. While this continues to be a supported authentication identity for your Azure Resource Manager based runbooks, the recommended method is to use an Azure Run As account.

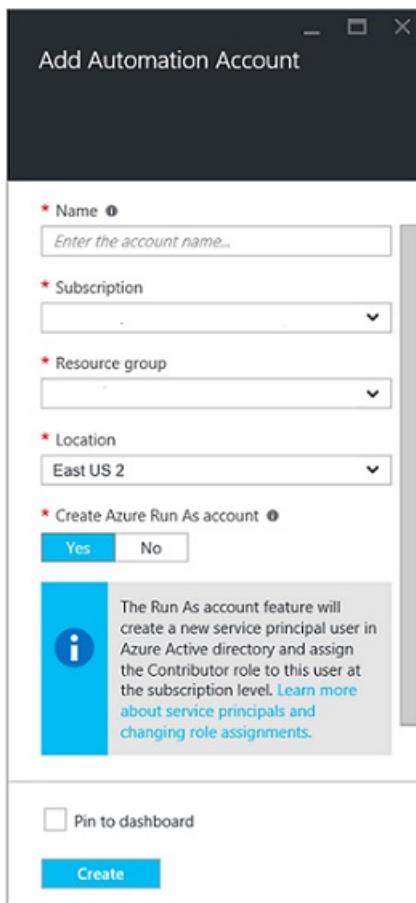
Create a new Azure Active Directory user

1. Log in to the Azure portal as a service administrator for the Azure subscription you want to manage.
2. Select **Azure Active Directory** > **Users and groups** > **All users** > **New user**.
3. Enter details for the user, like **Name** and **User name**.
4. Note the user's full name and temporary password.
5. Select **Directory role**.
6. Assign role Global or Limited Administrator.
7. Log out of Azure and then log back in with the account you just created. You are prompted to change the user's password.

Create an Automation account in the Azure portal

In this section, perform the following steps to create an Azure Automation account in the Azure portal for use with your runbooks managing resources in Azure Resource Manager mode.

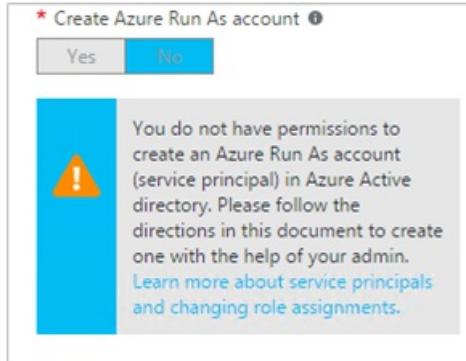
1. Log in to the Azure portal as a service administrator for the Azure subscription you want to manage.
2. Select **Automation Accounts**.
3. Select **Add**.



4. In the **Add Automation Account** blade, in the **Name** box type in a name for your new Automation account.
5. If you have more than one subscription, specify the one for the new account, as well as a new or existing **Resource group** and an Azure datacenter **Location**.
6. Select the value **Yes** for the **Create Azure Run As account** option, and click the **Create** button.

NOTE

If you choose to not create the Run As account by selecting the option **No**, you are presented with a warning message in the **Add Automation Account** blade. While the account is created and assigned to the **Contributor** role in the subscription, it does not have a corresponding authentication identity within your subscriptions directory service and therefore, no access resources in your subscription. This prevents any runbooks referencing this account from being able to authenticate and perform tasks against Azure Resource Manager resources.



7. While Azure creates the Automation account, you can track the progress under **Notifications** from the menu.

When the creation of the credential is completed, you need to create a Credential Asset to associate the Automation Account with the AD User account created earlier. Remember, you only created the Automation account and it is not associated with an authentication identity. Perform the steps outlined in the [Credential assets](#)

in Azure Automation article and enter the value for **username** in the format **domain\user**.

Use the credential in a runbook

You can retrieve the credential in a runbook using the [Get-AutomationPSCredential](#) activity and then use it with [Add-AzureAccount](#) to connect to your Azure subscription. If the credential is an administrator of multiple Azure subscriptions, then you should also use [Select-AzureSubscription](#) to specify the correct one. This is shown in the following PowerShell sample that typically appears at the top of most Azure Automation runbooks.

```
$cred = Get-AutomationPSCredential -Name "myuseraccount.onmicrosoft.com"
Add-AzureAccount -Credential $cred
Select-AzureSubscription -SubscriptionName "My Subscription"
```

Repeat these lines after any [checkpoints](#) in your runbook. If the runbook is suspended and then resumes on another worker, then it needs to perform the authentication again.

Next Steps

- Review the different runbook types and steps for creating your own runbooks from the following article [Azure Automation runbook types](#)

Authenticate Runbooks with Amazon Web Services

5/21/2018 • 2 minutes to read • [Edit Online](#)

Automating common tasks with resources in Amazon Web Services (AWS) can be accomplished with Automation runbooks in Azure. You can automate many tasks in AWS using Automation runbooks just like you can with resources in Azure. All that is required are two things:

- An AWS subscription and a set of credentials. Specifically your AWS Access Key and Secret Key. For more information, review the article [Using AWS Credentials](#).
- An Azure subscription and Automation account.

To authenticate with AWS, you must specify a set of AWS credentials to authenticate your runbooks running from Azure Automation. If you already have an Automation account created and you want to use that to authenticate with AWS, you can follow the steps in the following section: If you want to dedicate an account for runbooks targeting AWS resources, you should first create a new [Automation account](#) (skip the option to create a service principal) and use the following steps:

Configure Automation account

For Azure Automation to communicate with AWS, you first need to retrieve your AWS credentials and store them as assets in Azure Automation. Perform the following steps documented in the AWS document [Managing Access Keys for your AWS Account](#) to create an Access Key and copy the **Access Key ID** and **Secret Access Key** (optionally download your key file to store it somewhere safe).

After you have created and copied your AWS security keys, you need to create a Credential asset with an Azure Automation account to securely store them and reference them with your runbooks. Follow the steps in the section: **To create a new credential** in the [Credential assets in Azure Automation](#) article and enter the following information:

1. In the **Name** box, enter **AWScred** or an appropriate value following your naming standards.
2. In the **User name** box, type your **Access ID** and your **Secret Access Key** in the **Password** and **Confirm password** box.

Next steps

- Review the solution article [Automating deployment of a VM in Amazon Web Services](#) to learn how to create runbooks to automate tasks in AWS.

Update your Automation account authentication with Run As accounts

7/3/2018 • 11 minutes to read • [Edit Online](#)

You can update your existing Automation account from the Azure portal or use PowerShell if:

- You create an Automation account but do not create the Run As account.
- You already use an Automation account to manage Resource Manager resources and you want to update the account to include the Run As account for runbook authentication.
- You already use an Automation account to manage classic resources and you want to update it to use the Classic Run As account instead of creating a new account and migrating your runbooks and assets to it.

The process creates the following items in your Automation account.

For Run As accounts:

- Creates an Azure AD application with a self-signed certificate, creates a service principal account for the application in Azure AD, and assigns the Contributor role for the account in your current subscription. You can change this setting to Owner or any other role. For more information, see [Role-based access control in Azure Automation](#).
- Creates an Automation certificate asset named *AzureRunAsCertificate* in the specified Automation account. The certificate asset holds the certificate private key that's used by the Azure AD application.
- Creates an Automation connection asset named *AzureRunAsConnection* in the specified Automation account. The connection asset holds the applicationId, tenantId, subscriptionId, and certificate thumbprint.

For Classic Run As accounts:

- Creates an Automation certificate asset named *AzureClassicRunAsCertificate* in the specified Automation account. The certificate asset holds the certificate private key used by the management certificate.
- Creates an Automation connection asset named *AzureClassicRunAsConnection* in the specified Automation account. The connection asset holds the subscription name, subscriptionId, and certificate asset name.

Prerequisites

If you choose to [use PowerShell to create the Run As accounts](#), this process requires:

- Windows 10 and Windows Server 2016 with Azure Resource Manager modules 3.4.1 and later. The PowerShell script does not support earlier versions of Windows.
- Azure PowerShell 1.0 and later. For information about the PowerShell 1.0 release, see [How to install and configure Azure PowerShell](#).
- An Automation account, which is referenced as the value for the *-AutomationAccountName* and *-ApplicationDisplayName* parameters.

To get the values for *SubscriptionID*, *ResourceGroup*, and *AutomationAccountName*, which are required parameters for the script, do the following:

1. In the Azure portal, click **All services**. In the list of resources, type **Automation**. As you begin typing, the list filters based on your input. Select **Automation Accounts**.
2. On the Automation account page, select your Automation account, and then under **Account Settings** select **Properties**.

3. Note the values on the **Properties** page.

The screenshot shows the Azure portal's Properties page for an Automation Account named "TestAzureAuto". The left sidebar lists various settings: Modules, Modules gallery, Credentials, Connections, Certificates, Variables, Workspace, Unlink workspace, Event grid, Properties (which is selected and highlighted in blue), Keys, Pricing, Source control, and Run as accounts. The main pane displays the following details:

- NAME:** TestAzureAuto
- SUBSCRIPTION:** Microsoft Azure
- SUBSCRIPTION ID:** [Empty input field]
- CREATED:** 11/29/2017 7:55 AM
- LAST MODIFIED:** 1/18/2018 12:19 PM
- LAST MODIFIED BY:** [Empty input field]
- RESOURCE GROUP:** TestAzureAuto
- REGION:** eastus2

Required permissions to update your Automation account

To update an Automation account, you must have the following specific privileges and permissions required to complete this topic.

- Your AD user account must be added to a role with permissions equivalent to the Contributor role for Microsoft.Automation resources as outlined in article [Role-based access control in Azure Automation](#).
- Non-admin users in your Azure AD tenant can [register AD applications](#) if the Azure AD tenant's **Users can register applications** option in **User settings** page is set to **Yes**. If the app registrations setting is set to **No**, the user performing this action must be a global administrator in Azure AD.

If you are not a member of the subscription's Active Directory instance before you are added to the global administrator/co-administrator role of the subscription, you are added to Active Directory as a guest. In this situation, you receive a "You do not have permissions to create..." warning on the **Add Automation Account** blade. Users who were added to the global administrator/co-administrator role first can be removed from the

subscription's Active Directory instance and readded to make them a full User in Active Directory. To verify this situation, from the **Azure Active Directory** pane in the Azure portal, select **Users and groups**, select **All users** and, after you select the specific user, select **Profile**. The value of the **User type** attribute under the users profile should not equal **Guest**.

Create Run As account from the portal

In this section, perform the following steps to update your Azure Automation account in the Azure portal. You create the Run As and Classic Run As accounts individually. If you don't need to manage classic resources, you can just create the Azure Run As account.

1. Sign in to the Azure portal with an account that is a member of the Subscription Admins role and co-administrator of the subscription.
2. In the Azure portal, click **All services**. In the list of resources, type **Automation**. As you begin typing, the list filters based on your input. Select **Automation Accounts**.
3. On the **Automation Accounts** page, select your Automation account from the list of Automation accounts.
4. In the left-hand pane, select **Run As Accounts** under the section **Account Settings**.
5. Depending on which account you require, select either **Azure Run As Account** or **Azure Classic Run As Account**. After selecting either the **Add Azure Run As** or **Add Azure Classic Run As Account** pane appears and after reviewing the overview information, click **Create** to proceed with Run As account creation.
6. While Azure creates the Run As account, you can track the progress under **Notifications** from the menu. A banner is also displayed stating the account is being created. This process can take a few minutes to complete.

Create Run As account using PowerShell script

This PowerShell script includes support for the following configurations:

- Create a Run As account by using a self-signed certificate.
- Create a Run As account and a Classic Run As account by using a self-signed certificate.
- Create a Run As account and a Classic Run As account by using a certificate issued by your enterprise certification authority (CA).
- Create a Run As account and a Classic Run As account by using a self-signed certificate in the Azure Government cloud.

NOTE

If you select either option for creating a Classic Run As account, after the script is executed, upload the public certificate (.cer file name extension) to the management store for the subscription that the Automation account was created in.

1. Save the following script on your computer. In this example, save it with the filename *New-RunAsAccount.ps1*.

```
#Requires -RunAsAdministrator
Param (
    [Parameter(Mandatory = $true)]
    [String] $ResourceGroup,
    [Parameter(Mandatory = $true)]
    [String] $AutomationAccountName,
    [Parameter(Mandatory = $true)]
    [String] $ApplicationDisplayName,
    [Parameter(Mandatory = $true)]
    [String] $SubscriptionId,
```

```

[Parameter(Mandatory = $true)]
[Boolean] $CreateClassicRunAsAccount,

[Parameter(Mandatory = $true)]
[String] $SelfSignedCertPlainPassword,

[Parameter(Mandatory = $false)]
[String] $EnterpriseCertPathForRunAsAccount,

[Parameter(Mandatory = $false)]
[String] $EnterpriseCertPlainPasswordForRunAsAccount,

[Parameter(Mandatory = $false)]
[String] $EnterpriseCertPathForClassicRunAsAccount,

[Parameter(Mandatory = $false)]
[String] $EnterpriseCertPlainPasswordForClassicRunAsAccount,

[Parameter(Mandatory = $false)]
[ValidateSet("AzureCloud", "AzureUSGovernment")]
[string]$EnvironmentName = "AzureCloud",

[Parameter(Mandatory = $false)]
[int] $SelfSignedCertNoOfMonthsUntilExpired = 12
)

function CreateSelfSignedCertificate([string] $certificateName, [string] $selfSignedCertPlainPassword,
[string] $certPath, [string] $certPathCer, [string] $selfSignedCertNoOfMonthsUntilExpired ) {
$Cert = New-SelfSignedCertificate -DnsName $certificateName -CertStoreLocation cert:\LocalMachine\My
`-KeyExportPolicy Exportable -Provider "Microsoft Enhanced RSA and AES Cryptographic Provider" `

-NotAfter (Get-Date).AddMonths($selfSignedCertNoOfMonthsUntilExpired) -HashAlgorithm SHA256

$CertPassword = ConvertTo-SecureString $selfSignedCertPlainPassword -AsPlainText -Force
Export-PfxCertificate -Cert ("Cert:\localmachine\my\" + $Cert.Thumbprint) -FilePath $certPath -
Password $CertPassword -Force | Write-Verbose
Export-Certificate -Cert ("Cert:\localmachine\my\" + $Cert.Thumbprint) -FilePath $certPathCer -Type
CERT | Write-Verbose
}

function CreateServicePrincipal([System.Security.Cryptography.X509Certificates.X509Certificate2]
$PfxCert, [string] $applicationDisplayName) {
$keyValue = [System.Convert]::ToBase64String($PfxCert.GetRawCertData())
$keyId = (New-Guid).Guid

#Create an Azure AD application, AD App Credential, AD ServicePrincipal
$Application = New-AzureRmADApplication -DisplayName $ApplicationDisplayName -HomePage ("http://" +
$applicationDisplayName) -IdentifierUris ("http://" + $keyId)
$ApplicationCredential = New-AzureRmADAppCredential -ApplicationId $Application.ApplicationId -
CertValue $keyValue -StartDate $PfxCert.NotBefore -EndDate $PfxCert.NotAfter
$ServicePrincipal = New-AzureRMServicePrincipal -ApplicationId $Application.ApplicationId
$GetServicePrincipal = Get-AzureRmADServicePrincipal -ObjectId $ServicePrincipal.Id

# Sleep here for a few seconds to allow the service principal application to become active
(ordinarily takes a few seconds)
Sleep -s 15
$NewRole = New-AzureRMRoleAssignment -RoleDefinitionName Contributor -ServicePrincipalName
$Application.ApplicationId -ErrorAction SilentlyContinue
$Retries = 0;
While ($NewRole -eq $null -and $Retries -le 6) {
    Sleep -s 10
    New-AzureRMRoleAssignment -RoleDefinitionName Contributor -ServicePrincipalName
$Application.ApplicationId | Write-Verbose -ErrorAction SilentlyContinue
    $NewRole = Get-AzureRMRoleAssignment -ServicePrincipalName $Application.ApplicationId -
ErrorAction SilentlyContinue
    $Retries++;
}
return $Application.ApplicationId.ToString();

```

```

}

function CreateAutomationCertificateAsset ([string] $resourceGroup, [string] $automationAccountName,
[string] $certificateAssetName, [string] $certPath, [string] $certPlainPassword, [Boolean] $Exportable) {
    $CertPassword = ConvertTo-SecureString $certPlainPassword -AsPlainText -Force
    Remove-AzureRmAutomationCertificate -ResourceGroupName $resourceGroup -AutomationAccountName
$automationAccountName -Name $certificateAssetName -ErrorAction SilentlyContinue
    New-AzureRmAutomationCertificate -ResourceGroupName $resourceGroup -AutomationAccountName
$automationAccountName -Path $certPath -Name $certificateAssetName -Password $CertPassword -
Exportable:$Exportable | write-verbose
}

function CreateAutomationConnectionAsset ([string] $resourceGroup, [string] $automationAccountName,
[string] $connectionAssetName, [string] $connectionTypeName, [System.Collections.Hashtable]
$connectionFieldValues ) {
    Remove-AzureRmAutomationConnection -ResourceGroupName $resourceGroup -AutomationAccountName
$automationAccountName -Name $connectionAssetName -Force -ErrorAction SilentlyContinue
    New-AzureRmAutomationConnection -ResourceGroupName $ResourceGroup -AutomationAccountName
$automationAccountName -Name $connectionAssetName -ConnectionTypeName $connectionTypeName -
ConnectionFieldValues $connectionFieldValues
}

Import-Module AzureRM.Profile
Import-Module AzureRM.Resources

$AzureRMProfileVersion = (Get-Module AzureRM.Profile).Version
if (!((($AzureRMProfileVersion.Major -ge 3 -and $AzureRMProfileVersion.Minor -ge 4) -or
($AzureRMProfileVersion.Major -gt 3))) {
    Write-Error -Message "Please install the latest Azure PowerShell and retry. Relevant doc url :
https://docs.microsoft.com/powershell/azureps-cmdlets-docs/"
    return
}

Connect-AzureRmAccount -Environment $EnvironmentName
$Subscription = Select-AzureRmSubscription -SubscriptionId $SubscriptionId

# Create a Run As account by using a service principal
$CertificateAssetName = "AzureRunAsCertificate"
$ConnectionAssetName = "AzureRunAsConnection"
$ConnectionTypeName = "AzureServicePrincipal"

if ($EnterpriseCertPathForRunAsAccount -and $EnterpriseCertPlainPasswordForRunAsAccount) {
    $PfxCertPathForRunAsAccount = $EnterpriseCertPathForRunAsAccount
    $PfxCertPlainPasswordForRunAsAccount = $EnterpriseCertPlainPasswordForRunAsAccount
}
else {
    $CertificateName = $AutomationAccountName + $CertificateAssetName
    $PfxCertPathForRunAsAccount = Join-Path $env:TEMP ($CertificateName + ".pfx")
    $PfxCertPlainPasswordForRunAsAccount = $SelfSignedCertPlainPassword
    $CerCertPathForRunAsAccount = Join-Path $env:TEMP ($CertificateName + ".cer")
    CreateSelfSignedCertificate $CertificateName $PfxCertPlainPasswordForRunAsAccount
    $PfxCertPathForRunAsAccount $CerCertPathForRunAsAccount $SelfSignedCertNoOfMonthsUntilExpired
}

# Create a service principal
$PfxCert = New-Object -TypeName System.Security.Cryptography.X509Certificates.X509Certificate2 -
ArgumentList @($PfxCertPathForRunAsAccount, $PfxCertPlainPasswordForRunAsAccount)
$ApplicationId = CreateServicePrincipal $PfxCert $ApplicationDisplayName

# Create the Automation certificate asset
CreateAutomationCertificateAsset $ResourceGroup $AutomationAccountName $CertificateAssetName
$PfxCertPathForRunAsAccount $PfxCertPlainPasswordForRunAsAccount $true

# Populate the ConnectionFieldValues
$SubscriptionInfo = Get-AzureRmSubscription -SubscriptionId $SubscriptionId
$TenantID = $SubscriptionInfo | Select TenantId -First 1
$Thumbprint = $PfxCert.Thumbprint
$ConnectionFieldValues = @{"ApplicationId" = $ApplicationId; "TenantId" = $TenantID.TenantId;
"CertificateThumbprint" = $Thumbprint; "SubscriptionId" = $SubscriptionId}

```

```

# Create an Automation connection asset named AzureRunAsConnection in the Automation account. This
connection uses the service principal.
CreateAutomationConnectionAsset $ResourceGroup $AutomationAccountName $ConnectionAssetName
$ConnectionTypeName $ConnectionFieldValues

if ($CreateClassicRunAsAccount) {
    # Create a Run As account by using a service principal
    $ClassicRunAsAccountCertificateAssetName = "AzureClassicRunAsCertificate"
    $ClassicRunAsAccountConnectionAssetName = "AzureClassicRunAsConnection"
    $ClassicRunAsAccountConnectionTypeName = "AzureClassicCertificate "
    $UploadMessage = "Please upload the .cer format of #CERT# to the Management store by following the
steps below." + [Environment]::NewLine +
    "Log in to the Microsoft Azure portal (https://portal.azure.com) and select Subscriptions ->
Management Certificates." + [Environment]::NewLine +
    "Then click Upload and upload the .cer format of #CERT#"

    if ($EnterpriseCertPathForClassicRunAsAccount -and
$EnterpriseCertPlainPasswordForClassicRunAsAccount ) {
        $PfxCertPathForClassicRunAsAccount = $EnterpriseCertPathForClassicRunAsAccount
        $PfxCertPlainPasswordForClassicRunAsAccount = $EnterpriseCertPlainPasswordForClassicRunAsAccount
        $UploadMessage = $UploadMessage.Replace("#CERT#", $PfxCertPathForClassicRunAsAccount)
    }
    else {
        $ClassicRunAsAccountCertificateName = $AutomationAccountName +
$ClassicRunAsAccountCertificateAssetName
        $PfxCertPathForClassicRunAsAccount = Join-Path $env:TEMP ($ClassicRunAsAccountCertificateName +
".pfx")
        $PfxCertPlainPasswordForClassicRunAsAccount = $SelfSignedCertPlainPassword
        $CerCertPathForClassicRunAsAccount = Join-Path $env:TEMP ($ClassicRunAsAccountCertificateName +
".cer")
        $UploadMessage = $UploadMessage.Replace("#CERT#", $CerCertPathForClassicRunAsAccount)
        CreateSelfSignedCertificate $ClassicRunAsAccountCertificateName
$PfxCertPlainPasswordForClassicRunAsAccount $PfxCertPathForClassicRunAsAccount
$CerCertPathForClassicRunAsAccount $SelfSignedCertNoOfMonthsUntilExpired
    }

    # Create the Automation certificate asset
    CreateAutomationCertificateAsset $ResourceGroup $AutomationAccountName
$ClassicRunAsAccountCertificateAssetName $PfxCertPathForClassicRunAsAccount
$PfxCertPlainPasswordForClassicRunAsAccount $false

    # Populate the ConnectionFieldValues
    $SubscriptionName = $subscription.Subscription.Name
    $ClassicRunAsAccountConnectionFieldValues = @{
        "SubscriptionName" = $SubscriptionName;
        "SubscriptionId" = $SubscriptionId; "CertificateAssetName" = $ClassicRunAsAccountCertificateAssetName
    }

    # Create an Automation connection asset named AzureRunAsConnection in the Automation account. This
connection uses the service principal.
    CreateAutomationConnectionAsset $ResourceGroup $AutomationAccountName
$ClassicRunAsAccountConnectionAssetName $ClassicRunAsAccountConnectionTypeName
$ClassicRunAsAccountConnectionFieldValues

    Write-Host -ForegroundColor red      $UploadMessage
}

```

IMPORTANT

Add-AzureRmAccount is now an alias for **Connect-AzureRMAccount**. When searching your library items, if you do not see **Connect-AzureRMAccount**, you can use **Add-AzureRmAccount**, or you can update your modules in your Automation Account.

2. On your computer, start **Windows PowerShell** from the **Start** screen with elevated user rights.
3. From the elevated command-line shell, go to the folder that contains the script you created in step 1.

4. Execute the script by using the parameter values for the configuration you require.

Create a Run As account by using a self-signed certificate

```
.\New-RunAsAccount.ps1 -ResourceGroup <ResourceGroupName> -AutomationAccountName  
<NameofAutomationAccount> -SubscriptionId <SubscriptionId> -ApplicationDisplayName  
<DisplayNameofAADApplication> -SelfSignedCertPlainPassword <StrongPassword> -CreateClassicRunAsAccount  
$false
```

Create a Run As account and a Classic Run As account by using a self-signed certificate

```
.\New-RunAsAccount.ps1 -ResourceGroup <ResourceGroupName> -AutomationAccountName  
<NameofAutomationAccount> -SubscriptionId <SubscriptionId> -ApplicationDisplayName  
<DisplayNameofAADApplication> -SelfSignedCertPlainPassword <StrongPassword> -CreateClassicRunAsAccount  
$true
```

Create a Run As account and a Classic Run As account by using an enterprise certificate

```
.\New-RunAsAccount.ps1 -ResourceGroup <ResourceGroupName> -AutomationAccountName  
<NameofAutomationAccount> -SubscriptionId <SubscriptionId> -ApplicationDisplayName  
<DisplayNameofAADApplication> -SelfSignedCertPlainPassword <StrongPassword> -CreateClassicRunAsAccount  
$true -EnterpriseCertPathForRunAsAccount <EnterpriseCertPfxPathForRunAsAccount> -  
EnterpriseCertPlainPasswordForRunAsAccount <StrongPassword> -EnterpriseCertPathForClassicRunAsAccount  
<EnterpriseCertPfxPathForClassicRunAsAccount> -EnterpriseCertPlainPasswordForClassicRunAsAccount  
<StrongPassword>
```

Create a Run As account and a Classic Run As account by using a self-signed certificate in the Azure Government cloud

```
.\New-RunAsAccount.ps1 -ResourceGroup <ResourceGroupName> -AutomationAccountName  
<NameofAutomationAccount> -SubscriptionId <SubscriptionId> -ApplicationDisplayName  
<DisplayNameofAADApplication> -SelfSignedCertPlainPassword <StrongPassword> -CreateClassicRunAsAccount  
$true -EnvironmentName AzureUSGovernment
```

NOTE

After the script has executed, you will be prompted to authenticate with Azure. Sign in with an account that is a member of the subscription administrators role and co-administrator of the subscription.

After the script has executed successfully, note the following:

- If you created a Classic Run As account with a self-signed public certificate (.cer file), the script creates and saves it to the temporary files folder on your computer under the user profile %USERPROFILE%\AppData\Local\Temp, which you used to execute the PowerShell session.
- If you created a Classic Run As account with an enterprise public certificate (.cer file), use this certificate. Follow the instructions for [uploading a management API certificate to the Azure portal](#), and then validate the credential configuration with classic deployment resources by using the [sample code to authenticate with Azure Classic Deployment Resources](#).
- If you did *not* create a Classic Run As account, authenticate with Resource Manager resources and validate the credential configuration by using the [sample code for authenticating with Service Management resources](#).

Limiting Run As account permissions

To control targeting of automation against resources in Azure Automation, the Run As account by default is granted contributor rights in the subscription. If you need to restrict what the RunAs service principal can do, you can remove the account from the contributor role to the subscription and add it as a contributor to the resource groups you want to specify.

In the Azure portal, select **Subscriptions** and choose the subscription of your Automation Account. Select **Access control (IAM)** and search for the service principal for your Automation Account (it looks like <AutomationAccountName>_unique identifier). Select the account and click **Remove** to remove it from the subscription.

The screenshot shows the Microsoft Azure - Access control (IAM) interface. On the left, there's a navigation sidebar with links like Overview, Access control (IAM), Diagnose and solve problems, COST MANAGEMENT + BILLING, Partner information, SETTINGS, Programmatic deployment, Resource groups, and Resources. The main area has a search bar at the top. Below it, there are four filter dropdowns: Name (TestAzureAuto), Type (All), Role (3 selected), and Scope (All scopes). A message indicates a filtered set of results with a total of 18 assignments. A table below shows one item: TestAzureAuto_D2THqif... (App) assigned the Contributor role to This resource.

NAME	TYPE	ROLE	SCOPE
TestAzureAuto_D2THqif...	App	Contributor	This resource

To add the service principal to a resource group, select the resource group in the Azure portal and select **Access control (IAM)**. Select **Add**, this opens the **Add permissions** page. For **Role**, select **Contributor**. In the **Select** text box type in the name of the service principal for your Run As account, and select it from the list. Click **Save** to save the changes. Do this for the resources groups you want to give your Azure Automation Run As service principal access to.

Next steps

- For more information about Service Principals, see [Application Objects and Service Principal Objects](#).
- For more information about certificates and Azure services, see [Certificates overview for Azure Cloud Services](#).

Test Azure Automation Run As account authentication

7/3/2018 • 3 minutes to read • [Edit Online](#)

After an Automation account is successfully created, you can perform a simple test to confirm you are able to successfully authenticate in Azure Resource Manager or Azure classic deployment using your newly created or updated Automation Run As account.

Automation Run As authentication

Use the sample code below to [create a PowerShell runbook](#) to verify authentication using the Run As account and also in your custom runbooks to authenticate and manage Resource Manager resources with your Automation account.

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

    "Logging in to Azure..."
    Connect-AzureRmAccount ` 
        -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
    }
}

#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 ("")
}
```

Notice the cmdlet used for authenticating in the runbook - **Connect-AzureRmAccount**, uses the *ServicePrincipalCertificate* parameter set. It authenticates by using service principal certificate, not credentials.

IMPORTANT

Add-AzureRmAccount is now an alias for **Connect-AzureRMAccount**. When searching your library items, if you do not see **Connect-AzureRMAccount**, you can use **Add-AzureRmAccount**, or you can update your modules in your Automation Account.

When you [run the runbook](#) to validate your Run As account, a [runbook job](#) is created, the job page is displayed, and the job status displayed in the **Job Summary** tile. The job status will start as *Queued* indicating that it is waiting for a runbook worker in the cloud to become available. It will then move to *Starting* when a worker claims the job, and then *Running* when the runbook actually starts running. When the runbook job completes, we should see a status of **Completed**.

To see the detailed results of the runbook, click on the **Output** tile. On the **Output** page, you should see it has successfully authenticated and returns a list of all resources in all resource groups in your subscription.

Just remember to remove the block of code starting with the comment

```
#Get all ARM resources from all resource groups
```

 when you reuse the code for your runbooks.

Classic Run As authentication

Use the sample code below to [create a PowerShell runbook](#) to verify authentication using the Classic Run As account and also in your custom runbooks to authenticate and manage resources in the classic deployment model.

```
$ConnectionAssetName = "AzureClassicRunAsConnection"
# Get the connection
$connection = Get-AutomationConnection -Name $connectionAssetName

# Authenticate to Azure with certificate
Write-Verbose "Get connection asset: $ConnectionAssetName" -Verbose
$Conn = Get-AutomationConnection -Name $ConnectionAssetName
if ($Conn -eq $null)
{
    throw "Could not retrieve connection asset: $ConnectionAssetName. Assure that this asset exists in the Automation account."
}

$CertificateAssetName = $Conn.CertificateAssetName
Write-Verbose "Getting the certificate: $CertificateAssetName" -Verbose
$AzureCert = Get-AutomationCertificate -Name $CertificateAssetName
if ($AzureCert -eq $null)
{
    throw "Could not retrieve certificate asset: $CertificateAssetName. Assure that this asset exists in the Automation account."
}

Write-Verbose "Authenticating to Azure with certificate." -Verbose
Set-AzureSubscription -SubscriptionName $Conn.SubscriptionName -SubscriptionId $Conn.SubscriptionID -
Certificate $AzureCert
Select-AzureSubscription -SubscriptionId $Conn.SubscriptionID

#Get all VMs in the subscription and return list with name of each
Get-AzureVM | ft Name
```

When you [run the runbook](#) to validate your Run As account, a [runbook job](#) is created, the Job page is displayed, and the job status displayed in the **Job Summary** tile. The job status will start as *Queued* indicating that it is waiting for a runbook worker in the cloud to become available. It will then move to *Starting* when a worker claims the job, and then *Running* when the runbook actually starts running. When the runbook job completes, we should see a status of **Completed**.

To see the detailed results of the runbook, click on the **Output** tile. On the **Output** page, you should see it has successfully authenticated and returns a list of all Azure VMs by VMName that are deployed in your subscription.

Just remember to remove the cmdlet **Get-AzureVM** when you reuse the code for your runbooks.

Next steps

- To get started with PowerShell runbooks, see [My first PowerShell runbook](#).
- To learn more about Graphical Authoring, see [Graphical authoring in Azure Automation](#).

Role-based access control in Azure Automation

6/14/2018 • 13 minutes to read • [Edit Online](#)

Role-based access control (RBAC) enables access management for Azure resources. Using [RBAC](#), you can segregate duties within your team and grant only the amount of access to users, groups, and applications that they need to perform their jobs. Role-based access can be granted to users using the Azure portal, Azure Command-Line tools, or Azure Management APIs.

Roles in Automation accounts

In Azure Automation, access is granted by assigning the appropriate RBAC role to users, groups, and applications at the Automation account scope. Following are the built-in roles supported by an Automation account:

ROLE	DESCRIPTION
Owner	The Owner role allows access to all resources and actions within an Automation account including providing access to other users, groups, and applications to manage the Automation account.
Contributor	The Contributor role allows you to manage everything except modifying other user's access permissions to an Automation account.
Reader	The Reader role allows you to view all the resources in an Automation account but cannot make any changes.
Automation Operator	The Automation Operator role allows you to view runbook name and properties and to create and manage jobs for all runbooks in an Automation account. This role is helpful if you want to protect your Automation Account resources like credentials assets and runbooks from being viewed or modified but still allow members of your organization to execute these runbooks.
Automation Job Operator	The Automation Job Operator role allows you to create and manage jobs for all runbooks in an Automation account.
Automation Runbook Operator	The Automation Runbook Operator role allows you to view a runbook's name and properties.
Log Analytics Contributor	The Log Analytics Contributor role allows you to read all monitoring data and edit monitoring settings. Editing monitoring settings includes adding the VM extension to VMs, reading storage account keys to be able to configure collection of logs from Azure storage, creating and configuring Automation accounts, adding solutions, and configuring Azure diagnostics on all Azure resources.
Log Analytics Reader	The Log Analytics Reader role allows you to view and search all monitoring data as well as view monitoring settings. This includes viewing the configuration of Azure diagnostics on all Azure resources.

ROLE	DESCRIPTION
Monitoring Contributor	The Monitoring Contributor role allows you to read all monitoring data and update monitoring settings.
Monitoring Reader	The Monitoring Reader role allows you to read all monitoring data.
User Access Administrator	The User Access Administrator role allows you to manage user access to Azure Automation accounts.

Role permissions

The following tables describe the specific permissions given to each role. This can include Actions, which give permissions, and NotActions, which restrict them.

Owner

An Owner can manage everything, including access. The following table shows the permissions granted for the role:

ACTIONS	DESCRIPTION
Microsoft.Automation/automationAccounts/	Create and manage resources of all types.

Contributor

A Contributor can manage everything except access. The following table shows the permissions granted and denied for the role:

ACTIONS	DESCRIPTION
Microsoft.Automation/automationAccounts/	Create and manage resources of all types
Not Actions	
Microsoft.Authorization/*/Delete	Delete roles and role assignments.
Microsoft.Authorization/*/Write	Create roles and role assignments.
Microsoft.Authorization/elevateAccess/Action	Denies the ability to create a User Access Administrator.

Reader

A Reader can view all the resources in an Automation account but cannot make any changes.

ACTIONS	DESCRIPTION
Microsoft.Automation/automationAccounts/read	View all resources in an Automation account.

Automation Operator

An Automation Operator is able to create and manage jobs, and read runbook names and properties for all runbooks in an Automation account. Note: If you want to control operator access to individual runbooks then don't set this role, and instead use the 'Automation Job Operator' and 'Automation Runbook Operator' roles in combination. The following table shows the permissions granted for the role:

ACTIONS	DESCRIPTION
Microsoft.Authorization/*/read	Read authorization.
Microsoft.Automation/automationAccounts/hybridRunbookWorkerGroups/read	Read Hybrid Runbook Worker Resources.
Microsoft.Automation/automationAccounts/jobs/read	List jobs of the runbook.
Microsoft.Automation/automationAccounts/jobs/resume/action	Resume a job that is paused.
Microsoft.Automation/automationAccounts/jobs/stop/action	Cancel a job in progress.
Microsoft.Automation/automationAccounts/jobsstreams/read	Read the Job Streams and Output.
Microsoft.Automation/automationAccounts/jobs/output/read	Get the Output of a job.
Microsoft.Automation/automationAccounts/jobs/suspend/action	Pause a job in progress.
Microsoft.Automation/automationAccounts/jobs/write	Create jobs.
Microsoft.Automation/automationAccounts/jobSchedules/read	Get an Azure Automation job schedule.
Microsoft.Automation/automationAccounts/jobSchedules/write	Create an Azure Automation job schedule.
Microsoft.Automation/automationAccounts/linkedWorkspace/read	Get the workspace linked to the automation account.
Microsoft.Automation/automationAccounts/read	Get an Azure Automation account.
Microsoft.Automation/automationAccounts/runbooks/read	Get an Azure Automation runbook.
Microsoft.Automation/automationAccounts/schedules/read	Get an Azure Automation schedule asset.
Microsoft.Automation/automationAccounts/schedules/write	Create or update an Azure Automation schedule asset.
Microsoft.Resources/subscriptions/resourceGroups/read	Read roles and role assignments.
Microsoft.Resources/deployments/*	Create and manage resource group deployments.
Microsoft.Insights/alertRules/*	Create and manage alert rules.
Microsoft.Support/*	Create and manage support tickets.

Automation Job Operator

An Automation Job Operator role is granted at the Automation account scope. This allows the operator permissions to create and manage jobs for all runbooks in the account. The following table shows the permissions granted for the role:

ACTIONS	DESCRIPTION
Microsoft.Authorization/*/read	Read authorization.
Microsoft.Automation/automationAccounts/jobs/read	List jobs of the runbook.
Microsoft.Automation/automationAccounts/jobs/resume/action	Resume a job that is paused.
Microsoft.Automation/automationAccounts/jobs/stop/action	Cancel a job in progress.
Microsoft.Automation/automationAccounts/jobsstreams/read	Read the Job Streams and Output.
Microsoft.Automation/automationAccounts/jobs/suspend/action	Pause a job in progress.
Microsoft.Automation/automationAccounts/jobs/write	Create jobs.
Microsoft.Resources/subscriptions/resourceGroups/read	Read roles and role assignments.
Microsoft.Resources/deployments/*	Create and manage resource group deployments.
Microsoft.Insights/alertRules/*	Create and manage alert rules.
Microsoft.Support/*	Create and manage support tickets.

Automation Runbook Operator

An Automation Runbook Operator role is granted at the Runbook scope. An Automation Runbook Operator can view the runbook's name and properties. This role combined with the 'Automation Job Operator' role enables the operator to also create and manage jobs for the runbook. The following table shows the permissions granted for the role:

ACTIONS	DESCRIPTION
Microsoft.Automation/automationAccounts/runbooks/read	List the runbooks.
Microsoft.Authorization/*/read	Read authorization.
Microsoft.Resources/subscriptions/resourceGroups/read	Read roles and role assignments.
Microsoft.Resources/deployments/*	Create and manage resource group deployments.
Microsoft.Insights/alertRules/*	Create and manage alert rules.
Microsoft.Support/*	Create and manage support tickets.

Log Analytics Contributor

A Log Analytics Contributor can read all monitoring data and edit monitoring settings. Editing monitoring settings includes adding the VM extension to VMs; reading storage account keys to be able to configure collection of logs from Azure Storage; creating and configuring Automation accounts; adding solutions; and configuring Azure diagnostics on all Azure resources. The following table shows the permissions granted for the role:

ACTIONS	DESCRIPTION
*/read	Read resources of all types, except secrets.
Microsoft.Automation/automationAccounts/*	Manage automation accounts.
Microsoft.ClassicCompute/virtualMachines/extensions/*	Create and manage virtual machine extensions.
Microsoft.ClassicStorage/storageAccounts/listKeys/action	List classic storage account keys.
Microsoft.Compute/virtualMachines/extensions/*	Create and manage classic virtual machine extensions.
Microsoft.Insights/alertRules/*	Read/write/delete alert rules.
Microsoft.Insights/diagnosticSettings/*	Read/write/delete diagnostic settings.
Microsoft.OperationalInsights/*	Manage Log Analytics.
Microsoft.OperationsManagement/*	Manage solutions in workspaces.
Microsoft.Resources/deployments/*	Create and manage resource group deployments.
Microsoft.Resources/subscriptions/resourcegroups/deployments/*	Create and manage resource group deployments.
Microsoft.Storage/storageAccounts/listKeys/action	List storage account keys.
Microsoft.Support/*	Create and manage support tickets.

Log Analytics Reader

A Log Analytics Reader can view and search all monitoring data as well as and view monitoring settings, including viewing the configuration of Azure diagnostics on all Azure resources. The following table shows the permissions granted or denied for the role:

ACTIONS	DESCRIPTION
*/read	Read resources of all types, except secrets.
Microsoft.OperationalInsights/workspaces/analytics/query/act on	Manage queries in Log Analytics.
Microsoft.OperationalInsights/workspaces/search/action	Search Log Analytics data.
Microsoft.Support/*	Create and manage support tickets.
Not Actions	
Microsoft.OperationalInsights/workspaces/sharedKeys/read	Not able to read the shared access keys.

Monitoring Contributor

A Monitoring Contributor can read all monitoring data and update monitoring settings. The following table shows the permissions granted for the role:

ACTIONS	DESCRIPTION
*/read	Read resources of all types, except secrets.
Microsoft.AlertsManagement/alerts/*	Manage Alerts.
Microsoft.AlertsManagement/alertsSummary/*	Manage the Alert dashboard.
Microsoft.Insights/AlertRules/*	Manage alert rules.
Microsoft.Insights/components/*	Manage Application Insights components.
Microsoft.Insights/DiagnosticSettings/*	Manage diagnostic settings.
Microsoft.Insights/eventtypes/*	List Activity Log events (management events) in a subscription. This permission is applicable to both programmatic and portal access to the Activity Log.
Microsoft.Insights/LogDefinitions/*	This permission is necessary for users who need access to Activity Logs via the portal. List log categories in Activity Log.
Microsoft.Insights/MetricDefinitions/*	Read metric definitions (list of available metric types for a resource).
Microsoft.Insights/Metrics/*	Read metrics for a resource.
Microsoft.Insights/Register/Action	Register the Microsoft.Insights provider.
Microsoft.Insights/webtests/*	Manage Application Insights web tests.
Microsoft.OperationalInsights/workspaces/intelligencepacks/*	Manage Log Analytics solution packs.
Microsoft.OperationalInsights/workspaces/savedSearches/*	Manage Log Analytics saved searches.
Microsoft.OperationalInsights/workspaces/search/action	Search Log Analytics workspaces.
Microsoft.OperationalInsights/workspaces/sharedKeys/action	List keys for a Log Analytics workspace.
Microsoft.OperationalInsights/workspaces/storageinsightconfigs/*	Manage Log Analytics storage insight configurations.
Microsoft.Support/*	Create and manage support tickets.
Microsoft.WorkloadMonitor/workloads/*	Manage Workloads.

Monitoring Reader

A Monitoring Reader can read all monitoring data. The following table shows the permissions granted for the role:

ACTIONS	DESCRIPTION
*/read	Read resources of all types, except secrets.
Microsoft.OperationalInsights/workspaces/search/action	Search Log Analytics workspaces.

ACTIONS	DESCRIPTION
Microsoft.Support/*	Create and manage support tickets

User Access Administrator

A User Access Administrator can manage user access to Azure resources. The following table shows the permissions granted for the role:

ACTIONS	DESCRIPTION
*/read	Read all resources
Microsoft.Authorization/*	Manage authorization
Microsoft.Support/*	Create and manage support tickets

Onboarding

The following tables show the minimum required permissions needed for onboarding virtual machines for the change tracking or update management solutions.

Onboarding from a virtual machine

ACTION	PERMISSION	MINIMUM SCOPE
Write new deployment	Microsoft.Resources/deployments/*	Subscription
Write new resource group	Microsoft.Resources/subscriptions/resourceGroups/write	Subscription
Create new default Workspace	Microsoft.OperationalInsights工作spaces/write	Resource group
Create new Account	Microsoft.Automation/automationAccounts/write	Resource group
Link workspace and account	Microsoft.OperationalInsights工作spaces/write Microsoft.Automation/automationAccounts/read	Workspace Automation account
Create solution	Microsoft.OperationalInsights工作spaces/intelligencepacks/write	Resource group
Create MMA extension	Microsoft.Compute/virtualMachines/write	Virtual Machine
Create saved search	Microsoft.OperationalInsights工作spaces/write	Workspace
Create scope config	Microsoft.OperationalInsights工作spaces/write	Workspace

ACTION	PERMISSION	MINIMUM SCOPE
Link solution to scope config	Microsoft.OperationalInsights/workspaces/intelligencepacks/write	Solution
Onboarding state check - Read workspace	Microsoft.OperationalInsights/workspaces/read	Workspace
Onboarding state check - Read linked workspace property of account	Microsoft.Automation/automationAccounts/read	Automation account
Onboarding state check - Read solution	Microsoft.OperationalInsights/workspaces/intelligencepacks/read	Solution
Onboarding state check - Read VM	Microsoft.Compute/virtualMachines/read	Virtual Machine
Onboarding state check - Read account	Microsoft.Automation/automationAccounts/read	Automation account

Onboarding from Automation account

ACTION	PERMISSION	MINIMUM SCOPE
Create new deployment	Microsoft.Resources/deployments/*	Subscription
Create new resource group	Microsoft.Resources/subscriptions/resourceGroups/write	Subscription
AutomationOnboarding blade - Create new workspace	Microsoft.OperationalInsights/workspaces/write	Resource group
AutomationOnboarding blade - read linked workspace	Microsoft.Automation/automationAccounts/read	Automation account
AutomationOnboarding blade - read solution	Microsoft.OperationalInsights/workspaces/intelligencepacks/read	Solution
AutomationOnboarding blade - read workspace	Microsoft.OperationalInsights/workspaces/intelligencepacks/read	Workspace
Create link for workspace and Account	Microsoft.OperationalInsights/workspaces/write	Workspace
Write account for shoebox	Microsoft.Automation/automationAccounts/write	Account
Create solution	Microsoft.OperationalInsights/workspaces/intelligencepacks/write	Resource Group
Create/edit saved search	Microsoft.OperationalInsights/workspaces/write	Workspace
Create/edit scope config	Microsoft.OperationalInsights/workspaces/write	Workspace

ACTION	PERMISSION	MINIMUM SCOPE
Link solution to scope config	Microsoft.OperationalInsights/workspaces/intelligencepacks/write	Solution
Step 2 - Onboard Multiple VMs		
VMOnboarding blade - Create MMA extension	Microsoft.Compute/virtualMachines/write	Virtual Machine
Create / edit saved search	Microsoft.OperationalInsights/workspaces/write	Workspace
Create / edit scope config	Microsoft.OperationalInsights/workspaces/write	Workspace

Update management

Update management reaches across multiple services to provide its service. The following table shows the permissions needed to manage update management deployments:

RESOURCE	ROLE	SCOPE
Automation account	Log Analytics Contributor	Automation account
Automation account	Virtual Machine Contributor	Resource Group for the account
Log Analytics workspace	Log Analytics Contributor	Log Analytics workspace
Log Analytics workspace	Log Analytics Reader	Subscription
Solution	Log Analytics Contributor	Solution
Virtual Machine	Virtual Machine Contributor	Virtual Machine

Configure RBAC for your Automation account

The following section shows you how to configure RBAC on your Automation Account through the [portal](#) and [PowerShell](#)

Configure RBAC using the Azure portal

1. Log in to the [Azure portal](#) and open your Automation account from the Automation Accounts page.
2. Click on the **Access control (IAM)** control at the top left corner. This opens the **Access control (IAM)** page where you can add new users, groups, and applications to manage your Automation account and view existing roles that can be configured for the Automation account.

The screenshot shows the 'Access control (IAM)' page for an 'Automation Account'. The left sidebar includes links for Overview, Activity log, Access control (IAM), Tags, and Configuration Management (Inventory, Change tracking, DSC nodes, configurations, and node configurations). The main area displays a table of 14 items (2 Users, 12 Service Principals) assigned to the 'Contributor' role with 'Subscription (Inherited)' scope. The table columns are NAME, TYPE, ROLE, and SCOPE.

NAME	TYPE	ROLE	SCOPE
AzureJavaTe...	App	Contributor	Subscription (Inherited)
ContosoAut...	App	Contributor	Subscription (Inherited)
ContosoAut...	App	Contributor	Subscription (Inherited)

Add a new user and assign a role

- From the **Access control (IAM)** page, click **+ Add** to open the **Add permissions** page where you can add a user, group, or application, and assign a role to them.
- Select a role from the list of available roles. You can choose any of the available built-in roles that an Automation account supports or any custom role you may have defined.
- Type the username of the user you want to give permissions to in the **Select** field. Select the user from the list and click **Save**.

The 'Add permissions' dialog box is open. The 'Role' dropdown is set to 'Reader'. The 'Assign access to' dropdown is set to 'Azure AD user, group, or application'. The 'Select' input field contains 'john.doe@example.com'. Below the input field, a list shows a selected member: 'John Doe' (John.Doe@example.com). At the bottom of the dialog are 'Save' and 'Discard' buttons.

Now you should see the user added to the **Users** page with the selected role assigned

Add Remove Roles Refresh Help

Name ?	Type ?	Role ?
<input type="text" value="Search by name or email"/>	All	Reader
Scope ?	Group by ?	
All scopes	Role	

i Showing a filtered set of results. Total number of role assignments: 15

1 items (1 Users)

<input type="checkbox"/>	NAME	TYPE	ROLE	SCOPE
	John Doe John.Doe@example.com	User	Reader ?	This resource

You can also assign a role to the user from the **Roles** page.

- Click **Roles** from the **Access control (IAM)** page to open the **Roles** page. From here, you can view the name of the role, the number of users and groups assigned to that role.

Roles

TestAzureAuto

NAME	USERS	GROUPS
Owner ?	0	1
Contributor ?	12	0
Reader ?	0	0
Automation Operator ?	0	0
Azure Service Deploy Release Management Contributor ?	0	0
Log Analytics Contributor ?	0	0
Log Analytics Reader ?	0	0
masterreader ?	0	0
Monitoring Contributor ?	0	0
Monitoring Reader ?	0	0
Resource Policy Contributor (Preview) ?	0	0
User Access Administrator ?	0	0

NOTE

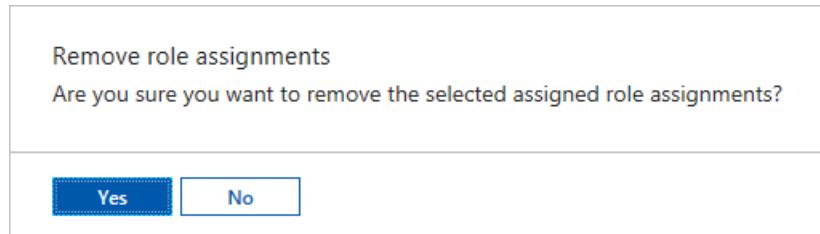
Role-based access control can only be set at the Automation account scope and not at any resource below the Automation account.

Remove a user

You can remove the access permission for a user who is not managing the Automation account, or who no longer works for the organization. Following are the steps to remove a user:

- From the **Access control (IAM)** page, select the user wish to remove and click **Remove**.

2. Click the **Remove** button in the assignment details pane.
3. Click **Yes** to confirm removal.



Configure RBAC using PowerShell

Role-based access can also be configured to an Automation account using the following [Azure PowerShell cmdlets](#):

[Get-AzureRmRoleDefinition](#) lists all RBAC roles that are available in Azure Active Directory. You can use this command along with the **Name** property to list all the actions that can be performed by a specific role.

```
Get-AzureRmRoleDefinition -Name 'Automation Operator'
```

The following is the example output:

```
Name      : Automation Operator
Id       : d3881f73-407a-4167-8283-e981cbba0404
IsCustom : False
Description : Automation Operators are able to start, stop, suspend, and resume jobs
Actions   : {Microsoft.Authorization/*/read, Microsoft.Automation/automationAccounts/jobs/read,
Microsoft.Automation/automationAccounts/jobs/resume/action,
Microsoft.Automation/automationAccounts/jobs/stop/action...}
NotActions : {}
AssignableScopes : {}
```

[Get-AzureRmRoleAssignment](#) lists Azure AD RBAC role assignments at the specified scope. Without any parameters, this command returns all the role assignments made under the subscription. Use the **ExpandPrincipalGroups** parameter to list access assignments for the specified user as well as the groups the user is a member of. **Example:** Use the following command to list all the users and their roles within an automation account.

```
Get-AzureRMRoleAssignment -scope '/subscriptions/<SubscriptionID>/resourcegroups/<Resource Group Name>/Providers/Microsoft.Automation/automationAccounts/<Automation account name>'
```

The following is the example output:

```
RoleAssignmentId  : /subscriptions/00000000-0000-0000-0000-
0000000000/resourceGroups/myResourceGroup/providers/Microsoft.Automation/automationAccounts/myAutomationAcco
unt/provid
ers/Microsoft.Authorization/roleAssignments/cc594d39-ac10-46c4-9505-f182a355c41f
Scope           : /subscriptions/00000000-0000-0000-0000-
0000000000/resourceGroups/myResourceGroup/providers/Microsoft.Automation/automationAccounts/myAutomationAcco
unt
DisplayName     : admin@contoso.com
SignInName      : admin@contoso.com
RoleDefinitionName : Automation Operator
RoleDefinitionId : d3881f73-407a-4167-8283-e981cbba0404
ObjectId        : 15f26a47-812d-489a-8197-3d4853558347
ObjectType      : User
```

[New-AzureRmRoleAssignment](#) to assign access to users, groups, and applications to a particular scope. **Example:** Use the following command to assign the "Automation Operator" role for a user in the Automation account scope.

```
New-AzureRmRoleAssignment -SignInName <sign-in Id of a user you wish to grant access> -RoleDefinitionName 'Automation operator' -Scope '/subscriptions/<SubscriptionID>/resourcegroups/<Resource Group Name>/Providers/Microsoft.Automation/automationAccounts/<Automation account name>'
```

The following is the example output:

```
RoleAssignmentId    : /subscriptions/00000000-0000-0000-0000-000000000000/resourcegroups/myResourceGroup/Providers/Microsoft.Automation/automationAccounts/myAutomationAccount/provid
                     ers/Microsoft.Authorization/roleAssignments/25377770-561e-4496-8b4f-7cba1d6fa346
Scope              : /subscriptions/00000000-0000-0000-0000-000000000000/resourcegroups/myResourceGroup/Providers/Microsoft.Automation/automationAccounts/myAutomationAcco
unt
DisplayName        : admin@contoso.com
SignInName         : admin@contoso.com
RoleDefinitionName : Automation Operator
RoleDefinitionId   : d3881f73-407a-4167-8283-e981cbba0404
ObjectId           : f5ecbe87-1181-43d2-88d5-a8f5e9d8014e
ObjectType         : User
```

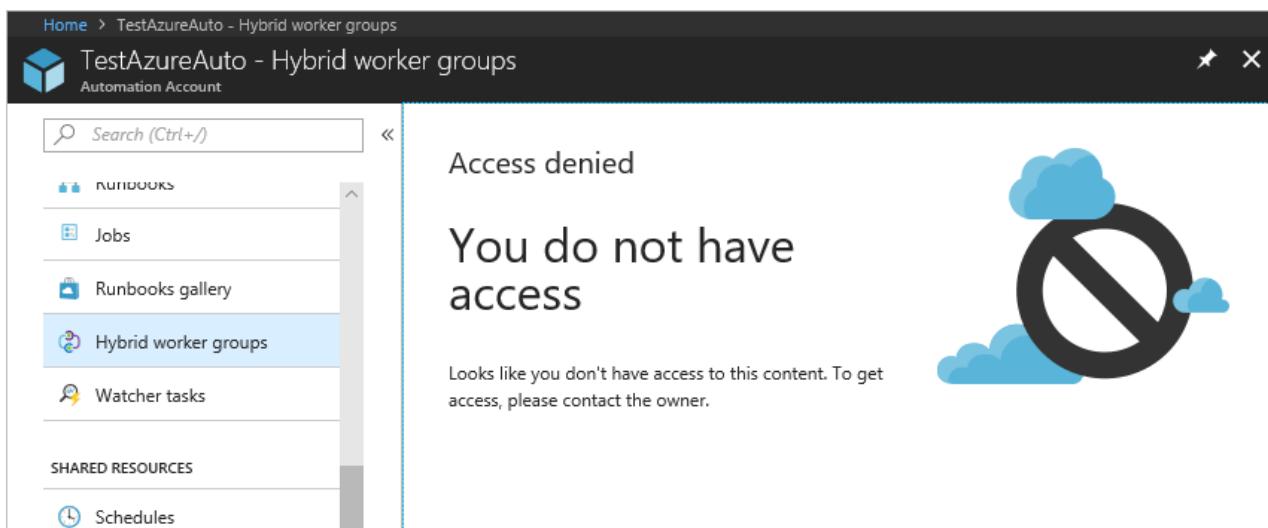
Use [Remove-AzureRmRoleAssignment](#) to remove access of a specified user, group, or application from a particular scope. **Example:** Use the following command to remove the user from the "Automation Operator" role in the Automation account scope.

```
Remove-AzureRmRoleAssignment -SignInName <sign-in Id of a user you wish to remove> -RoleDefinitionName 'Automation Operator' -Scope '/subscriptions/<SubscriptionID>/resourcegroups/<Resource Group Name>/Providers/Microsoft.Automation/automationAccounts/<Automation account name>'
```

In the preceding examples, replace **sign in Id**, **subscription Id**, **resource group name**, and **Automation account name** with your account details. Choose **yes** when prompted to confirm before continuing to remove user role assignment.

User experience for Automation operator role - Automation Account

When a user, who is assigned to the Automation Operator role on the Automation Account scope views the Automation account they are assigned to, they can only view the list of runbooks, runbook jobs, and schedules created in the Automation account but can't view their definition. They can start, stop, suspend, resume, or schedule the runbook job. The user does not have access to other Automation resources such as configurations, hybrid worker groups, or DSC nodes.



Configure RBAC for Runbooks

Azure Automation allows for you to assign RBAC to specific runbooks. To do this run the following script to add a user to a specific runbook. The following script can be ran by an Automation Account Admin or Tenant Admin.

```
$rgName = "<Resource Group Name>" # Resource Group name for the Automation Account
$automationAccountName = "<Automation Account Name>" # Name of the Automation Account
$rbName = "<Name of Runbook>" # Name of the runbook
$userID = "<User ObjectId>" # Azure Active Directory (AAD) user's ObjectId from the directory

# Gets the Automation Account resource
$aa = Get-AzureRmResource -ResourceGroupName $rgName -ResourceType "Microsoft.Automation/automationAccounts" -ResourceName $automationAccountName

# Get the Runbook resource
$rb = Get-AzureRmResource -ResourceGroupName $rgName -ResourceType "Microsoft.Automation/automationAccounts/runbooks" -ResourceName "$automationAccountName/$rbName"

# The Automation Job Operator role only needs to be ran once per user.
New-AzureRmRoleAssignment -ObjectId $userID -RoleDefinitionName "Automation Job Operator" -Scope
$aa.ResourceId

# Adds the user to the Automation Runbook Operator role to the Runbook scope
New-AzureRmRoleAssignment -ObjectId $userID -RoleDefinitionName "Automation Runbook Operator" -Scope
$rb.ResourceId
```

Once ran, have the user log in to the Azure portal and view **All Resources**. In the list they see the Runbook they were added as a **Automation Runbook Operator** for.

NAME	TYPE	RESOURCE GROUP	LOCATION	SUBSCRIPTION
Hello-World (TestAzureAutomationAc...)	Runbook	Contoso Resource...	East US	Contoso Subscription

User experience for Automation operator role - Runbook

When a user, who is assigned to the Automation Operator role on the Runbook scope views a Runbook they are assigned to, they can only start the runbook and view the runbook jobs.

The screenshot shows the Azure portal interface for a runbook named "Hello-World". The left sidebar contains a navigation menu with the following items:

- Overview (selected)
- Activity log
- Tags
- Diagnose and solve problems

Below this is a "RESOURCES" section with:

- Jobs
- Schedules
- Webhooks

At the bottom is a "RUNBOOK SETTINGS" section with:

- Properties

The main content area has a header with buttons: Start, View, Edit, Schedule, Webhook, Delete, and More. Below the header, it says "No access". A "Details" section for "Jobs" is shown, featuring a blue square icon with a checkmark and a question mark.

Next steps

- For information on different ways to configure RBAC for Azure Automation, refer to [manage RBAC with Azure PowerShell](#).
- For details on different ways to start a runbook, see [Starting a runbook](#)
- For information about different runbook types, refer to [Azure Automation runbook types](#)

Manage Azure Automation account

5/21/2018 • 2 minutes to read • [Edit Online](#)

At some point before your Automation account expires, you need to renew the certificate. If you believe that the Run As account has been compromised, you can delete and re-create it. This section discusses how to perform these operations.

Self-signed certificate renewal

The self-signed certificate that you created for the Run As account expires one year from the date of creation. You can renew it at any time before it expires. When you renew it, the current valid certificate is retained to ensure that any runbooks that are queued up or actively running, and that authenticate with the Run As account, are not negatively affected. The certificate remains valid until its expiration date.

NOTE

If you have configured your Automation Run As account to use a certificate issued by your enterprise certificate authority and you use this option, the enterprise certificate is replaced by a self-signed certificate.

To renew the certificate, do the following:

1. In the Azure portal, open the Automation account.
2. On the **Automation Account**
3. , in the **Account properties** pane, under **Account Settings**, select **Run As Accounts**.

TestAutomationAcct
Automation Account

Search (Ctrl+ /)

Connections

Certificates

Variables

RELATED RESOURCES

Solutions

Workspace

Unlink Workspace

ACCOUNT SETTINGS

Properties

Keys

Pricing tier and usage

Source Control

Run As Accounts

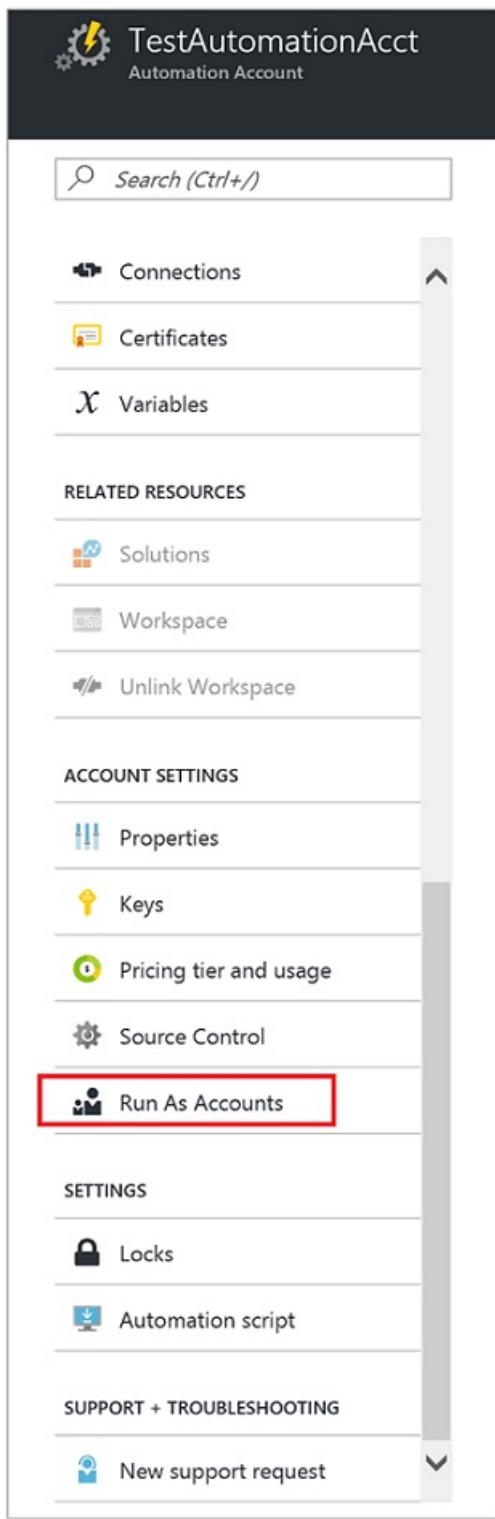
SETTINGS

Locks

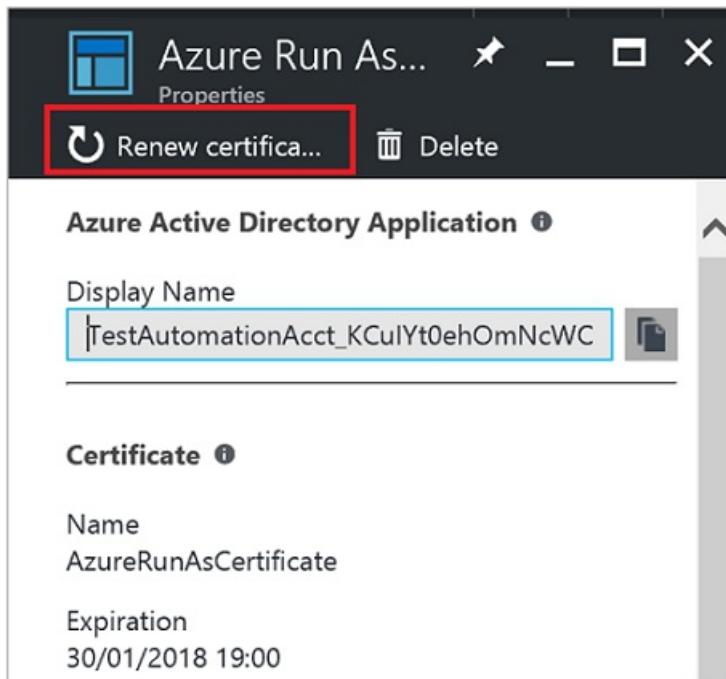
Automation script

SUPPORT + TROUBLESHOOTING

New support request



4. On the **Run As Accounts** properties page, select either the Run As account or the Classic Run As account that you want to renew the certificate for.
5. On the **Properties** pane for the selected account, click **Renew certificate**.

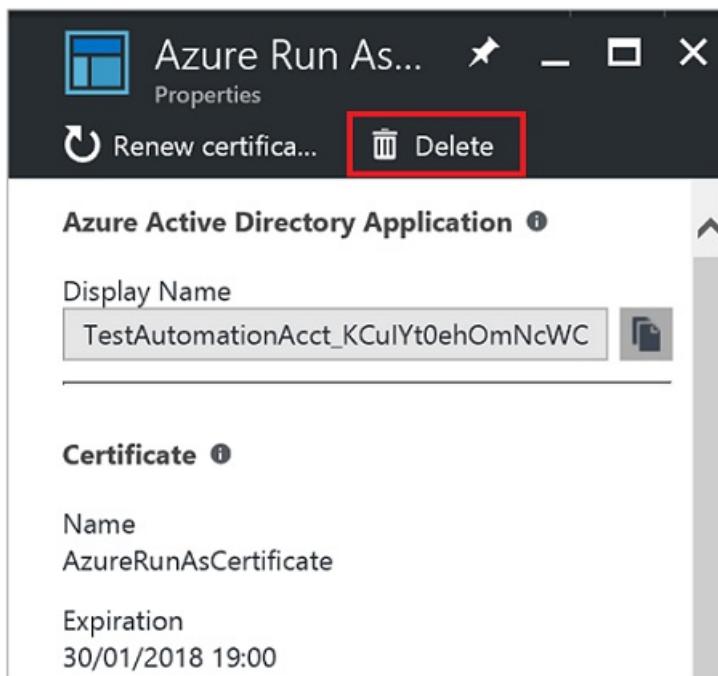


6. While the certificate is being renewed, you can track the progress under **Notifications** from the menu.

Delete a Run As or Classic Run As account

This section describes how to delete and re-create a Run As or Classic Run As account. When you perform this action, the Automation account is retained. After you delete a Run As or Classic Run As account, you can re-create it in the Azure portal.

1. In the Azure portal, open the Automation account.
2. On the **Automation account** page, select **Run As Accounts**.
3. On the **Run As Accounts** properties page, select either the Run As account or Classic Run As account that you want to delete. Then, on the **Properties** pane for the selected account, click **Delete**.



4. While the account is being deleted, you can track the progress under **Notifications** from the menu.
5. After the account has been deleted, you can re-create it on the **Run As Accounts** properties page by selecting the create option **Azure Run As Account**.

The screenshot shows the Azure portal interface for an Automation Account named 'TestAutomationAcct'. The left sidebar contains a search bar and links for 'Connections', 'Certificates', 'Variables', 'Solutions', 'Workspace', 'Unlink Workspace', 'Properties', 'Keys', 'Pricing tier and usage', 'Source Control', and 'Run As Accounts'. The 'Run As Accounts' link is highlighted with a blue background. The main pane displays two items: 'Azure Run As Account' (Create) and 'Azure Classic Run As Account' (Expires 30/01/2018 19:00).

Misconfiguration

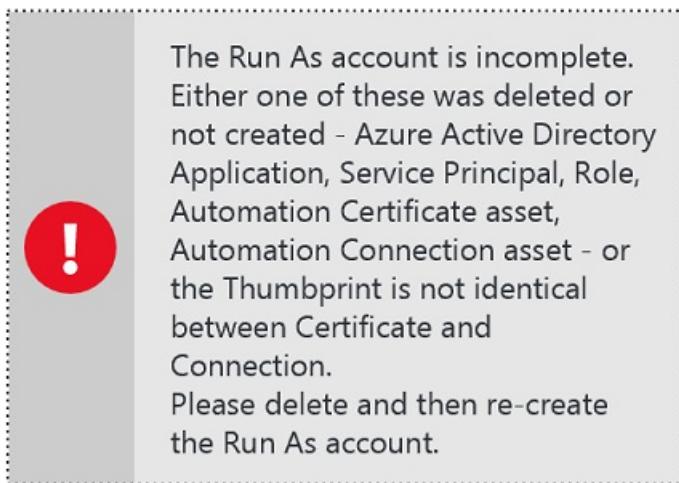
Some configuration items necessary for the Run As or Classic Run As account to function properly might have been deleted or created improperly during initial setup. The items include:

- Certificate asset
- Connection asset
- Run As account has been removed from the contributor role
- Service principal or application in Azure AD

In the preceding and other instances of misconfiguration, the Automation account detects the changes and displays a status of *Incomplete* on the **Run As Accounts** properties pane for the account.

The screenshot shows the Azure portal interface for an Automation Account named 'TestAutomationAcct'. The left sidebar contains navigation links: 'Connections', 'Certificates', 'Variables', 'Solutions', 'Workspace', 'Unlink Workspace', 'Properties', 'Keys', 'Pricing tier and usage', 'Source Control', and 'Run As Accounts'. The 'Run As Accounts' link is highlighted with a blue background. The main pane displays two entries under 'Run As Accounts': 'Azure Run As Account' (status: 'Incomplete') and 'Azure Classic Run As Account' (status: 'Expires 31/01/2018 19:00').

When you select the Run As account, the account **Properties** pane displays the following error message:



You can quickly resolve these Run As account issues by deleting and re-creating the account.

Next steps

- For more information about Service Principals, refer to [Application Objects and Service Principal Objects](#).
- For more information about Role-based Access Control in Azure Automation, refer to [Role-based access control in Azure Automation](#).
- For more information about certificates and Azure services, refer to [Certificates overview for Azure Cloud Services](#).

Azure Automation runbook types

6/29/2018 • 4 minutes to read • [Edit Online](#)

Azure Automation supports several types of runbooks that are briefly described in the following table. The sections below provide further information about each type including considerations on when to use each.

Type	Description
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.
PowerShell	Text runbook based on Windows PowerShell script.
PowerShell Workflow	Text runbook based on Windows PowerShell Workflow.
Python	Text runbook based on Python.

Graphical runbooks

[Graphical](#) and Graphical PowerShell Workflow runbooks are created and edited with the graphical editor in the Azure portal. You can export them to a file and then import them into another automation account, but you cannot create or edit them with another tool. Graphical runbooks generate PowerShell code, but you can't directly view or modify the code. Graphical runbooks cannot be converted to one of the [text formats](#), nor can a text runbook be converted to graphical format. Graphical runbooks can be converted to Graphical PowerShell Workflow runbooks during import and vice-versa.

Advantages

- Visual insert-link-configure authoring model
- Focus on how data flows through the process
- Visually represent management processes
- Include other runbooks as child runbooks to create high-level workflows
- Encourages modular programming

Limitations

- Can't edit runbook outside of Azure portal.
- May require a Code activity containing PowerShell code to perform complex logic.
- Can't view or directly edit the PowerShell code that is created by the graphical workflow. Note that you can view the code you create in any Code activities.

PowerShell runbooks

PowerShell runbooks are based on Windows PowerShell. You directly edit the code of the runbook using the text editor in the Azure portal. You can also use any offline text editor and [import the runbook](#) into Azure Automation.

Advantages

- Implement all complex logic with PowerShell code without the additional complexities of PowerShell Workflow.
- Runbook starts faster than 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 and Graphical runbooks can only be included as child runbooks by using the Start-AzureAutomationRunbook cmdlet which creates a new job.

Known Issues

Following are current known issues with PowerShell runbooks.

- PowerShell runbooks cannot retrieve an unencrypted [variable asset](#) with a null value.
- PowerShell runbooks cannot retrieve a [variable asset](#) with ~ in the name.
- Get-Process in a loop in a PowerShell runbook may crash after about 80 iterations.
- A PowerShell runbook may fail if it attempts to write a very large amount of data to the output stream at once. You can typically work around this issue by outputting just the information you need when working with large objects. For example, instead of outputting something like `Get-Process`, you can output just the required fields with `Get-Process | Select ProcessName, CPU`.

PowerShell Workflow runbooks

PowerShell Workflow runbooks are text runbooks based on [Windows PowerShell Workflow](#). You directly edit the code of the runbook using the text editor in the Azure portal. You can also use any offline text editor and [import the runbook](#) into Azure Automation.

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.
- Can 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.
- PowerShell runbooks can only be included as child runbooks by using the Start-AzureAutomationRunbook cmdlet which creates a new job.

Python runbooks

Python runbooks compile under Python 2. You can directly edit the code of the runbook using the text editor in the Azure portal, or you can use any offline text editor and [import the runbook](#) into Azure Automation.

Advantages

- Utilize the robust standard library of Python.

Limitations

- Must be familiar with Python scripting.

- Only Python 2 is supported at the moment, meaning Python 3 specific functions will fail.

Known Issues

Following are current known issues with Python runbooks.

- In order to utilize third-party libraries, the runbook must be run on a [Windows Hybrid Runbook Worker](#) or [Linux Hybrid Runbook Worker](#) with the libraries already installed on the machine before the runbook is started.

Considerations

You should take into account the following additional considerations when determining which type to use for a particular runbook.

- You can't convert runbooks from graphical to textual type or vice-versa.
- There are limitations using runbooks of different types as a child runbook. See [Child runbooks in Azure Automation](#) for more information.

Next steps

- To learn more about Graphical runbook authoring, see [Graphical authoring in Azure Automation](#)
- To understand the differences between PowerShell and PowerShell workflows for runbooks, see [Learning Windows PowerShell Workflow](#)
- For more information on how to create or import a Runbook, see [Creating or Importing a Runbook](#)

Creating or importing a runbook in Azure Automation

5/23/2018 • 4 minutes to read • [Edit Online](#)

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](#). This article provides information on creating and importing runbooks from a file. You can get all of the details on accessing community runbooks and modules in [Runbook and module galleries for Azure Automation](#).

Creating a new runbook

You can create a new runbook in Azure Automation using one of the Azure portals or Windows PowerShell. Once the runbook has been created, you can edit it using information in [Learning PowerShell Workflow](#) and [Graphical authoring in Azure Automation](#).

To create a new Azure Automation runbook with the Azure portal

1. In the Azure portal, open your Automation account.
2. From the Hub, select **Runbooks** to open the list of runbooks.
3. Click on the **Add a runbook** button and then **Create a new runbook**.
4. Type a **Name** for the runbook and select its **Type**. The runbook name must start with a letter and can have letters, numbers, underscores, and dashes.
5. Click **Create** to create the runbook and open the editor.

To create a new Azure Automation runbook with Windows PowerShell

You can use the `New-AzureRmAutomationRunbook` cmdlet to create an empty [PowerShell Workflow](#) runbook. You can either specify the **Name** parameter to create an empty runbook that you can later edit, or you can specify the **Path** parameter to import a runbook file. The **Type** parameter should also be included to specify one of the four runbook types.

The following sample commands show how to create a new empty runbook.

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

Importing a runbook from a file into Azure Automation

You can create a new runbook in Azure Automation by importing a PowerShell script or PowerShell Workflow (.ps1 extension), an exported graphical runbook (.graphrunbook), or a Python 2 script (.py extension). You must specify the [type of runbook](#) that is created during import, taking into account the following considerations.

- A .graphrunbook file may only be imported into a new [graphical runbook](#), and graphical runbooks can only be created from a .graphrunbook file.
- A .ps1 file containing a PowerShell Workflow can only be imported into a [PowerShell Workflow runbook](#). If the file contains multiple PowerShell Workflows, then the import will fail. You must save each workflow to its own file and import each separately.
- A .ps1 file that does not contain a workflow can be imported into either a [PowerShell runbook](#) or a [PowerShell Workflow runbook](#). If it is imported into a PowerShell Workflow runbook, then it is converted to a workflow, and comments are included in the runbook specifying the changes that were made.

To import a runbook from a file with the Azure portal

You can use the following procedure to import a script file into Azure Automation.

NOTE

Note that you can only import a .ps1 file into a PowerShell Workflow runbook using the portal.

1. In the Azure portal, open your Automation account.
2. From the Hub, select **Runbooks** to open the list of runbooks.
3. Click on the **Add a runbook** button and then **Import**.
4. Click **Runbook file** to select the file to import
5. If the **Name** field is enabled, then you have the option to change it. The runbook name must start with a letter and can have letters, numbers, underscores, and dashes.
6. The **runbook type** is automatically selected, but you can change the type after taking the applicable restrictions into account.
7. The new runbook appears in the list of runbooks for the Automation Account.
8. You must [publish the runbook](#) before you can run it.

NOTE

After you import a graphical runbook or a graphical PowerShell workflow runbook, you have the option to convert to the other type if wanted. You can't convert to a textual runbook.

To import a runbook from a script file with Windows PowerShell

You can use the [Import-AzureRMAutomationRunbook](#) cmdlet to import a script file as a draft PowerShell Workflow runbook. If the runbook already exists, the import fails unless you use the **-Force** parameter.

The following sample commands show how to import a script file into a runbook.

```
$automationAccountName = "AutomationAccount"
$runbookName = "Sample_TestRunbook"
$scriptPath = "C:\Runbooks\Sample_TestRunbook.ps1"
$RGName = "ResourceGroup"

Import-AzureRMAutomationRunbook -Name $runbookName -Path $scriptPath ` 
-ResourceGroupName $RGName -AutomationAccountName $automationAccountName ` 
-Type PowerShellWorkflow
```

Publishing a runbook

When you create or import a new runbook, you must publish it before you can run it. Each runbook in Automation has a Draft and a Published version. Only the Published version is available to be run, and only the Draft version can be edited. The Published version is unaffected by any changes to the Draft version. When the Draft version should be made available, then you publish it which overwrites the Published version with the Draft version.

To publish a runbook using the Azure portal

1. Open the runbook in the Azure portal.
2. Click the **Edit** button.
3. Click the **Publish** button and then **Yes** to the verification message.

To publish a runbook using Windows PowerShell

You can use the [Publish-AzureRmAutomationRunbook](#) cmdlet to publish a runbook with Windows PowerShell. The following sample commands show how to publish a sample runbook.

```
$automationAccountName = "AutomationAccount"
$runbookName = "Sample_TestRunbook"
$RGName = "ResourceGroup"

Publish-AzureRmAutomationRunbook -AutomationAccountName $automationAccountName ` 
-Name $runbookName -ResourceGroupName $RGName
```

Next Steps

- To learn about how you can benefit from the Runbook and PowerShell Module Gallery, see [Runbook and module galleries for Azure Automation](#)
- To learn more about editing PowerShell and PowerShell Workflow runbooks with a textual editor, see [Editing textual runbooks in Azure Automation](#)
- To learn more about Graphical runbook authoring, see [Graphical authoring in Azure Automation](#)

My first graphical runbook

5/21/2018 • 13 minutes to read • [Edit Online](#)

This tutorial walks you through the creation of a [graphical runbook](#) in Azure Automation. You start with a simple runbook that tests and publishes while learning how to track the status of the runbook job. Then you modify the runbook to actually manage Azure resources, in this case starting an Azure virtual machine. Then you complete the tutorial by making the runbook more robust by adding runbook parameters and conditional links.

Prerequisites

To complete this tutorial, you need the following:

- Azure subscription. If you don't have one yet, you can [activate your MSDN subscriber benefits](#) or sign up for a [free account](#).
- [Automation account](#) to hold the runbook and authenticate to Azure resources. This account must have permission to start and stop the virtual machine.
- An Azure virtual machine. You stop and start this machine so it should not be a production VM.

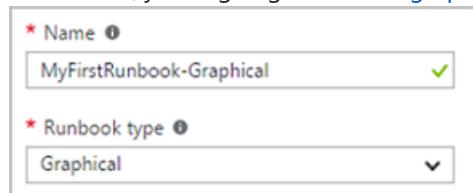
Create runbook

Start by creating a simple runbook that outputs the text *Hello World*.

1. In the Azure portal, open your Automation account.

The Automation account page gives you a quick view of the resources in this account. You should already have some Assets. Most of those assets are the modules that are automatically included in a new Automation account. You should also have the Credential asset that's mentioned in the [prerequisites](#).

2. Select **Runbooks** under **PROCESS MANAGEMENT** to open the list of runbooks.
3. Create a new runbook by selecting **+ Add a runbook**, then click **Create a new runbook**.
4. Give the runbook the name *MyFirstRunbook-Graphical*.
5. In this case, you're going to create a [graphical runbook](#) so select **Graphical** for **Runbook type**.

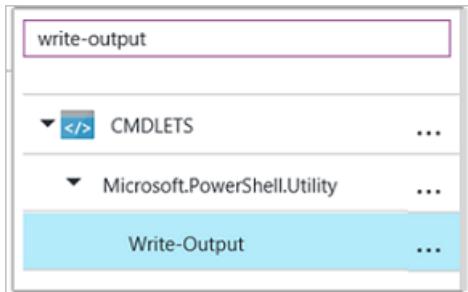


6. Click **Create** to create the runbook and open the graphical editor.

Add activities

The Library control on the left side of the editor allows you to select activities to add to your runbook. You're going to add a **Write-Output** cmdlet to output text from the runbook.

1. In the Library control, click in the search textbox and type **Write-Output**. The search results are shown in the following image:



2. Scroll down to the bottom of the list. You can either right-click **Write-Output** and select **Add to canvas**, or click the ellipses next to the cmdlet and then select **Add to canvas**.
3. Click the **Write-Output** activity on the canvas. This action opens the Configuration control page, which allows you to configure the activity.
4. The **Label** defaults to the name of the cmdlet, but you can change it to something more friendly. Change it to *Hello World*.
5. Click **Parameters** to provide values for the cmdlet's parameters.

Some cmdlets have multiple parameter sets, and you need to select which one to use. In this case, **Write-Output** has only one parameter set, so you don't need to select one.

6. Select the **InputObject** parameter. This is the parameter where you specify the text to send to the output stream.
7. In the **Data source** dropdown, select **PowerShell expression**. The **Data source** dropdown provides different sources that you use to populate a parameter value.

You can use output from such sources such as another activity, an Automation asset, or a PowerShell expression. In this case, the output is just *Hello World*. You can use a PowerShell expression and specify a string.

8. In the **Expression** box, type "*Hello World*" and then click **OK** twice to return to the canvas.
9. Save the runbook by clicking **Save**.

Test the runbook

Before you publish the runbook to make it available in production, you want to test it to make sure that it works properly. When you test a runbook, you run its **Draft** version and view its output interactively.

1. Select **Test pane** to open the Test page.
2. Click **Start** to start the test. This should be the only enabled option.
3. A **runbook job** is created and its status displayed in the pane.

The job status starts as *Queued* indicating that it is waiting for a runbook worker in the cloud to become available. It then moves to *Starting* when a worker claims the job, and then *Running* when the runbook actually starts running.

4. When the runbook job completes, its output is displayed. In this case, you see *Hello World*.

The screenshot shows the Azure Runbook Test page for 'MyFirstRunbook-Graphical1'. The left sidebar contains sections for 'Parameters' (No input parameters), 'Run Settings' (Run on: Azure, Hybrid Worker), and 'Activity-level tracing' (Trace level: None, Basic, Detailed). The main area displays the job status as 'Completed' and the output stream containing the text 'Hello World'.

5. Close the Test page to return to the canvas.

Publish and start the runbook

The runbook that you created is still in Draft mode. It needs to be published before you can run it in production. When you publish a runbook, you overwrite the existing Published version with the Draft version. In this case, you don't have a Published version yet because you just created the runbook.

1. Select **Publish** to publish the runbook and then **Yes** when prompted.
2. If you scroll left to view the runbook in the **Runbooks** page, it shows an **Authoring Status** of **Published**.
3. Scroll back to the right to view the page for **MyFirstRunbook-Graphical**.

The options across the top allow us to start the runbook, schedule it to start at some time in the future, or create a [webhook](#) so it can be started through an HTTP call.

4. Select **Start** and then **Yes** when prompted to start the runbook.
5. A job page is opened for the runbook job that was created. Verify that the **Job status** shows **Completed**.
6. Once the runbook status shows *Completed*, click **Output**. The **Output** page is opened, and you can see the *Hello World* in the pane.
7. Close the Output page.
8. Click **All Logs** to open the Streams page for the runbook job. You should only see *Hello World* in the output stream, but this can show other streams for a runbook job such as Verbose and Error if the runbook writes to them.
9. Close the All Logs page and the Job page to return to the MyFirstRunbook-Graphical page.
10. To view all the jobs for the runbook close the **Job** page and select **Jobs** under **RESOURCES**. This lists all the jobs created by this runbook. You should only see one job listed since you only ran the job once.
11. You can click this job to open the same Job pane that you viewed when you started the runbook. This allows you to go back in time and view the details of any job that was created for a particular runbook.

Create variable assets

You've tested and published your runbook, but so far it doesn't do anything useful. You want to have it manage Azure resources. Before you configure the runbook to authenticate, you create a variable to hold the subscription ID and reference it after you set up the activity to authenticate in step 6 below. Including a reference to the subscription context allows you to easily work between multiple subscriptions. Before proceeding, copy your

subscription ID from the Subscriptions option off the Navigation pane.

1. In the Automation Accounts page, select **Variables** under **SHARED RESOURCES**.
2. Select **Add a variable**.
3. In the New variable page, in the **Name** box, enter **AzureSubscriptionId** and in the **Value** box enter your Subscription ID. Keep *string* for the **Type** and the default value for **Encryption**.
4. Click **Create** to create the variable.

Add authentication

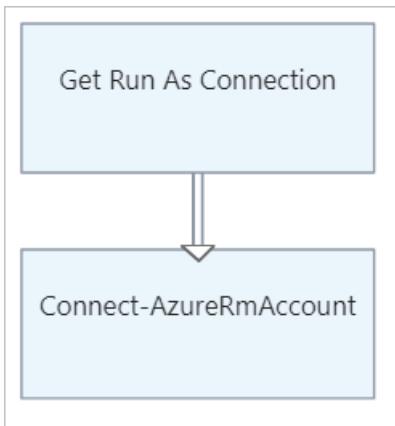
Now that you have a variable to hold the subscription ID, you can configure the runbook to authenticate with the Run As credentials that are referred to in the [prerequisites](#). You do that by adding the Azure Run As connection **Asset** and **Connect-AzureRmAccount** cmdlet to the canvas.

1. Navigate back to your runbook and select **Edit** on the MyFirstRunbook-Graphical page.
2. You don't need the **Write Hello World to output** anymore, so click the ellipses (...) and select **Delete**.
3. In the Library control, expand **ASSETS, Connections**, and add **AzureRunAsConnection** to the canvas by selecting **Add to canvas**.
4. In the Library control, type **Connect-AzureRmAccount** in the search textbox.

IMPORTANT

Add-AzureRmAccount is now an alias for **Connect-AzureRMAccount**. When searching your library items, if you do not see **Connect-AzureRMAccount**, you can use **Add-AzureRmAccount**, or you can update your modules in your Automation Account.

5. Add **Connect-AzureRmAccount** to the canvas.
6. Hover over **Get Run As Connection** until a circle appears on the bottom of the shape. Click the circle and drag the arrow to **Connect-AzureRmAccount**. The arrow that you created is a *link*. The runbook starts with **Get Run As Connection** and then run **Connect-AzureRmAccount**.



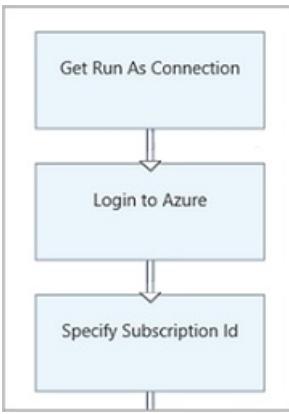
7. On the canvas, select **Connect-AzureRmAccount** and in the Configuration control pane type **Login to Azure** in the **Label** textbox.
8. Click **Parameters** and the Activity Parameter Configuration page appears.
9. **Connect-AzureRmAccount** has multiple parameter sets, so you need to select one before you can provide parameter values. Click **Parameter Set** and then select the **ServicePrincipalCertificate** parameter set.
10. Once you select the parameter set, the parameters are displayed in the Activity Parameter Configuration page. Click **APPLICATIONID**.

The screenshot shows the 'Parameter sets' and 'Parameters' sections of the Azure Runbook configuration. The 'Parameters' section is expanded, showing the following configuration:

- APPLICATIONID**: Set to 'Get Run As Connection (Activity output...)'.
- CERTIFICATETHUMBPRINT**: Set to 'Get Run As Connection (Activity output...)'.
- ENVIRONMENT**: Set to 'Not configured'.
- ENVIRONMENTNAME**: Set to 'Not configured'.
- SERVICEPRINCIPAL**: Set to 'true (Constant value)'.
- TENANTID**: Set to 'Get Run As Connection (Activity output...)'.

11. In the Parameter Value page, select **Activity output** for the **Data source** and select **Get Run As Connection** from the list, in the **Field path** textbox type **ApplicationId**, and then click **OK**. You are specifying the name of the property for the Field path because the activity outputs an object with multiple properties.
12. Click **CERTIFICATETHUMBPRINT**, and in the Parameter Value page, select **Activity output** for the **Data source**. Select **Get Run As Connection** from the list, in the **Field path** textbox type **CertificateThumbprint**, and then click **OK**.
13. Click **SERVICEPRINCIPAL**, and in the Parameter Value page, select **ConstantValue** for the **Data source**, click the option **True**, and then click **OK**.
14. Click **TENANTID**, and in the Parameter Value page, select **Activity output** for the **Data source**. Select **Get Run As Connection** from the list, in the **Field path** textbox type **TenantId**, and then click **OK** twice.
15. In the Library control, type **Set-AzureRmContext** in the search textbox.
16. Add **Set-AzureRmContext** to the canvas.
17. On the canvas, select **Set-AzureRmContext** and in the Configuration control pane type **Specify Subscription Id** in the **Label** textbox.
18. Click **Parameters** and the Activity Parameter Configuration page appears.
19. **Set-AzureRmContext** has multiple parameter sets, so you need to select one before you can provide parameter values. Click **Parameter Set** and then select the **SubscriptionId** parameter set.
20. Once you select the parameter set, the parameters are displayed in the Activity Parameter Configuration page. Click **SubscriptionID**
21. In the Parameter Value page, select **Variable Asset** for the **Data source** and select **AzureSubscriptionId** from the list and then click **OK** twice.
22. Hover over **Login to Azure** until a circle appears on the bottom of the shape. Click the circle and drag the arrow to **Specify Subscription Id**.

Your runbook should look like the following at this point:

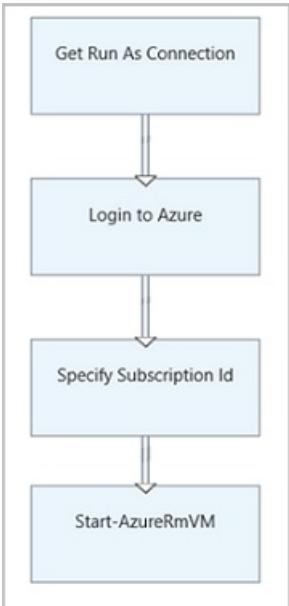


Add activity to start a VM

Here you add a **Start-AzureRmVM** activity to start a virtual machine. You can pick any virtual machine in your Azure subscription, and for now you hardcode that name into the cmdlet.

1. In the Library control, type **Start-AzureRm** in the search textbox.
2. Add **Start-AzureRmVM** to the canvas and then click and drag it underneath **Specify Subscription Id**.
3. Hover over **Specify Subscription Id** until a circle appears on the bottom of the shape. Click the circle and drag the arrow to **Start-AzureRmVM**.
4. Select **Start-AzureRmVM**. Click **Parameters** and then **Parameter Set** to view the sets for **Start-AzureRmVM**. Select the **ResourceGroupNameParameterSetName** parameter set. **ResourceGroupName** and **Name** have exclamation points next them. This indicates that they are required parameters. Also note both expect string values.
5. Select **Name**. Select **PowerShell expression** for the **Data source** and type in the name of the virtual machine surrounded with double quotes that you start with this runbook. Click **OK**.
6. Select **ResourceGroupName**. Use **PowerShell expression** for the **Data source** and type in the name of the resource group surrounded with double quotes. Click **OK**.
7. Click Test pane so that you can test the runbook.
8. Click **Start** to start the test. Once it completes, check that the virtual machine was started.

Your runbook should look like the following at this point:

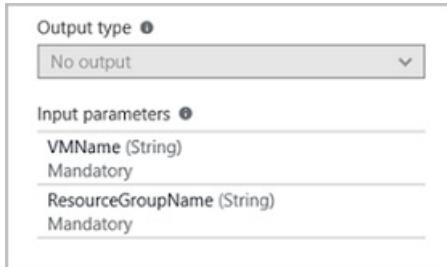


Add additional input parameters

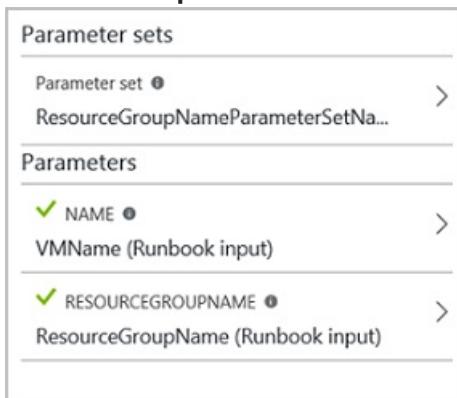
Our runbook currently starts the virtual machine in the resource group that you specified in the **Start-**

AzureRmVM cmdlet. The runbook would be more useful if we could specify both when the runbook is started. You now add input parameters to the runbook to provide that functionality.

1. Open the graphical editor by clicking **Edit** on the **MyFirstRunbook-Graphical** pane.
2. Select **Input and output** and then **Add input** to open the Runbook Input Parameter pane.
3. Specify **VMName** for the **Name**. Keep *string* for the **Type**, but change **Mandatory** to Yes. Click **OK**.
4. Create a second mandatory input parameter called **ResourceGroupName** and then click **OK** to close the **Input and Output** pane.



5. Select the **Start-AzureRmVM** activity and then click **Parameters**.
6. Change the **Data source** for **Name** to **Runbook input** and then select **VMName**.
7. Change the **Data source** for **ResourceGroupName** to **Runbook input** and then select **ResourceGroupName**.



8. Save the runbook and open the Test pane. You can now provide values for the two input variables that you use in the test.
9. Close the Test pane.
10. Click **Publish** to publish the new version of the runbook.
11. Stop the virtual machine that you started in the previous step.
12. Click **Start** to start the runbook. Type in the **VMName** and **ResourceGroupName** for the virtual machine that you're going to start.
13. When the runbook completes, check that the virtual machine was started.

Create a conditional link

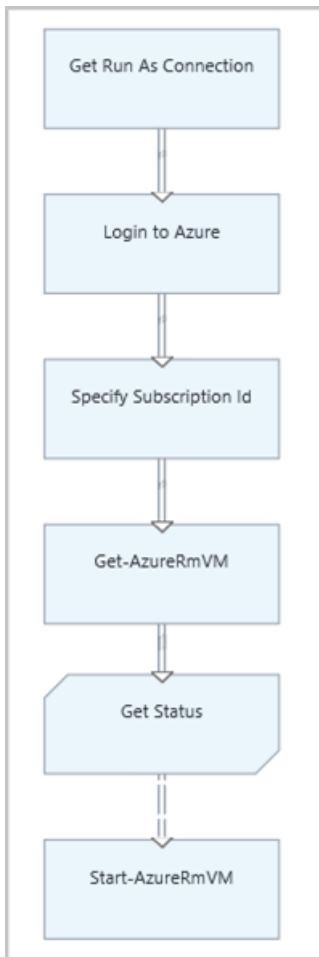
You now modify the runbook so that it only attempts to start the virtual machine if it is not already started. You do this by adding a **Get-AzureRmVM** cmdlet to the runbook that gets the instance level status of the virtual machine. Then you add a PowerShell Workflow code module called **Get Status** with a snippet of PowerShell code to determine if the virtual machine state is running or stopped. A conditional link from the **Get Status** module only runs **Start-AzureRmVM** if the current running state is stopped. Finally, You output a message to inform you if the VM was successfully started or not using the PowerShell Write-Output cmdlet.

1. Open **MyFirstRunbook-Graphical** in the graphical editor.
2. Remove the link between **Specify Subscription Id** and **Start-AzureRmVM** by clicking on it and then pressing the *Delete* key.
3. In the Library control, type **Get-AzureRm** in the search textbox.

4. Add **Get-AzureRmVM** to the canvas.
5. Select **Get-AzureRmVM** and then **Parameter Set** to view the sets for **Get-AzureRmVM**. Select the **GetVirtualMachineInResourceGroupNameParamSet** parameter set. **ResourceGroupName** and **Name** have exclamation points next them. This indicates that they are required parameters. Also note both expect string values.
6. Under **Data source** for **Name**, select **Runbook input** and then select **VMName**. Click **OK**.
7. Under **Data source** for **ResourceGroupName**, select **Runbook input** and then select **ResourceGroupName**. Click **OK**.
8. Under **Data source** for **Status**, select **Constant value** and then click **True**. Click **OK**.
9. Create a link from **Specify Subscription Id** to **Get-AzureRmVM**.
10. In the library control, expand **Runbook Control** and add **Code** to the canvas.
11. Create a link from **Get-AzureRmVM** to **Code**.
12. Click **Code** and in the Configuration pane, change label to **Get Status**.
13. Select **Code** parameter, and the **Code Editor** page appears.
14. In the code editor, paste the following snippet of code:

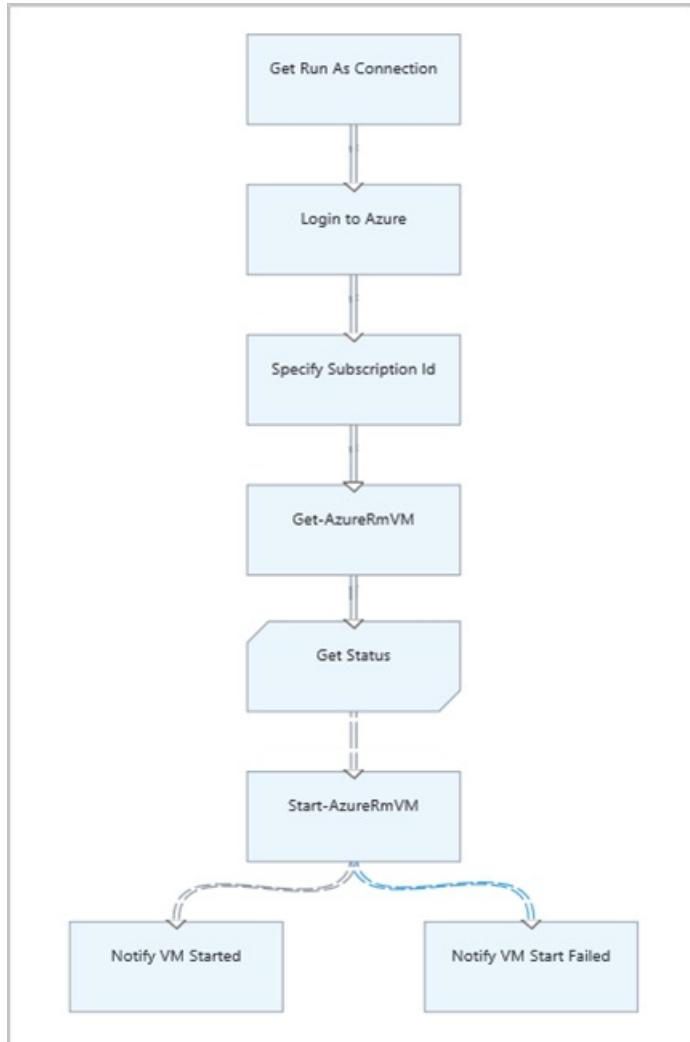
```
$StatusesJson = $ActivityOutput['Get-AzureRmVM'].StatusesText
$Statuses = ConvertFrom-Json $StatusesJson
$StatusOut =""
foreach ($Status in $Statuses){
    if($Status.Code -eq "Powerstate/running"){$StatusOut = "running"}
    elseif ($Status.Code -eq "Powerstate/deallocated") { $StatusOut = "stopped"}
}
$StatusOut
```

15. Create a link from **Get Status** to **Start-AzureRmVM**.



16. Select the link and in the Configuration pane, change **Apply condition** to **Yes**. Note the link turns to a dashed

- line indicating that the target activity only runs if the condition resolves to true.
17. For the **Condition expression**, type `$ActivityOutput['Get Status'] -eq "Stopped"`. **Start-AzureRmVM** now only runs if the virtual machine is stopped.
 18. In the Library control, expand **Cmdlets** and then **Microsoft.PowerShell.Utility**.
 19. Add **Write-Output** to the canvas twice.
 20. On the first **Write-Output** control, click **Parameters** and change the **Label** value to *Notify VM Started*.
 21. For **InputObject**, change **Data source** to **PowerShell expression** and type in the expression `"$VMName successfully started."`.
 22. On the second **Write-Output** control, click **Parameters** and change the **Label** value to *Notify VM Start Failed*.
 23. For **InputObject**, change **Data source** to **PowerShell expression** and type in the expression `"$VMName could not start."`.
 24. Create a link from **Start-AzureRmVM** to **Notify VM Started** and **Notify VM Start Failed**.
 25. Select the link to **Notify VM Started** and change **Apply condition** to **True**.
 26. For the **Condition expression**, type `$ActivityOutput['Start-AzureRmVM'].IsSuccessStatusCode -eq $true`. This Write-Output control now only runs if the virtual machine is successfully started.
 27. Select the link to **Notify VM Start Failed** and change **Apply condition** to **True**.
 28. For the **Condition expression**, type `$ActivityOutput['Start-AzureRmVM'].IsSuccessStatusCode -ne $true`. This Write-Output control now only runs if the virtual machine is not successfully started. Your runbook should look like the following image:



29. Save the runbook and open the Test pane.
30. Start the runbook with the virtual machine stopped, and it should start.

Next steps

- To learn more about Graphical Authoring, see [Graphical authoring in Azure Automation](#)
- To get started with PowerShell runbooks, see [My first PowerShell runbook](#)
- To get started with PowerShell workflow runbooks, see [My first PowerShell workflow runbook](#)

My first PowerShell runbook

7/3/2018 • 7 minutes to read • [Edit Online](#)

This tutorial walks you through the creation of a [PowerShell runbook](#) in Azure Automation. You start with a simple runbook that you test and publish while you learn how to track the status of the runbook job. Then you modify the runbook to actually manage Azure resources, in this case starting an Azure virtual machine. Lastly, you make the runbook more robust by adding runbook parameters.

Prerequisites

To complete this tutorial, you need the following:

- Azure subscription. If you don't have one yet, you can [activate your MSDN subscriber benefits](#) or sign up for a [free account](#).
- [Automation account](#) to hold the runbook and authenticate to Azure resources. This account must have permission to start and stop the virtual machine.
- An Azure virtual machine. You stop and start this machine so it should not be a production VM.

Step 1 - Create new runbook

You start by creating a simple runbook that outputs the text *Hello World*.

1. In the Azure portal, open your Automation account.

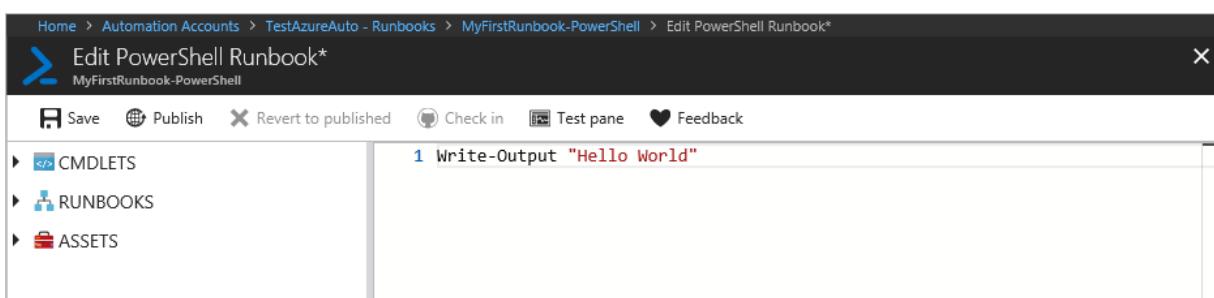
The Automation account page gives you a quick view of the resources in this account. You should already have some assets. Most of those are the modules that are automatically included in a new Automation account. You should also have the Credential asset that's mentioned in the [prerequisites](#).

2. Click **Runbooks** under **Process Automation** to open the list of runbooks.
3. Create a new runbook by clicking the **+ Add a runbook** button and then [Create a new runbook](#).
4. Give the runbook the name *MyFirstRunbook-PowerShell*.
5. In this case, you're going to create a [PowerShell runbook](#) so select **Powershell** for **Runbook type**.
6. Click **Create** to create the runbook and open the textual editor.

Step 2 - Add code to the runbook

You can either type code directly into the runbook, or you can select cmdlets, runbooks, and assets from the Library control and have them added to the runbook with any related parameters. For this walkthrough, you type directly in the runbook.

1. Your runbook is currently empty, type *Write-Output "Hello World."* in the body of the script.



```
1 Write-Output "Hello World"
```

2. Save the runbook by clicking **Save**.

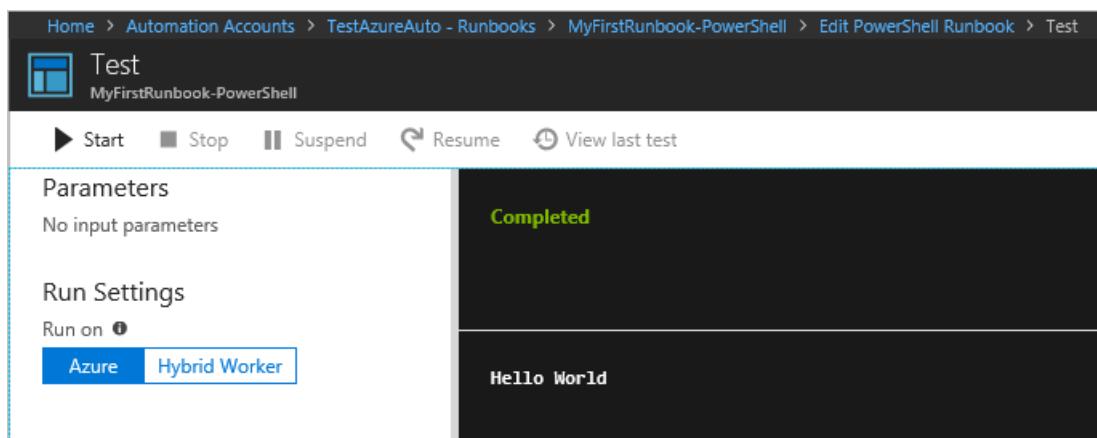
Step 3 - Test the runbook

Before you publish the runbook to make it available in production, you want to test it to make sure that it works properly. When you test a runbook, you run its **Draft** version and view its output interactively.

1. Click **Test pane** to open the Test pane.
2. Click **Start** to start the test. This should be the only enabled option.
3. A **runbook job** is created and its status displayed.

The job status starts as *Queued* indicating that it is waiting for a runbook worker in the cloud to come available. It moves to *Starting* when a worker claims the job, and then *Running* when the runbook actually starts running.

4. When the runbook job completes, its output is displayed. In your case, you should see *Hello World*.



5. Close the Test pane to return to the canvas.

Step 4 - Publish and start the runbook

The runbook that you created is still in Draft mode. You need to publish it before you can run it in production. When you publish a runbook, you overwrite the existing Published version with the Draft version. In your case, you don't have a Published version yet because you just created the runbook.

1. Click **Publish** to publish the runbook and then **Yes** when prompted.
2. If you scroll left to view the runbook in the **Runbooks** pane now, it shows an **Authoring Status** of **Published**.
3. Scroll back to the right to view the pane for **MyFirstRunbook-PowerShell**.
The options across the top allow us to start the runbook, view the runbook, schedule it to start at some time in the future, or create a **webhook** so it can be started through an HTTP call.
4. You want to start the runbook, so click **Start** and then click **Ok** when the Start Runbook page opens.
5. A job page is opened for the runbook job that you created. You can close this pane, but in this case you leave it open so you can watch the job's progress.
6. The job status is shown in **Job Summary** and matches the statuses that you saw when you tested the runbook.

MyFirstRunbook-PowerShell 1/24/2018 11:20 AM

Job

► Resume ■ Stop ■ Suspend

Essentials ^

Job Id 1c072925-05ca-4ce6-8b3e-7214d3e19e0a	Created 1/24/2018 11:20 AM
Job status Completed	Last Update 1/24/2018 11:20 AM
Run As User	Runbook MyFirstRunbook-PowerShell
Ran on Azure	Source snapshot View source snapshot

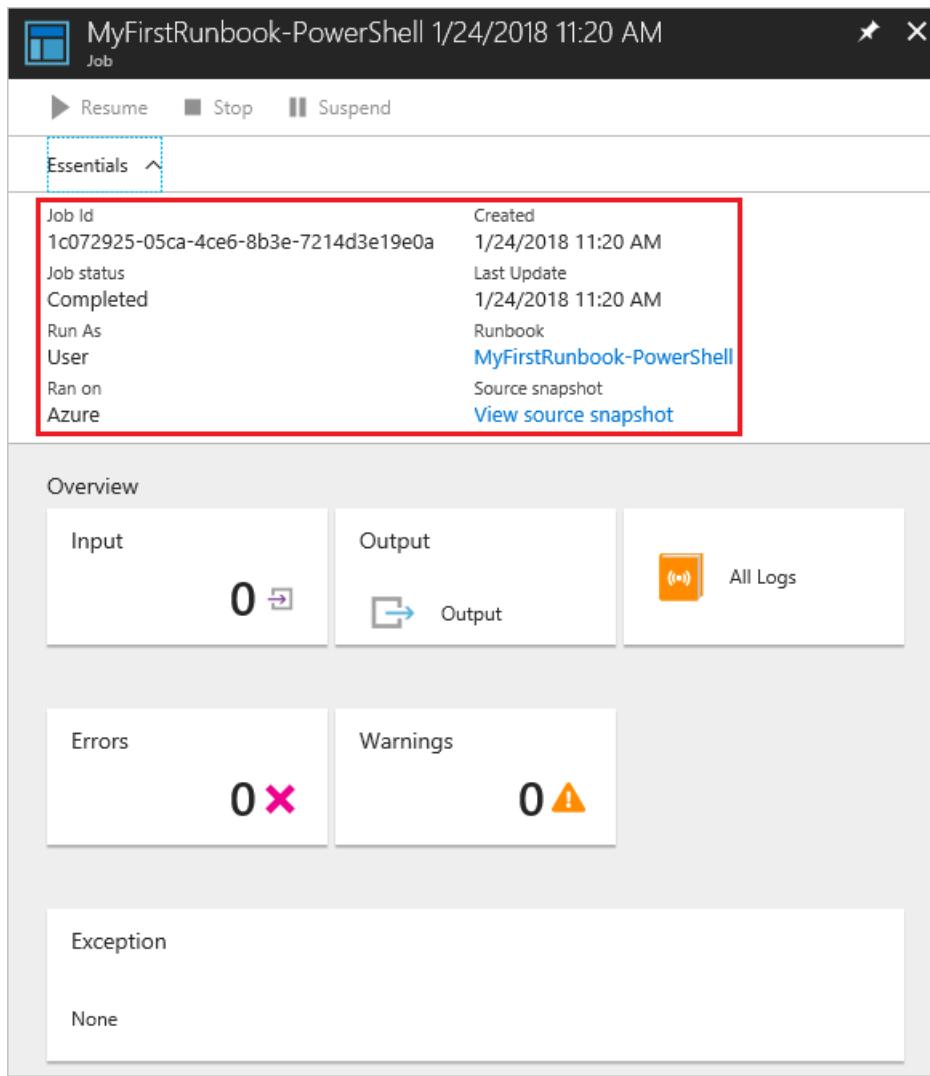
Overview

Input 0 ↗ Output 0 ➡ All Logs

Errors 0 ✘ Warnings 0 !

Exception

None

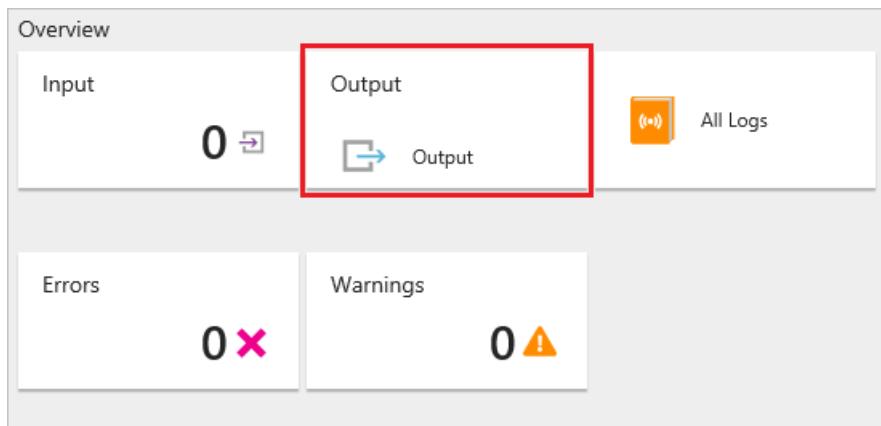


- Once the runbook status shows *Completed*, under **Overview** click **Output**. The Output pane is opened, and you can see your *Hello World*.

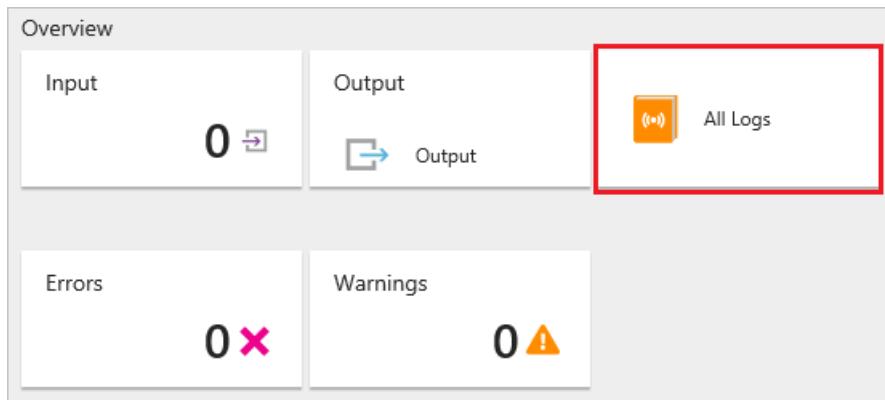
Overview

Input 0 ↗ Output 0 ➡ All Logs

Errors 0 ✘ Warnings 0 !



- Close the Output page.
- Click **All Logs** to open the Streams pane for the runbook job. You should only see *Hello World* in the output stream, but this can show other streams for a runbook job such as Verbose and Error if the runbook writes to them.



10. Close the Streams page and the Job page to return to the MyFirstRunbook-PowerShell page.
11. Under **Details**, click **Jobs** to open the Jobs pane for this runbook. This lists all of the jobs created by this runbook. You should only see one job listed since you only ran the job once.

12. You can click this job to open the same Job pane that you viewed when you started the runbook. This allows you to go back in time and view the details of any job that was created for a particular runbook.

Step 5 - Add authentication to manage Azure resources

You've tested and published your runbook, but so far it doesn't do anything useful. You want to have it manage Azure resources. It is not able to do that though unless you have it authenticate using the credentials that are referred to in the [prerequisites](#). You do that with the **Connect-AzureRmAccount** cmdlet.

1. Open the textual editor by clicking **Edit** on the MyFirstRunbook-PowerShell page.
2. You don't need the **Write-Output** line anymore, so go ahead and delete it.
3. Type or copy and paste the following code that handles the authentication with your Automation Run As account:

```
$Conn = Get-AutomationConnection -Name AzureRunAsConnection
Connect-AzureRmAccount -ServicePrincipal -Tenant $Conn.TenantID ` 
-ApplicationId $Conn.ApplicationID -CertificateThumbprint $Conn.CertificateThumbprint
```

IMPORTANT

Add-AzureRmAccount and **Login-AzureRmAccount** are now aliases for **Connect-AzureRMAccount**. If the **Connect-AzureRMAccount** cmdlet does not exist, you can use **Add-AzureRmAccount** or **Login-AzureRmAccount**, or you can update your modules in your Automation Account to the latest versions.

4. Click **Test pane** so that you can test the runbook.
5. Click **Start** to start the test. Once it completes, you should receive output similar to the following, displaying basic information from your account. This confirms that the credential is valid.

The screenshot shows the 'Test pane' results. At the top, it says 'Completed'. Below that, there's a table-like structure with two columns: 'Environments' and 'Context'. Under 'Environments', it lists several cloud environments separated by commas: [[AzureCloud, AzureCloud], [AzureChinaCloud, AzureChinaCloud], [AzureUSGovernment, AzureUSGovernment]]}. Microsoft.Azur...'. Under 'Context', it shows a single entry: Microsoft.Azur...

Step 6 - Add code to start a virtual machine

Now that your runbook is authenticating to your Azure subscription, you can manage resources. You add a command to start a virtual machine. You can pick any virtual machine in your Azure subscription, and for now you hardcode that name in the runbook.

1. After **Connect-AzureRmAccount**, type **Start-AzureRmVM -Name 'VMName' -ResourceGroupName 'NameofResourceGroup'** providing the name and Resource Group name of the virtual machine to start.

```
$Conn = Get-AutomationConnection -Name AzureRunAsConnection
Connect-AzureRmAccount -ServicePrincipal -Tenant $Conn.TenantID ` 
-ApplicationID $Conn.ApplicationID -CertificateThumbprint $Conn.CertificateThumbprint
Start-AzureRmVM -Name 'VMName' -ResourceGroupName 'ResourceGroupName'
```

2. Save the runbook and then click **Test pane** so that you can test it.
3. Click **Start** to start the test. Once it completes, check that the virtual machine was started.

Step 7 - Add an input parameter to the runbook

Your runbook currently starts the virtual machine that you hardcoded in the runbook, but it would be more useful if you specify the virtual machine when the runbook is started. You add input parameters to the runbook to provide that functionality.

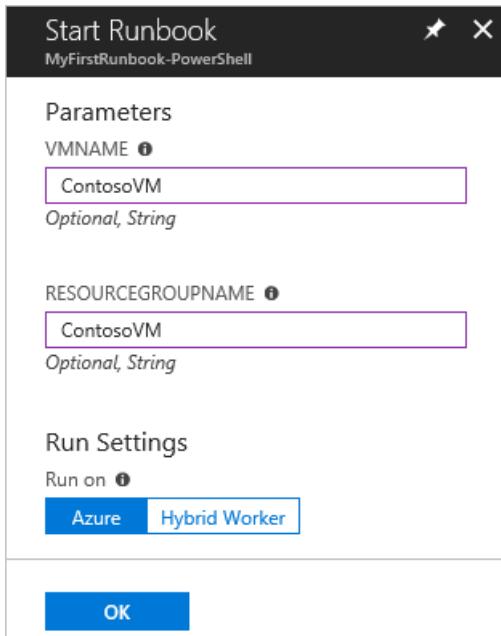
1. Add parameters for **VMName** and **ResourceGroupName** to the runbook and use these variables with the **Start-AzureRmVM** cmdlet as in the following example.

```

Param(
    [string]$VMName,
    [string]$ResourceGroupName
)
$Conn = Get-AutomationConnection -Name AzureRunAsConnection
Connect-AzureRmAccount -ServicePrincipal -Tenant $Conn.TenantID ` 
    -ApplicationID $Conn.ApplicationID -CertificateThumbprint $Conn.CertificateThumbprint
Start-AzureRmVM -Name $VMName -ResourceGroupName $ResourceGroupName

```

2. Save the runbook and open the Test pane. You can now provide values for the two input variables that are used in the test.
3. Close the Test pane.
4. Click **Publish** to publish the new version of the runbook.
5. Stop the virtual machine that you started in the previous step.
6. Click **OK** to start the runbook. Type in the **VMName** and **ResourceGroupName** for the virtual machine that you're going to start.



7. When the runbook completes, check that the virtual machine was started.

Differences from 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 whereas PowerShell runbooks can only resume from the beginning.
3. PowerShell Workflow runbooks support parallel and serial execution whereas PowerShell runbooks can only execute commands serially.
4. In a PowerShell Workflow runbook, an activity, a command, or a script block can have its own runspace whereas in a PowerShell runbook, everything in a script runs in a single runspace. There are also some [syntactic differences](#) between a native PowerShell runbook and a PowerShell Workflow runbook.

Next steps

- To get started with Graphical runbooks, see [My first graphical runbook](#)
- To get started with PowerShell workflow runbooks, see [My first PowerShell workflow runbook](#)
- To know more about runbook types, their advantages and limitations, see [Azure Automation runbook types](#)
- For more information on PowerShell script support feature, see [Native PowerShell script support in Azure Automation](#)

My first PowerShell Workflow runbook

7/9/2018 • 7 minutes to read • [Edit Online](#)

This tutorial walks you through the creation of a [PowerShell Workflow runbook](#) in Azure Automation. You start with a simple runbook that you test and publish while explaining how to track the status of the runbook job. Then you modify the runbook to actually manage Azure resources, in this case starting an Azure virtual machine. Lastly you make the runbook more robust by adding runbook parameters.

Prerequisites

To complete this tutorial, you need the following:

- Azure subscription. If you don't have one yet, you can [activate your MSDN subscriber benefits](#) or sign up for a [free account](#).
- [Automation account](#) to hold the runbook and authenticate to Azure resources. This account must have permission to start and stop the virtual machine.
- An Azure virtual machine. You stop and start this machine so it should not be a production VM.

Step 1 - Create new runbook

You start by creating a simple runbook that outputs the text *Hello World*.

1. In the Azure portal, open your Automation account.

The Automation account page gives you a quick view of the resources in this account. You should already have some assets. Most of those are the modules that are automatically included in a new Automation account. You should also have the Credential asset that's mentioned in the [prerequisites](#).

2. Click **Runbooks** under **Process Automation** to open the list of runbooks.
3. Create a new runbook by clicking on the **+ Add a runbook** button and then **Create a new runbook**.
4. Give the runbook the name *MyFirstRunbook-Workflow*.
5. In this case, you're going to create a [PowerShell Workflow runbook](#) so select **Powershell Workflow** for **Runbook type**.
6. Click **Create** to create the runbook and open the textual editor.

Step 2 - Add code to the runbook

You can either type code directly into the runbook, or you can select cmdlets, runbooks, and assets from the Library control and have them added to the runbook with any related parameters. For this walkthrough, you type directly into the runbook.

1. Your runbook is currently empty with only the required *workflow* keyword, the name of your runbook, and the braces that encases the entire workflow.

```
Workflow MyFirstRunbook-Workflow
{}
```

2. Type *Write-Output "Hello World."* between the braces.

```
Workflow MyFirstRunbook-Workflow
{
    Write-Output "Hello World"
}
```

3. Save the runbook by clicking **Save**.

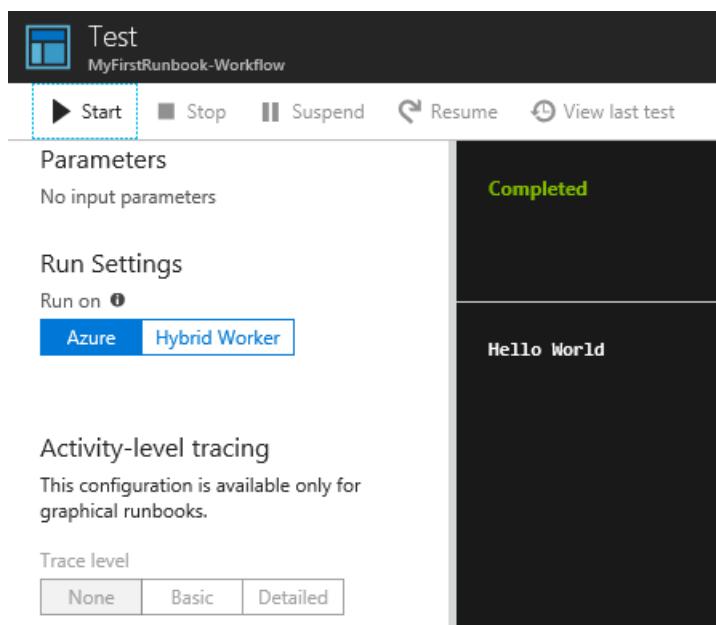
Step 3 - Test the runbook

Before you publish the runbook to make it available in production, you want to test it to make sure that it works properly. When you test a runbook, you run its **Draft** version and view its output interactively.

1. Click **Test pane** to open the Test pane.
2. Click **Start** to start the test. This should be the only enabled option.
3. A **runbook job** is created and its status displayed.

The job status will start as *Queued* indicating that it is waiting for a runbook worker in the cloud to come available. It moves to *Starting* when a worker claims the job, and then *Running* when the runbook actually starts running.

4. When the runbook job completes, its output is displayed. In your case, you should see *Hello World*.



5. Close the Test pane to return to the canvas.

Step 4 - Publish and start the runbook

The runbook that you created is still in Draft mode. You need to publish it before you can run it in production. When you publish a runbook, you overwrite the existing Published version with the Draft version. In your case, you don't have a Published version yet because you just created the runbook.

1. Click **Publish** to publish the runbook and then **Yes** when prompted.
2. If you scroll left to view the runbook in the **Runbooks** pane now, it shows an **Authoring Status** of **Published**.
3. Scroll back to the right to view the pane for **MyFirstRunbook-Workflow**.
The options across the top allow us to start the runbook, schedule it to start at some time in the future, or create a [webhook](#) so it can be started through an HTTP call.
4. you just want to start the runbook so click **Start** and then **Yes** when prompted.



5. A job pane is opened for the runbook job that you created. You can close this pane, but in this case you leave it open so you can watch the job's progress.
6. The job status is shown in **Job Summary** and matches the statuses that you saw when you tested the runbook.

The screenshot shows the 'Job Summary' page for a runbook named 'MyFirstRunbook-Workflow' from 1/24/2018 1:33 PM. The 'Essentials' section is expanded, displaying the following details:

Job Id	3eb16ac6-09ff-4037-b67e-61b961c6ff20	Created	1/24/2018 1:33 PM
Job status	Completed	Last Update	1/24/2018 1:34 PM
Run As	User	Runbook	MyFirstRunbook-Workflow
Ran on	Azure	Source snapshot	View source snapshot

The 'Overview' section includes metrics for Input (0), Output (0), and All Logs (0). It also shows Error (0) and Warning (0) counts. The 'Exception' section indicates 'None'.

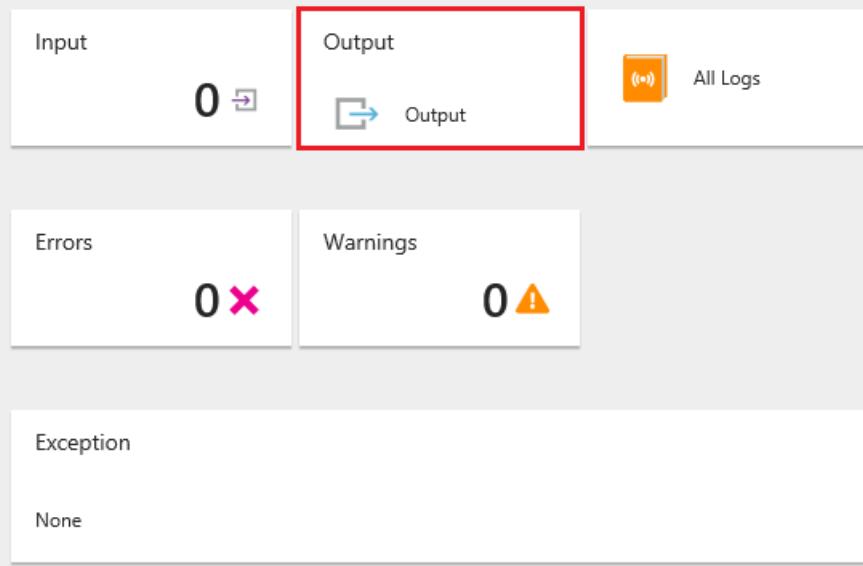
7. Once the runbook status shows *Completed*, click **Output**. The Output pane is opened, and you can see your *Hello World*.

▶ Resume ■ Stop || Suspend

Essentials ^

Job Id	Created
3eb16ac6-09ff-4037-b67e-61b961c6ff20	1/24/2018 1:33 PM
Job status	Last Update
Completed	1/24/2018 1:34 PM
Run As	Runbook
User	MyFirstRunbook-Workflow
Ran on	Source snapshot
Azure	View source snapshot

Overview



8. Close the Output pane.
9. Click **All Logs** to open the Streams pane for the runbook job. You should only see *Hello World* in the output stream, but this can show other streams for a runbook job such as Verbose and Error if the runbook writes to them.



Job

Resume Stop Suspend

Essentials ^

Job Id 3eb16ac6-09ff-4037-b67e-61b961c6ff20	Created 1/24/2018 1:33 PM
Job status Completed	Last Update 1/24/2018 1:34 PM
Run As User	Runbook MyFirstRunbook-Workflow
Ran on Azure	Source snapshot View source snapshot

Overview

Input 0 ↗	Output ↗ Output	All Logs
Errors 0 ✘	Warnings 0 ⚠	
Exception		
None		

10. Close the Streams pane and the Job pane to return to the MyFirstRunbook pane.
11. Click **Jobs** to open the Jobs pane for this runbook. This lists all of the jobs created by this runbook. you should only see one job listed since you only ran the job once.

Details

Jobs	Schedules	Webhooks
0	0	0

12. You can click on this job to open the same Job pane that you viewed when you started the runbook. This allows you to go back in time and view the details of any job that was created for a particular runbook.

Step 5 - Add authentication to manage Azure resources

you've tested and published your runbook, but so far it doesn't do anything useful. you want to have it manage Azure resources. It won't be able to do that though unless you have it authenticate using the credentials that are referred to in the [prerequisites](#). you do that with the **Connect-AzureRmAccount** cmdlet.

1. Open the textual editor by clicking **Edit** on the MyFirstRunbook-Workflow pane.
2. you don't need the **Write-Output** line anymore, so go ahead and delete it.
3. Position the cursor on a blank line between the braces.

- Type or copy and paste the following code that will handle the authentication with your Automation Run As account:

```
$Conn = Get-AutomationConnection -Name AzureRunAsConnection  
Connect-AzureRmAccount -ServicePrincipal -Tenant $Conn.TenantID `  
-ApplicationId $Conn.ApplicationID -CertificateThumbprint $Conn.CertificateThumbprint
```

IMPORTANT

Add-AzureRmAccount and **Login-AzureRmAccount** are now aliases for **Connect-AzureRMAccount**. If the **Connect-AzureRMAccount** cmdlet does not exist, you can use **Add-AzureRmAccount** or **Login-AzureRmAccount**, or you can [update your modules](#) in your Automation Account to the latest versions.

NOTE

You might need to [update your modules](#) even though you have just created a new automation account.

- Click **Test pane** so that you can test the runbook.
- Click **Start** to start the test. Once it completes, you should receive output similar to the following, displaying basic information from your account. This confirms that the credential is valid.



Step 6 - Add code to start a virtual machine

Now that your runbook is authenticating to your Azure subscription, you can manage resources. You add a command to start a virtual machine. You can pick any virtual machine in your Azure subscription, and for now you are hardcoding that name in the runbook.

- After `Connect-AzureRmAccount`, type `Start-AzureRmVM -Name 'VMName' -ResourceGroupName 'NameofResourceGroup'` providing the name and Resource Group name of the virtual machine to start.

```
workflow MyFirstRunbook-Workflow  
{  
$Conn = Get-AutomationConnection -Name AzureRunAsConnection  
Connect-AzureRmAccount -ServicePrincipal -Tenant $Conn.TenantID -ApplicationId $Conn.ApplicationID -  
CertificateThumbprint $Conn.CertificateThumbprint  
Start-AzureRmVM -Name 'VMName' -ResourceGroupName 'ResourceGroupName'  
}
```

- Save the runbook and then click **Test pane** so that you can test it.
- Click **Start** to start the test. Once it completes, check that the virtual machine was started.

Step 7 - Add an input parameter to the runbook

your runbook currently starts the virtual machine that you hardcoded in the runbook, but it would be more useful if you could specify the virtual machine when the runbook is started. You add input parameters to the runbook to provide that functionality.

1. Add parameters for **VMName** and **ResourceGroupName** to the runbook and use these variables with the **Start-AzureRmVM** cmdlet as in the example below.

```
workflow MyFirstRunbook-Workflow
{
    Param(
        [string]$VMName,
        [string]$ResourceGroupName
    )
    $Conn = Get-AutomationConnection -Name AzureRunAsConnection
    Connect-AzureRmAccount -ServicePrincipal -Tenant $Conn.TenantID -ApplicationId $Conn.ApplicationID -CertificateThumbprint $Conn.CertificateThumbprint
    Start-AzureRmVM -Name $VMName -ResourceGroupName $ResourceGroupName
}
```

2. Save the runbook and open the Test pane. You can now provide values for the two input variables that are in the test.
3. Close the Test pane.
4. Click **Publish** to publish the new version of the runbook.
5. Stop the virtual machine that you started in the previous step.
6. Click **Start** to start the runbook. Type in the **VMName** and **ResourceGroupName** for the virtual machine that you're going to start.

Parameters

VMNAME ⓘ
ContosoVM1
Optional, String

RESOURCEGROUPNAME ⓘ
ContosoVM
Optional, String

Run Settings

Run on ⓘ
Azure Hybrid Worker

7. When the runbook completes, check that the virtual machine was started.

Next steps

- To get started with Graphical runbooks, see [My first graphical runbook](#)
- To get started with PowerShell runbooks, see [My first PowerShell runbook](#)
- To learn more about runbook types, their advantages and limitations, see [Azure Automation runbook types](#)
- For more information on PowerShell script support feature, see [Native PowerShell script support in Azure Automation](#)

My first Python runbook

6/26/2018 • 7 minutes to read • [Edit Online](#)

This tutorial walks you through the creation of a [Python runbook](#) in Azure Automation. You start with a simple runbook that you test and publish. Then you modify the runbook to actually manage Azure resources, in this case starting an Azure virtual machine. Lastly, you make the runbook more robust by adding runbook parameters.

Prerequisites

To complete this tutorial, you need the following:

- Azure subscription. If you don't have one yet, you can [activate your MSDN subscriber benefits](#) or sign up for a [free account](#).
- [Automation account](#) to hold the runbook and authenticate to Azure resources. This account must have permission to start and stop the virtual machine.
- An Azure virtual machine. You stop and start this machine so it should not be a production VM.

Create a new runbook

You start by creating a simple runbook that outputs the text *Hello World*.

1. In the Azure portal, open your Automation account.

The Automation account page gives you a quick view of the resources in this account. You should already have some assets. Most of those assets are the modules that are automatically included in a new Automation account. You should also have the Credential asset that's mentioned in the [prerequisites](#).

2. Select **Runbooks** under **PROCESS MANAGEMENT** to open the list of runbooks.
3. Select **+ Add a runbook** to create a new runbook.
4. Give the runbook the name *MyFirstRunbook-Python*.
5. In this case, you're going to create a [Python runbook](#) so select **Python 2** for **Runbook type**.
6. Click **Create** to create the runbook and open the textual editor.

Add code to the runbook

Now you add a simple command to print the text "Hello World":

```
print("Hello World!")
```

Click **Save** to save the runbook.

Test the runbook

Before you publish the runbook to make it available in production, you want to test it to make sure that it works properly. When you test a runbook, you run its **Draft** version and view its output interactively.

1. Click **Test pane** to open the Test pane.
2. Click **Start** to start the test. This should be the only enabled option.
3. A [runbook job](#) is created and its status displayed. The job status starts as *Queued* indicating that it is waiting for a runbook worker in the cloud to come available. It moves to *Starting* when a worker claims the job, and

then *Running* when the runbook actually starts running.

4. When the runbook job completes, its output is displayed. In this case, you should see *Hello World*.
5. Close the Test pane to return to the canvas.

Publish and start the runbook

The runbook that you created is still in draft mode. You need to publish it before you can run it in production. When you publish a runbook, you overwrite the existing published version with the Draft version. In this case, you don't have a published version yet because you just created the runbook.

1. Click **Publish** to publish the runbook and then **Yes** when prompted.
2. If you scroll left to view the runbook in the **Runbooks** pane now, it shows an **Authoring Status** of **Published**.
3. Scroll back to the right to view the pane for **MyFirstRunbook-Python**. The options across the top allow us to start the runbook, view the runbook, schedule it to start at some time in the future, or create a [webhook](#) so it can be started through an HTTP call.
4. You want to start the runbook, so click **Start** and then click **Ok** when the Start Runbook blade opens.
5. A job pane is opened for the runbook job that you created. You can close this pane, but in this case you leave it open so you can watch the job's progress.
6. The job status is shown in **Job Summary** and matches the statuses that you saw when you tested the runbook.
7. Once the runbook status shows *Completed*, click **Output**. The Output pane is opened, and you can see your *Hello World*.
8. Close the Output pane.
9. Click **All Logs** to open the Streams pane for the runbook job. You should only see *Hello World* in the output stream, but this can show other streams for a runbook job such as Verbose and Error if the runbook writes to them.
10. Close the Streams pane and the Job pane to return to the MyFirstRunbook-Python pane.
11. Click **Jobs** to open the Jobs pane for this runbook. This lists all of the jobs created by this runbook. You should only see one job listed since you only ran the job once.
12. You can click this job to open the same Job pane that you viewed when you started the runbook. This allows you to go back in time and view the details of any job that was created for a particular runbook.

Add authentication to manage Azure resources

You've tested and published your runbook, but so far it doesn't do anything useful. You want to have it manage Azure resources. To manage Azure resources, the script has to authenticate using the credentials from your [Automation account](#).

NOTE

The Automation account must have been created with the service principal feature for there to be a runas certificate. If your automation account was not created with the service principal, you can authenticate by using the method described at [Authenticate with the Azure Management Libraries for Python](#).

1. Open the textual editor by clicking **Edit** on the MyFirstRunbook-Python pane.
2. Add the following code to authenticate to Azure:

```

import os
from azure.mgmt.compute import ComputeManagementClient
import azure.mgmt.resource
import automationassets

def get_automation_runas_credential(runas_connection):
    from OpenSSL import crypto
    import binascii
    from msrestazure import azure_active_directory
    import adal

    # Get the Azure Automation RunAs service principal certificate
    cert = automationassets.get_automation_certificate("AzureRunAsCertificate")
    pks12_cert = crypto.load_pkcs12(cert)
    pem_pkey = crypto.dump_privatekey(crypto.FILETYPE_PEM,pks12_cert.get_privatekey())

    # Get run as connection information for the Azure Automation service principal
    application_id = runas_connection["ApplicationId"]
    thumbprint = runas_connection["CertificateThumbprint"]
    tenant_id = runas_connection["TenantId"]

    # Authenticate with service principal certificate
    resource ="https://management.core.windows.net/"
    authority_url = ("https://login.microsoftonline.com/" +tenant_id)
    context = adal.AuthenticationContext(authority_url)
    return azure_active_directory.AdalAuthentication(
        lambda: context.acquire_token_with_client_certificate(
            resource,
            application_id,
            pem_pkey,
            thumbprint)
    )

# Authenticate to Azure using the Azure Automation RunAs service principal
runas_connection = automationassets.get_automation_connection("AzureRunAsConnection")
azure_credential = get_automation_runas_credential(runas_connection)

```

Add code to create Python Compute client and start the VM

To work with Azure VMs, create an instance of the [Azure Compute client for Python](#).

Use the Compute client to start the VM. Add the following code to the runbook:

```

# Initialize the compute management client with the RunAs credential and specify the subscription to work
against.
compute_client = ComputeManagementClient(
    azure_credential,
    str(runas_connection["SubscriptionId"])
)

print('\nStart VM')
async_vm_start = compute_client.virtual_machines.start("MyResourceGroup", "TestVM")
async_vm_start.wait()

```

Where *MyResourceGroup* is the name of the resource group that contains the VM, and *TestVM* is the name of the VM you want to start.

Test and run the runbook again to see that it starts the VM.

Use input parameters

The runbook currently uses hard-coded values for the names of the Resource Group and the VM. Now let's add code that gets these values from input parameters.

You use the `sys.argv` variable to get the parameter values. Add the following code to the runbook immediately after the other `import` statements:

```
import sys

resource_group_name = str(sys.argv[1])
vm_name = str(sys.argv[2])
```

This imports the `sys` module, and creates two variables to hold the Resource Group and VM names. Notice that the element of the argument list, `sys.argv[0]`, is the name of the script, and is not input by the user.

Now you can modify the last two lines of the runbook to use the input parameter values instead of using hard-coded values:

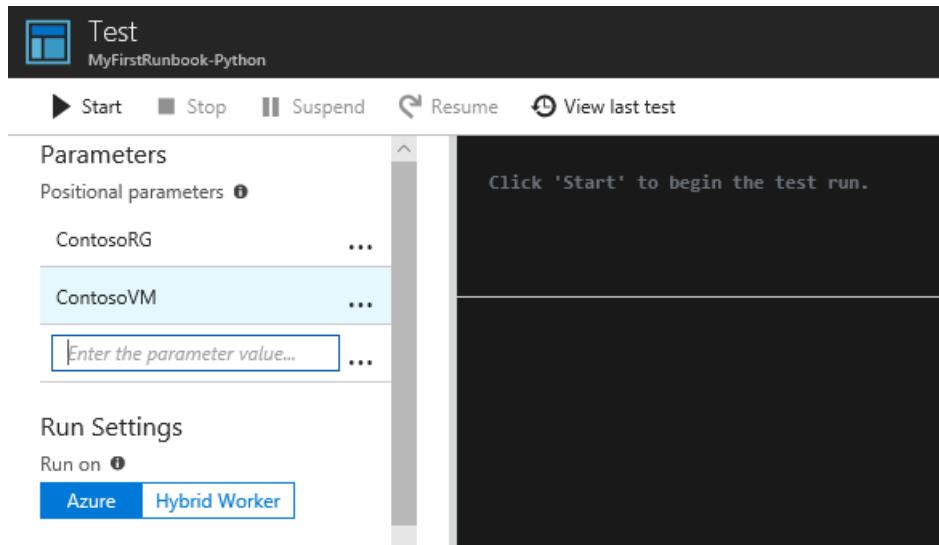
```
async_vm_start = compute_client.virtual_machines.start(resource_group_name, vm_name)
async_vm_start.wait()
```

When you start a Python runbook (either on the **Test** page or as a published runbook), you can enter the values for parameters in the **Start Runbook** page under **Parameters**.

After you start entering a value in the first box, a second will appear, and so on, so that you can enter as many parameter values as necessary.

The values are available to the script as the `sys.argv` array as in the code you just added.

Enter the name of your resource group as the value for the first parameter, and the name of the VM to start as the value of the second parameter.



Click **OK** to start the runbook. The runbook runs and starts the VM that you specified.

Next steps

- To get started with PowerShell runbooks, see [My first PowerShell runbook](#)
- To get started with Graphical runbooks, see [My first graphical runbook](#)
- To get started with PowerShell workflow runbooks, see [My first PowerShell workflow runbook](#)
- To know more about runbook types, their advantages and limitations, see [Azure Automation runbook types](#)
- To learn about developing for Azure with Python, see [Azure for Python developers](#)

- To view sample Python 2 runbooks, see the [Azure Automation GitHub](#)

Editing textual runbooks in Azure Automation

5/21/2018 • 3 minutes to read • [Edit Online](#)

The textual editor in Azure Automation can be used to edit [PowerShell runbooks](#) and [PowerShell Workflow runbooks](#). This has the typical features of other code editors such as intellisense and color coding with additional special features to assist you in accessing resources common to runbooks. This article provides detailed steps for performing different functions with this editor.

The textual editor includes a feature to insert code for cmdlets, assets, and child runbooks into a runbook. Rather than typing in the code yourself, you can select from a list of available resources and have the appropriate code inserted into the runbook.

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. The Published version cannot be edited. See [Publishing a runbook](#) for more information.

To work with [Graphical Runbooks](#), see [Graphical authoring in Azure Automation](#).

To edit a runbook with the Azure portal

Use the following procedure to open a runbook for editing in the textual editor.

1. In the Azure portal, select your automation account.
2. Under **PROCESS AUTOMATION**, select **Runbooks** to open the list of runbooks.
3. Select the runbook you want to edit and then click the **Edit** button.
4. Perform the required editing.
5. Click **Save** when your edits are complete.
6. Click **Publish** if you want the latest draft version of the runbook to be published.

To insert a cmdlet into a runbook

1. In the Canvas of the textual editor, position the cursor where you want to place the cmdlet.
2. Expand the **Cmdlets** node in the Library control.
3. Expand the module containing the cmdlet you want to use.
4. Right click the cmdlet to insert and select **Add to canvas**. If the cmdlet has more than one parameter set, then the default set will be added. You can also expand the cmdlet to select a different parameter set.
5. The code for the cmdlet is inserted with its entire list of parameters.
6. Provide an appropriate value in place of the data type surrounded by braces <> for any required parameters. Remove any parameters you don't need.

To insert code for a child runbook into a runbook

1. In the Canvas of the textual editor, position the cursor where you want to place the code for the [child runbook](#).
2. Expand the **Runbooks** node in the Library control.
3. Right click the runbook to insert and select **Add to canvas**.
4. The code for the child runbook is inserted with any placeholders for any runbook parameters.
5. Replace the placeholders with appropriate values for each parameter.

To insert an asset into a runbook

1. In the Canvas of the textual editor, position the cursor where you want to place the code for the child runbook.
2. Expand the **Assets** node in the Library control.

3. Expand the node for the type of asset you want.
4. Right click the asset to insert and select **Add to canvas**. For [variable assets](#), select either **Add "Get Variable" to canvas** or **Add "Set Variable" to canvas** depending on whether you want to get or set the variable.
5. The code for the asset is inserted into the runbook.

To edit an Azure Automation runbook using Windows PowerShell

To edit a runbook with Windows PowerShell, you use the editor of your choice and save it to a .ps1 file. You can use the [Get-AzureAutomationRunbookDefinition](#) cmdlet to retrieve the contents of the runbook and then [Set-AzureAutomationRunbookDefinition](#) cmdlet to replace the existing draft runbook with the modified one.

To Retrieve the Contents of a Runbook Using Windows PowerShell

The following sample commands show how to retrieve the script for a runbook and save it to a script file. In this example, the Draft version is retrieved. It is also possible to retrieve the Published version of the runbook although this version cannot be changed.

```
$automationAccountName = "MyAutomationAccount"
$runbookName = "Sample-TestRunbook"
$scriptPath = "c:\runbooks\Sample-TestRunbook.ps1"

$runbookDefinition = Get-AzureAutomationRunbookDefinition -AutomationAccountName $automationAccountName -Name $runbookName -Slot Draft
$runbookContent = $runbookDefinition.Content

Out-File -InputObject $runbookContent -FilePath $scriptPath
```

To Change the Contents of a Runbook Using Windows PowerShell

The following sample commands show how to replace the existing contents of a runbook with the contents of a script file. Note that this is the same sample procedure as in [To import a runbook from a script file with Windows PowerShell](#).

```
$automationAccountName = "MyAutomationAccount"
$runbookName = "Sample-TestRunbook"
$scriptPath = "c:\runbooks\Sample-TestRunbook.ps1"

Set-AzureAutomationRunbookDefinition -AutomationAccountName $automationAccountName -Name $runbookName -Path $scriptPath -Overwrite
Publish-AzureAutomationRunbook -AutomationAccountName $automationAccountName -Name $runbookName
```

Related articles

- [Creating or importing a runbook in Azure Automation](#)
- [Learning PowerShell workflow](#)
- [Graphical authoring in Azure Automation](#)
- [Certificates](#)
- [Connections](#)
- [Credentials](#)
- [Schedules](#)
- [Variables](#)

Graphical authoring in Azure Automation

7/3/2018 • 22 minutes to read • [Edit Online](#)

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. If you have ever worked with System Center Orchestrator or Service Management Automation (SMA), then this should look familiar to you.

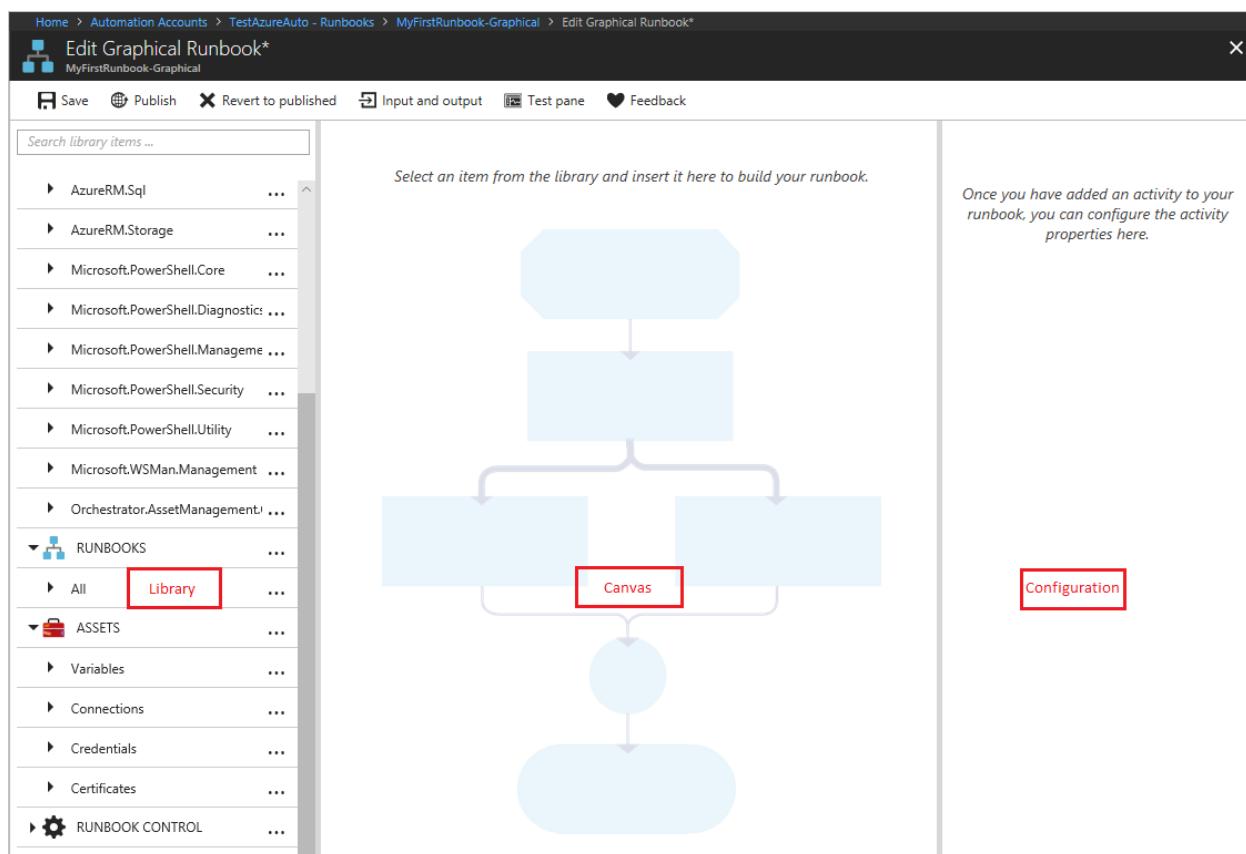
This article provides an introduction to graphical authoring and the concepts you need to get started in creating a graphical runbook.

Graphical runbooks

All runbooks in Azure Automation are Windows PowerShell Workflows. Graphical and Graphical PowerShell Workflow runbooks generate PowerShell code that is run by the Automation workers, but you are not able to view it or directly modify it. A Graphical runbook can be converted to a Graphical PowerShell Workflow runbook and vice-versa, but they cannot be converted to a textual runbook. An existing textual runbook cannot be imported into the graphical editor.

Overview of graphical editor

You can open the graphical editor in the Azure portal by creating or editing a graphical runbook.



The following sections describe the controls in the graphical editor.

Canvas

The Canvas is where you design your runbook. You add activities from the nodes in the Library control to the runbook and connect them with links to define the logic of the runbook.

You can use the controls at the bottom of the canvas to zoom in and out.

Library control

The Library control is where you select [activities](#) to add to your runbook. You add them to the canvas where you connect them to other activities. It includes four sections described in the following table:

SECTION	DESCRIPTION
Cmdlets	Includes all the cmdlets that can be used in your runbook. Cmdlets are organized by module. All of the modules that you have installed in your automation account are available.
Runbooks	Includes the runbooks in your automation account. These runbooks can be added to the canvas to be used as child runbooks. Only runbooks of the same core type as the runbook being edited are shown; for Graphical runbooks only PowerShell-based runbooks are shown, while for Graphical PowerShell Workflow runbooks only PowerShell-Workflow-based runbooks are shown.
Assets	Includes the automation assets in your automation account that can be used in your runbook. When you add an asset to a runbook, it adds a workflow activity that gets the selected asset. In the case of variable assets, you can select whether to add an activity to get the variable or set the variable.
Runbook Control	Includes runbook control activities that can be used in your current runbook. A <i>Junction</i> takes multiple inputs and waits until all have completed before continuing the workflow. A <i>Code</i> activity runs one or more lines of PowerShell or PowerShell Workflow code depending on the graphical runbook type. You can use this activity for custom code or for functionality that is difficult to achieve with other activities.

Configuration control

The Configuration control is where you provide details for an object selected on the canvas. The properties available in this control depends on the type of object selected. When you select an option in the Configuration control, it opens additional blades in order to provide additional information.

Test control

The Test control is not displayed when the graphical editor is first started. It is opened when you interactively [test a graphical runbook](#).

Graphical runbook procedures

Exporting and importing a graphical runbook

You can only export the published version of a graphical runbook. If the runbook has not yet been published, then the **Export** button is disabled. When you click the **Export** button, the runbook is downloaded to your local computer. The name of the file matches the name of the runbook with a *graphrunbook* extension.

You can import a Graphical or Graphical PowerShell Workflow runbook file by selecting the **Import** option when adding a runbook. When you select the file to import, you can keep the same **Name** or provide a new one. The Runbook Type field will display the type of runbook after it assesses the file selected and if you attempt to select a different type that is not correct, a message will be presented noting there are potential conflicts and during conversion, there could be syntax errors.

The screenshot shows the 'Import Runbook' dialog box. At the top, it says 'Import Runbook' and has a close button. Below that, it says 'Select a file smaller than 1 MB to import.' A file input field contains the path 'StartAzureRMVM.graphrunbook' with a blue folder icon to its right. The next section is labeled 'Runbook type' with a dropdown menu showing 'Graphical'. The 'Name' field is filled with 'StartAzureRMVM' and has a green checkmark icon to its right. There is also a 'Description' field which is currently empty. At the bottom is a large blue 'Create' button.

Testing a graphical runbook

You can test the Draft version of a runbook in the Azure portal while leaving the published version of the runbook unchanged, or you can test a new runbook before it has been published. This allows you to verify that the runbook is working correctly before replacing the published version. When you test a runbook, the Draft runbook is executed and any actions that it performs are completed. No job history is created, but output is displayed in the Test Output Pane.

Open the Test control for a runbook by opening the runbook for edit and then click on the **Test pane** button.

The Test control prompts for any input parameters, and you can start the runbook by clicking on the **Start** button.

Publishing a graphical runbook

Each runbook in Azure Automation has a Draft and a Published version. Only the Published version is available to be run, and only the Draft version can be edited. The Published version is unaffected by any changes to the Draft version. When the Draft version is ready to be available, then you publish it, which overwrites the Published version with the Draft version.

You can publish a graphical runbook by opening the runbook for editing and then clicking on the **Publish** button.

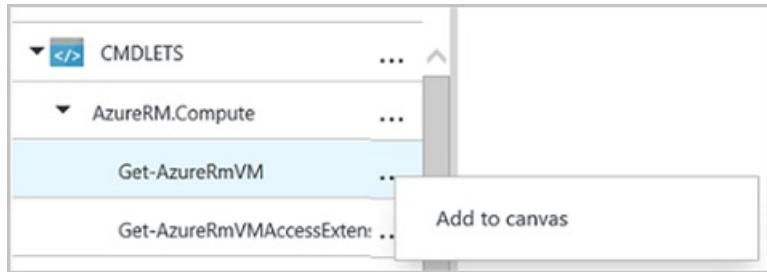
When a runbook has not yet been published, it has a status of **New**. When it is published, it has a status of **Published**. If you edit the runbook after it has been published, and the Draft and Published versions are different, the runbook has a status of **In edit**.

NAME	AUTHORING STATUS	LAST MODIFIED	TAGS
>Hello-World	✓ Published	12/15/2017 4:08 PM	
MyFirstRunbook-Graphical	✍ In edit	12/6/2017 8:02 AM	
MyFirstRunbook-PowerShell	✓ Published	1/24/2018 12:04 PM	
MyFirstRunbook-Python	☀ New	1/24/2018 12:32 PM	

You also have the option to revert to the Published version of a runbook. This throws away any changes made since the runbook was last published and replaces the Draft version of the runbook with the Published version.

Activities

Activities are the building blocks of a runbook. An activity can be a PowerShell cmdlet, a child runbook, or a workflow activity. You add an activity to the runbook by right-clicking it in the Library control and selecting **Add to canvas**. You can then click and drag the activity to place it anywhere on the canvas that you like. The location of the activity on the canvas does not affect the operation of the runbook in any way. You can lay out your runbook however you find it most suitable to visualize its operation.



Select the activity on the canvas to configure its properties and parameters in the Configuration blade. You can change the **Label** of the activity to something that is descriptive to you. The original cmdlet is still being run, you are simply changing its display name that is used in the graphical editor. The label must be unique within the runbook.

Parameter sets

A parameter set defines the mandatory and optional parameters that accept values for a particular cmdlet. All cmdlets have at least one parameter set, and some have multiple. If a cmdlet has multiple parameter sets, then you must select which one you use before you can configure parameters. The parameters that you can configure depends on the parameter set that you choose. You can change the parameter set used by an activity by selecting **Parameter Set** and selecting another set. In this case, any parameter values that you configured are lost.

In the following example, the Get-AzureRmVM cmdlet has three parameter sets. You cannot configure parameter values until you select one of the parameter sets. The **ListVirtualMachineInResourceGroupParamSet** parameter set is for returning all virtual machines in a resource group and has a single optional parameter. The **GetVirtualMachineInResourceGroupParamSet** is for specifying the virtual machine you want to return and has two mandatory and one optional parameter.

A screenshot of the Configuration blade for the 'Get-AzureRmVM' cmdlet. On the left, there are sections for 'Name' (Get-AzureRmVM), 'Label' (* Label) containing 'Get all VMs in Sub', 'Comment' (Get all the Azure RM VMs in the subscription.), 'Parameters' (Configure parameters), 'Optional additional parameters' (Configure parameters), and 'Retry behavior' (Configure retry behavior). In the center, the 'Parameter sets' section is open, showing a dropdown menu with three options: 'GetVirtualMachineInResourceGroupParamSet', 'ListNextLinkVirtualMachinesParamSet', and 'ListVirtualMachineInResourceGroupParamSet'. The 'GetVirtualMachineInResourceGroupParamSet' option is currently selected. On the right, detailed descriptions of these parameter sets are shown.

Parameter values

When you specify a value for a parameter, you select a data source to determine how the value is specified. The data sources that are available for a particular parameter depends on the valid values for that parameter. For example, Null is not an available option for a parameter that does not allow null values.

DATA SOURCE	DESCRIPTION
-------------	-------------

DATA SOURCE	DESCRIPTION
Constant Value	Type in a value for the parameter. This is only available for the following data types: Int32, Int64, String, Boolean, DateTime, Switch.
Activity Output	Output from an activity that precedes the current activity in the workflow. All valid activities are listed. Select just the activity to use its output for the parameter value. If the activity outputs an object with multiple properties, then you can type in the name of the property after selecting the activity.
Runbook Input	Select a runbook input parameter as input to the activity parameter.
Variable Asset	Select an Automation Variable as input.
Credential Asset	Select an Automation Credential as input.
Certificate Asset	Select an Automation Certificate as input.
Connection Asset	Select an Automation Connection as input.
PowerShell Expression	Specify simple PowerShell expression . The expression is evaluated before the activity and the result used for the parameter value. You can use variables to refer to the output of an activity or a runbook input parameter.
Not configured	Clears any value that was previously configured.

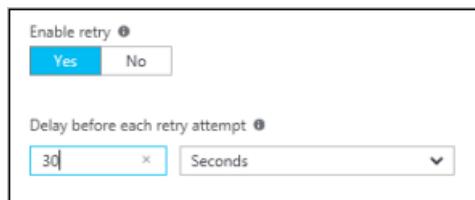
Optional additional parameters

All cmdlets have the option to provide additional parameters. These are PowerShell common parameters or other custom parameters. You are presented with a text box where you can provide parameters using PowerShell syntax. For example, to use the **Verbose** common parameter, you would specify "**-Verbose:\$True**".

Retry activity

Retry Behavior allows an activity to be run multiple times until a particular condition is met, much like a loop. You can use this feature for activities that should run multiple times, are error prone, and may need more than one attempt for success, or test the output information of the activity for valid data.

When you enable retry for an activity, you can set a delay and a condition. The delay is the time (measured in seconds or minutes) that the runbook waits before it runs the activity again. If no delay is specified, then the activity will run again immediately after it completes.



The retry condition is a PowerShell expression that is evaluated after each time the activity runs. If the expression resolves to True, then the activity runs again. If the expression resolves to False, then the activity does not run again, and the runbook moves on to the next activity.

Retry until this condition is true ⓘ

```
$RetryData.NumberOfAttempts -ge 10
```

Examples that can be used in retry condition: ⓘ

\$RetryData.NumberOfAttempts -ge 10 # retry 10 times	
\$RetryData.Output.Count -ge 1 # retry until output is produced	
\$RetryData.TotalDuration.TotalMinutes -ge 2 # retry for maximum of 2 minutes	

The retry condition can use a variable called \$RetryData that provides access to information about the activity retries. This variable has the properties in the following table:

PROPERTY	DESCRIPTION
NumberOfAttempts	Number of times that the activity has been run.
Output	Output from the last run of the activity.
TotalDuration	Timed elapsed since the activity was started the first time.
StartedAt	Time in UTC format the activity was first started.

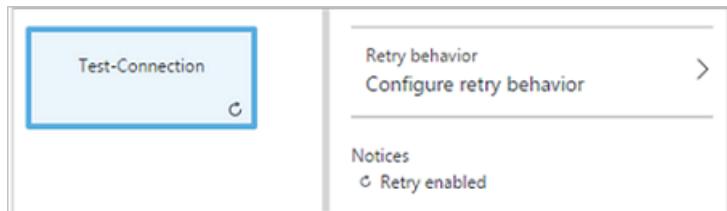
Following are examples of activity retry conditions.

```
# Run the activity exactly 10 times.
$RetryData.NumberOfAttempts -ge 10
```

```
# Run the activity repeatedly until it produces any output.
$RetryData.Output.Count -ge 1
```

```
# Run the activity repeatedly until 2 minutes has elapsed.
$RetryData.TotalDuration.TotalMinutes -ge 2
```

After you configure a retry condition for an activity, the activity includes two visual cues to remind you. One is presented in the activity and the other is when you review the configuration of the activity.



Workflow Script control

A Code control is a special activity that accepts PowerShell or PowerShell Workflow script depending on the type of graphical runbook being authored in order to provide functionality that may otherwise not be available. It cannot accept parameters, but it can use variables for activity output and runbook input parameters. Any output of the activity is added to the databus unless it has no outgoing link in which case it is added to the

output of the runbook.

For example, the following code performs date calculations using a runbook input variable called \$NumberOfDays. It then sends a calculated date time as output to be used by subsequent activities in the runbook.

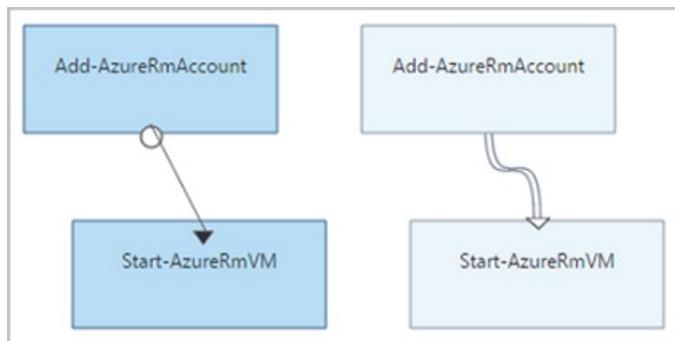
```
$DateTimeNow = (Get-Date).ToUniversalTime()  
$DateTimeStart = ($DateTimeNow).AddDays(-$NumberOfDays)  
$DateTimeStart
```

Links and workflow

A **link** in a graphical runbook connects two activities. It is displayed on the canvas as an arrow pointing from the source activity to the destination activity. The activities run in the direction of the arrow with the destination activity starting after the source activity completes.

Create a link

Create a link between two activities by selecting the source activity and clicking the circle at the bottom of the shape. Drag the arrow to the destination activity and release.



Select the link to configure its properties in the Configuration blade. This includes the link type, which is described in the following table:

LINK TYPE	DESCRIPTION
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. It receives an array of objects from the source activity. Output from the source activity is available as an array of objects.

Starting activity

A graphical runbook starts with any activities that do not have an incoming link. This is often only one activity, which would act as the starting activity for the runbook. If multiple activities do not have an incoming link, then the runbook starts by running them in parallel. It follows the links to run other activities as each completes.

Conditions

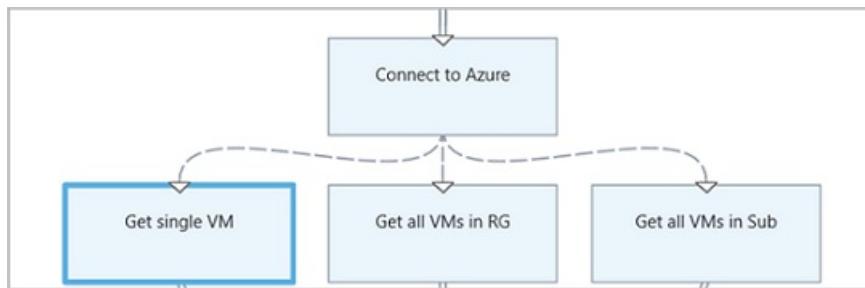
When you specify a condition on a link, the destination activity is only run if the condition resolves to true. You typically use an \$ActivityOutput variable in a condition to retrieve the output from the source activity.

For a pipeline link, you specify a condition for a single object, and the condition is evaluated for each object output by the source activity. The destination activity is then run for each object that satisfies the condition. For

example, with a source activity of Get-AzureRmVm, the following syntax could be used for a conditional pipeline link to retrieve only virtual machines in the resource group named *Group1*.

```
$ActivityOutput['Get Azure VMs'].Name -match "Group1"
```

For a sequence link, the condition is only evaluated once since a single array is returned containing all objects output from the source activity. Because of this, a sequence link cannot be used for filtering like a pipeline link but will simply determine whether or not the next activity is run. Take for example the following set of activities in our Start VM runbook.

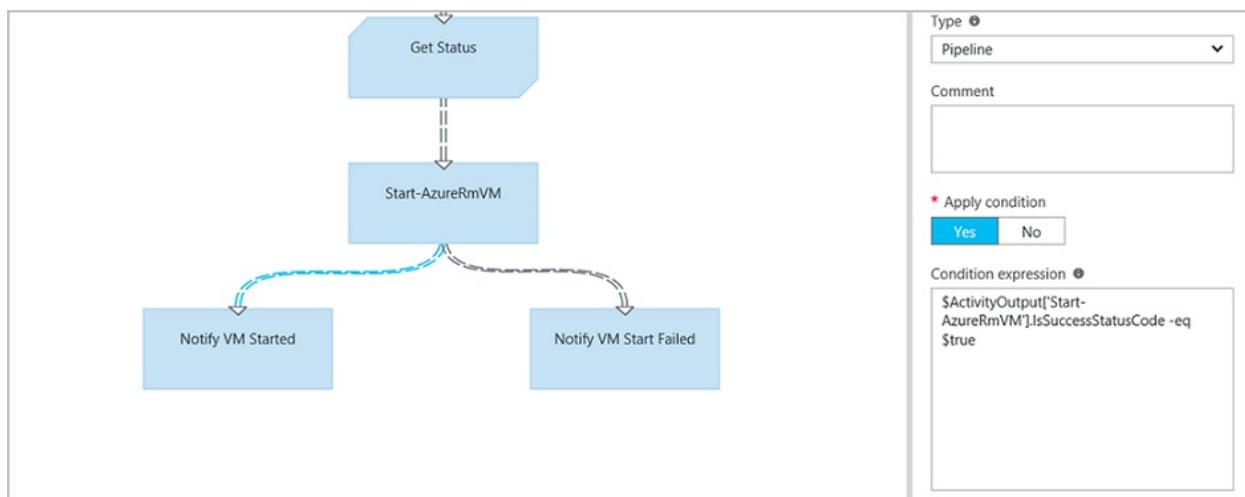


There are three different sequence links that are verifying values were provided to two runbook input parameters representing VM name and Resource Group name in order to determine which is the appropriate action to take - start a single VM, start all VMs in the resource group, or all VMs in a subscription. For the sequence link between Connect to Azure and Get single VM, here is the condition logic:

```
<#
Both VMName and ResourceGroupName runbook input parameters have values
#>
(
    (($VMName -ne $null) -and ($VMName.Length -gt 0))
) -and (
    (($ResourceGroupName -ne $null) -and ($ResourceGroupName.Length -gt 0))
)
```

When you use a conditional link, the data available from the source activity to other activities in that branch is filtered by the condition. If an activity is the source to multiple links, then the data available to activities in each branch depend on the condition in the link connecting to that branch.

For example, the **Start-AzureRmVm** activity in the runbook below starts all virtual machines. It has two conditional links. The first conditional link uses the expression `$ActivityOutput['Start-AzureRmVM'].IsSuccessStatusCode -eq $true` to filter if the Start-AzureRmVm activity completed successfully. The second uses the expression `$ActivityOutput['Start-AzureRmVM'].IsSuccessStatusCode -ne $true` to filter if the Start-AzureRmVm activity failed to start the virtual machine.



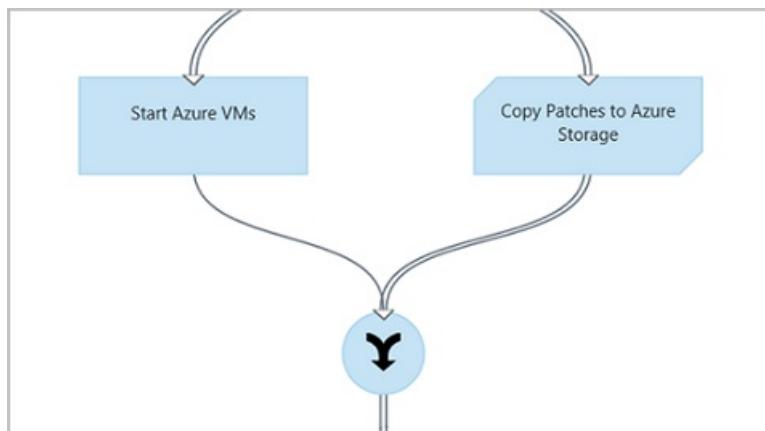
Any activity that follows the first link and uses the activity output from Get-AzureVM will only get the virtual machines that were started at the time that Get-AzureVM was run. Any activity that follows the second link only gets the virtual machines that were stopped at the time that Get-AzureVM was run. Any activity following the third link gets all virtual machines regardless of their running state.

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.

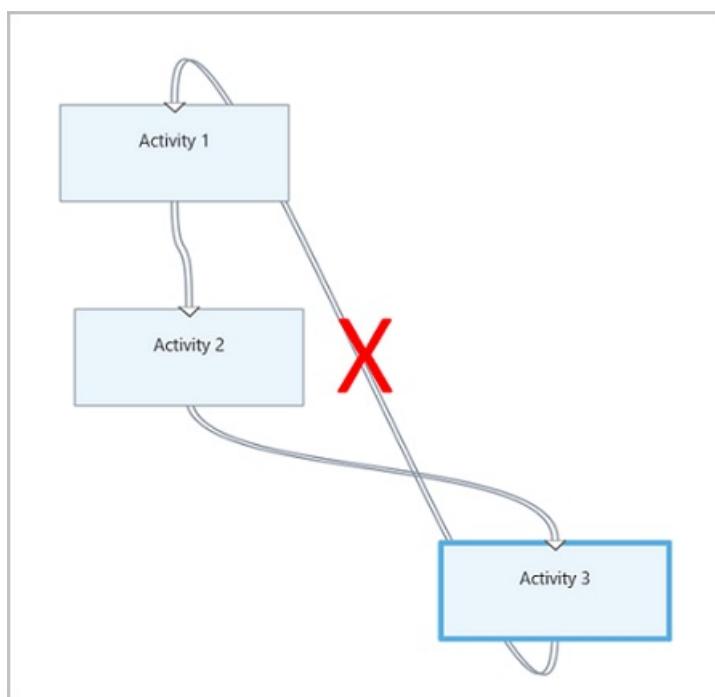
While a junction can have an unlimited number of incoming links, not more than one of those links can be a pipeline. The number of incoming sequence links is not constrained. You are allowed to create the junction with multiple incoming pipeline links and save the runbook, but it fails when it is run.

The example below is part of a runbook that starts a set of virtual machines while simultaneously downloading patches to be applied to those machines. A junction is used to ensure that both processes are completed before the runbook continues.



Cycles

A cycle is when a destination activity links back to its source activity or to another activity that eventually links back to its source. Cycles are currently not allowed in graphical authoring. If your runbook has a cycle, it saves properly but receives an error when it runs.



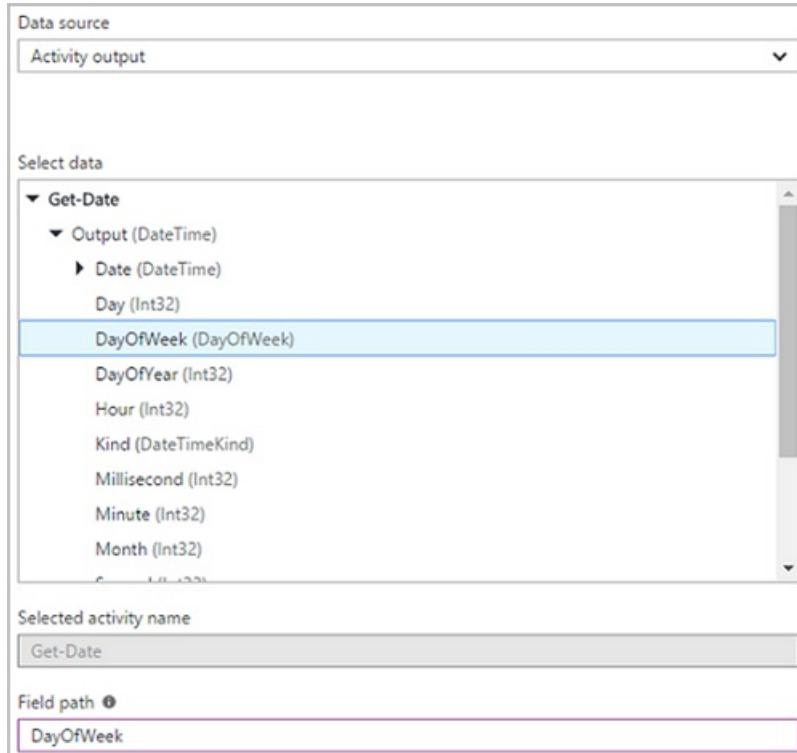
Sharing data between activities

Any data that is output by an activity with an outgoing link is written to the *databus* for the runbook. Any

activity in the runbook can use data on the databus to populate parameter values or include in script code. An activity can access the output of any previous activity in the workflow.

How the data is written to the databus depends on the type of link on the activity. For a **pipeline**, the data is output as multiples objects. For a **sequence** link, the data is output as an array. If there is only one value, it is output as a single element array.

You can access data on the databus using one of two methods. First is using an **Activity Output** data source to populate a parameter of another activity. If the output is an object, you can specify a single property.



You can also retrieve the output of an activity in a **PowerShell Expression** data source or from a **Workflow Script** activity with an **ActivityOutput** variable. If the output is an object, you can specify a single property. **ActivityOutput** variables use the following syntax.

```
$ActivityOutput['Activity Label']
$ActivityOutput['Activity Label'].PropertyName
```

Checkpoints

You can set **checkpoints** in a Graphical PowerShell Workflow runbook by selecting *Checkpoint runbook* on any activity. This causes a checkpoint to be set after the activity runs.

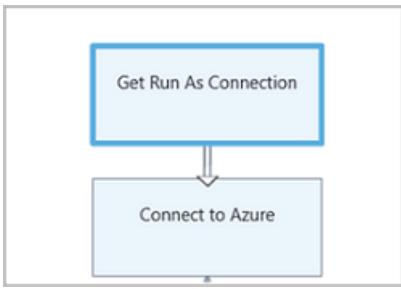


Checkpoints are only enabled in Graphical PowerShell Workflow runbooks, it is not available in Graphical runbooks. If the runbook uses Azure cmdlets, you should follow any checkpointed activity with an `Connect-AzureRmAccount` in case the runbook is suspended and restarts from this checkpoint on a different worker.

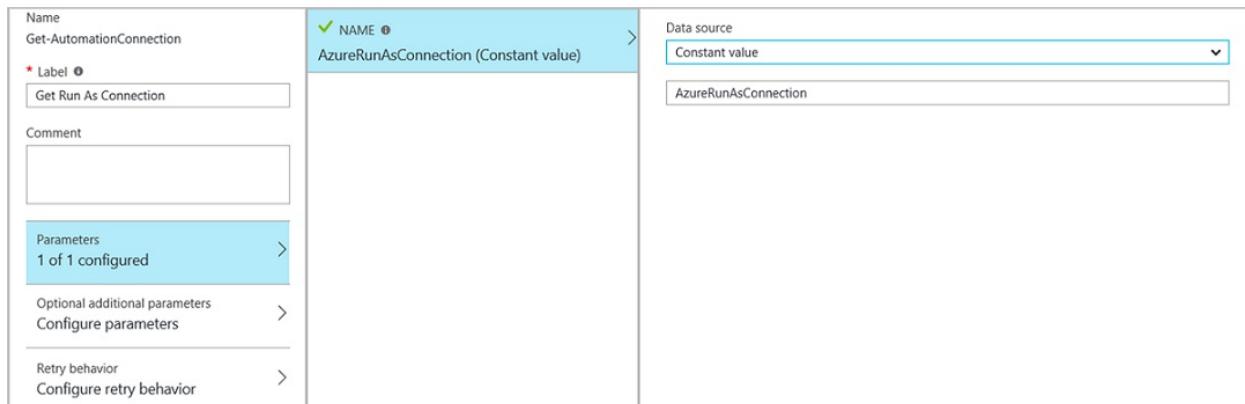
Authenticating to Azure resources

Runbooks in Azure Automation that manage Azure resources require authentication to Azure. The [Run As account](#) (also referred to as a service principal) is the default method to access Azure Resource Manager resources in your subscription with Automation runbooks. You can add this functionality to a graphical runbook

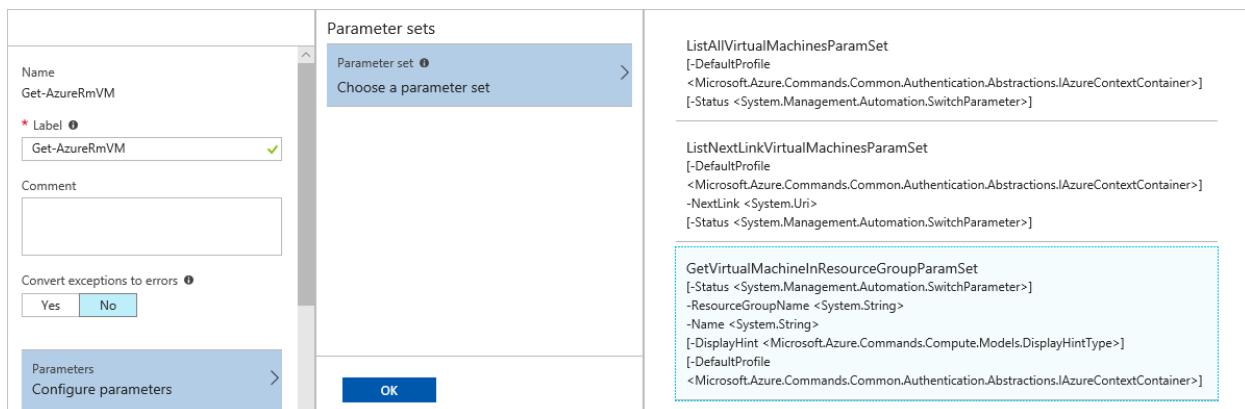
by adding the **AzureRunAsConnection** Connection asset, which is using the PowerShell **Get-AutomationConnection** cmdlet, and **Connect-AzureRmAccount** cmdlet to the canvas. This is illustrated in the following example:



The Get Run As Connection activity (that is, `Get-AutomationConnection`), is configured with a constant value data source named `AzureRunAsConnection`.



The next activity, `Connect-AzureRmAccount`, adds the authenticated Run As account for use in the runbook.



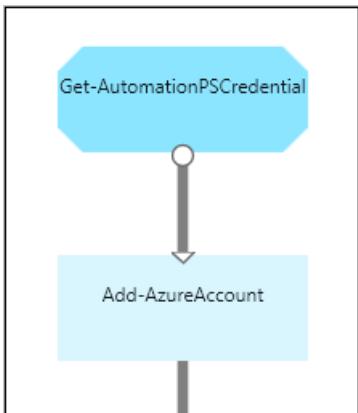
IMPORTANT

Add-AzureRmAccount is now an alias for **Connect-AzureRMAccount**. When searching your library items, if you do not see **Connect-AzureRMAccount**, you can use **Add-AzureRmAccount**, or you can update your modules in your Automation Account.

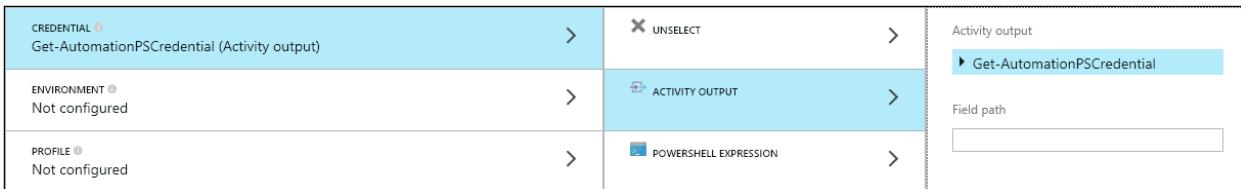
For the parameters **APPLICATIONID**, **CERTIFICATEHUMPRINT**, and **TENANTID** you need to specify the name of the property for the Field path because the activity outputs an object with multiple properties. Otherwise when you execute the runbook, it fails attempting to authenticate. This is what you need at a minimum to authenticate your runbook with the Run As account.

To maintain backwards compatibility for subscribers who have created an Automation account using an [Azure AD User account](#) to manage Azure classic deployment or for Azure Resource Manager resources, the method to authenticate is the `Add-AzureAccount` cmdlet with a [credential asset](#) that represents an Active Directory user with access to the Azure account.

You can add this functionality to a graphical runbook by adding a credential asset to the canvas followed by an Add-AzureAccount activity. Add-AzureAccount uses the credential activity for its input. This is illustrated in the following example:



You have to authenticate at the start of the runbook and after each checkpoint. This means adding an addition Add-AzureAccount activity after any Checkpoint-Workflow activity. You do not need an addition credential activity since you can use the same



Runbook input and output

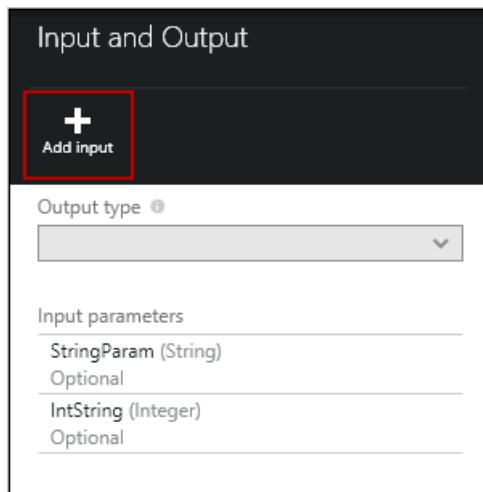
Runbook input

A runbook may require input either from a user when they start the runbook through the Azure portal or from another runbook if the current one is used as a child. For example, if you have a runbook that creates a virtual machine, you may need to provide information such as the name of the virtual machine and other properties each time you start the runbook.

You accept input for a runbook by defining one or more input parameters. You provide values for these parameters each time the runbook is started. When you start a runbook with the Azure portal, it prompts you to provide values for each of the runbook's input parameters.

You can access input parameters for a runbook by clicking the **Input and output** button on the runbook toolbar.

This opens the **Input and Output** control where you can edit an existing input parameter or create a new one by clicking **Add input**.



Each input parameter is defined by the properties in the following table:

PROPERTY	DESCRIPTION
Name	The unique name of the parameter. This can only contain alpha numeric characters and cannot contain a space.
Description	An optional description for the input parameter.
Type	Data type expected for the parameter value. The Azure portal provides an appropriate control for the data type for each parameter when prompting for input.
Mandatory	Specifies whether a value must be provided for the parameter. The runbook cannot be started if you do not provide a value for each mandatory parameter that does not have a default value defined.
Default Value	Specifies what value is used for the parameter if one is not provided. This can either be Null or a specific value.

Runbook output

Data created by any activity that does not have an outgoing link is saved to the [output of the runbook](#). The output is saved with the runbook job and is available to a parent runbook when the runbook is used as a child.

PowerShell expressions

One of the advantages of graphical authoring is providing you with the ability to build a runbook with minimal knowledge of PowerShell. Currently, you do need to know a bit of PowerShell though for populating certain [parameter values](#) and for setting [link conditions](#). This section provides a quick introduction to PowerShell expressions for those users who may not be familiar with it. Full details of PowerShell are available at [Scripting with Windows PowerShell](#).

PowerShell expression data source

You can use a PowerShell expression as a data source to populate the value of an [activity parameter](#) with the results of some PowerShell code. This could be a single line of code that performs some simple function or multiple lines that perform some complex logic. Any output from a command that is not assigned to a variable is output to the parameter value.

For example, the following command would output the current date.

For example, the following command would output the current date.

```
Get-Date
```

The following commands build a string from the current date and assign it to a variable. The contents of the variable are then sent to the output

```
$string = "The current date is " + (Get-Date)  
$string
```

The following commands evaluate the current date and return a string indicating whether the current day is a weekend or weekday.

```
$date = Get-Date  
if (($date.DayOfWeek = "Saturday") -or ($date.DayOfWeek = "Sunday")) { "Weekend" }  
else { "Weekday" }
```

Activity output

To use the output from a previous activity in the runbook, use the `$ActivityOutput` variable with the following syntax.

```
$ActivityOutput['Activity Label'].PropertyName
```

For example, you may have an activity with a property that requires the name of a virtual machine in which case you could use the following expression:

```
$ActivityOutput['Get-AzureVm'].Name
```

If the property that required the virtual machine object instead of just a property, then you would return the entire object using the following syntax.

```
$ActivityOutput['Get-AzureVm']
```

You can also use the output of an activity in a more complex expression such as the following that concatenates text to the virtual machine name.

```
"The computer name is " + $ActivityOutput['Get-AzureVm'].Name
```

Conditions

Use [comparison operators](#) to compare values or determine if a value matches a specified pattern. A comparison returns a value of either `$true` or `$false`.

For example, the following condition determines whether the virtual machine from an activity named `Get-AzureVM` is currently *stopped*.

```
$ActivityOutput["Get-AzureVM"].PowerState -eq "Stopped"
```

The following condition checks whether the same virtual machine is in any state other than *stopped*.

```
$ActivityOutput["Get-AzureVM"].PowerState -ne "Stopped"
```

You can join multiple conditions using a [logical operator](#) such as **-and** or **-or**. For example, the following condition checks whether the same virtual machine in the previous example is in a state of *stopped* or *stopping*.

```
($ActivityOutput["Get-AzureVM"].PowerState -eq "Stopped") -or ($ActivityOutput["Get-AzureVM"].PowerState -eq "Stopping")
```

Hashtables

[Hashtables](#) are name/value pairs that are useful for returning a set of values. Properties for certain activities may expect a hashtable instead of a simple value. You may also see a hashtable referred to as a dictionary.

You create a hashtable with the following syntax. A hashtable can contain any number of entries but each is defined by a name and value.

```
@{ <name> = <value>; [<name> = <value> ] ...}
```

For example, the following expression creates a hashtable to be used in the data source for an activity parameter that expected a hashtable with values for an internet search.

```
$query = "Azure Automation"
$count = 10
$h = @{ 'q'=$query; 'lr'='lang_ja'; 'count'=$Count}
$h
```

The following example uses output from an activity called *Get Twitter Connection* to populate a hashtable.

```
@{ 'ApiKey'=$ActivityOutput['Get Twitter Connection'].ConsumerAPIKey;
    'ApiSecret'=$ActivityOutput['Get Twitter Connection'].ConsumerAPISecret;
    'AccessToken'=$ActivityOutput['Get Twitter Connection'].AccessToken;
    'AccessTokenSecret'=$ActivityOutput['Get Twitter Connection'].AccessTokenSecret}
```

Next Steps

- To get started with PowerShell workflow runbooks, see [My first PowerShell workflow runbook](#)
- To get started with Graphical runbooks, see [My first graphical runbook](#)
- To know more about runbook types, their advantages and limitations, see [Azure Automation runbook types](#)
- To understand how to authenticate using the Automation Run As account, see [Configure Azure Run As Account](#)

Testing a runbook in Azure Automation

5/21/2018 • 2 minutes to read • [Edit Online](#)

When you test a runbook, the [Draft version](#) is executed and any actions that it performs are completed. No job history is created, but the [Output](#) and [Warning and Error](#) streams are displayed in the Test output Pane. Messages to the [Verbose Stream](#) are displayed in the Output Pane only if the [\\$VerbosePreference variable](#) is set to Continue.

Even though the draft version is being run, the runbook still executes the workflow normally and performs any actions against resources in the environment. For this reason, you should only test runbooks at non-production resources.

The procedure to test each [type of runbook](#) is the same, and there is no difference in testing between the textual editor and the graphical editor in the Azure portal.

To test a runbook in the Azure portal

You can work with any [runbook type](#) in the Azure portal.

1. Open the Draft version of the runbook in either the [textual editor](#) or [graphical editor](#).
2. Click the **Test** button to open the Test blade.
3. If the runbook has parameters, they will be listed in the left pane where you can provide values to be used for the test.
4. If you want to run the test on a [Hybrid Runbook Worker](#), then change **Run Settings** to **Hybrid Worker** and select the name of the target group. Otherwise, keep the default **Azure** to run the test in the cloud.
5. Click the **Start** button to start the test.
6. If the runbook is [PowerShell Workflow](#) or [Graphical](#), then you can stop or suspend it while it is being tested with the buttons underneath the Output Pane. When you suspend the runbook, it completes the current activity before being suspended. Once the runbook is suspended, you can stop it or restart it.
7. Inspect the output from the runbook in the output pane.

Next Steps

- To learn how to create or import a runbook, see [Creating or importing a runbook in Azure Automation](#)
- To learn more about Graphical Authoring, see [Graphical authoring in Azure Automation](#)
- To get started with PowerShell workflow runbooks, see [My first PowerShell workflow runbook](#)
- To learn more about configuring runbooks to return status messages and errors, including recommended practices, see [Runbook output and messages in Azure Automation](#)

Learning key Windows PowerShell Workflow concepts for Automation runbooks

7/3/2018 • 9 minutes to read • [Edit Online](#)

Runbooks in Azure Automation are implemented as Windows PowerShell Workflows. A Windows PowerShell Workflow is similar to a Windows PowerShell script but has some significant differences that can be confusing to a new user. While this article is intended to help you write runbooks using PowerShell workflow, we recommend you write runbooks using PowerShell unless you need checkpoints. There are several syntax differences when authoring PowerShell Workflow runbooks and these differences require a bit more work to write effective workflows.

A workflow is a sequence of programmed, connected steps that perform long-running tasks or require the coordination of multiple steps across multiple devices or managed nodes. The benefits of a workflow over a normal script include the ability to simultaneously perform an action against multiple devices and the ability to automatically recover from failures. A Windows PowerShell Workflow is a Windows PowerShell script that uses Windows Workflow Foundation. While the workflow is written with Windows PowerShell syntax and launched by Windows PowerShell, it is processed by Windows Workflow Foundation.

For complete details on the topics in this article, see [Getting Started with Windows PowerShell Workflow](#).

Basic structure of a workflow

The first step to converting a PowerShell script to a PowerShell workflow is enclosing it with the **Workflow** keyword. A workflow starts with the **Workflow** keyword followed by the body of the script enclosed in braces. The name of the workflow follows the **Workflow** keyword as shown in the following syntax:

```
Workflow Test-Workflow
{
    <Commands>
}
```

The name of the workflow must match the name of the Automation runbook. If the runbook is being imported, then the filename must match the workflow name and must end in *.ps1*.

To add parameters to the workflow, use the **Param** keyword just as you would to a script.

Code changes

PowerShell workflow code looks almost identical to PowerShell script code except for a few significant changes. The following sections describe changes that you need to make to a PowerShell script for it to run in a workflow.

Activities

An activity is a specific task in a workflow. Just as a script is composed of one or more commands, a workflow is composed of one or more activities that are carried out in a sequence. Windows PowerShell Workflow automatically converts many of the Windows PowerShell cmdlets to activities when it runs a workflow. When you specify one of these cmdlets in your runbook, the corresponding activity is run by Windows Workflow Foundation. For those cmdlets without a corresponding activity, Windows PowerShell Workflow automatically runs the cmdlet within an **InlineScript** activity. There is a set of cmdlets that are excluded and cannot be used in a workflow unless you explicitly include them in an **InlineScript** block. For further details on these concepts, see [Using Activities in Script Workflows](#).

Workflow activities share a set of common parameters to configure their operation. For details about the workflow common parameters, see [about_WorkflowCommonParameters](#).

Positional parameters

You can't use positional parameters with activities and cmdlets in a workflow. All this means is that you must use parameter names.

For example, consider the following code that gets all running services.

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

If you try to run this same code in a workflow, you receive a message like "Parameter set cannot be resolved using the specified named parameters." To correct this, provide the parameter name as in the following.

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

Deserialized objects

Objects in workflows are deserialized. This means that their properties are still available, but not their methods.

For example, consider the following PowerShell code that stops a service using the Stop method of the Service object.

```
$Service = Get-Service -Name MyService
$Service.Stop()
```

If you try to run this in a workflow, you receive an error saying "Method invocation is not supported in a Windows PowerShell Workflow."

One option is to wrap these two lines of code in an [InlineScript](#) block in which case \$Service would be a service object within the block.

```
Workflow Stop-Service
{
    InlineScript {
        $Service = Get-Service -Name MyService
        $Service.Stop()
    }
}
```

Another option is to use another cmdlet that performs the same functionality as the method, if one is available. In our sample, the Stop-Service cmdlet provides the same functionality as the Stop method, and you could use the following for a workflow.

```
Workflow Stop-MyService
{
    $Service = Get-Service -Name MyService
    Stop-Service -Name $Service.Name
}
```

InlineScript

The **InlineScript** activity is useful when you need to run one or more commands as traditional PowerShell script

instead of PowerShell workflow. While commands in a workflow are sent to Windows Workflow Foundation for processing, commands in an **InlineScript** block are processed by Windows PowerShell.

InlineScript uses the following syntax shown below.

```
InlineScript
{
    <Script Block>
} <Common Parameters>
```

You can return output from an **InlineScript** by assigning the output to a variable. The following example stops a service and then outputs the service name.

```
Workflow Stop-MyService
{
    $Output = InlineScript {
        $Service = Get-Service -Name MyService
        $Service.Stop()
        $Service
    }

    $Output.Name
}
```

You can pass values into an **InlineScript** block, but you must use **\$Using** scope modifier. The following example is identical to the previous example except that the service name is provided by a variable.

```
Workflow Stop-MyService
{
    $serviceName = "MyService"

    $Output = InlineScript {
        $Service = Get-Service -Name $Using:serviceName
        $Service.Stop()
        $Service
    }

    $Output.Name
}
```

While **InlineScript** activities may be critical in certain workflows, they do not support workflow constructs and should only be used when necessary for the following reasons:

- You cannot use **checkpoints** inside an **InlineScript** block. If a failure occurs within the block, it must be resumed from the beginning of the block.
- You cannot use **parallel execution** inside an **InlineScriptBlock**.
- **InlineScript** affects scalability of the workflow since it holds the Windows PowerShell session for the entire length of the **InlineScript** block.

For more information on using **InlineScript**, see [Running Windows PowerShell Commands in a Workflow](#) and [about_InlineScript](#).

Parallel processing

One advantage of Windows PowerShell Workflows is the ability to perform a set of commands in parallel instead of sequentially as with a typical script.

You can use the **Parallel** keyword to create a script block with multiple commands that run concurrently. This uses

the following syntax shown below. In this case, Activity1 and Activity2 starts at the same time. Activity3 starts only after both Activity1 and Activity2 have completed.

```
Parallel
{
    <Activity1>
    <Activity2>
}
<Activity3>
```

For example, consider the following PowerShell commands that copy multiple files to a network destination. These commands are run sequentially so that one file must finish copying before the next is started.

```
Copy-Item -Path C:\LocalPath\file1.txt -Destination \\NetworkPath\file1.txt
Copy-Item -Path C:\LocalPath\file2.txt -Destination \\NetworkPath\file2.txt
Copy-Item -Path C:\LocalPath\file3.txt -Destination \\NetworkPath\file3.txt
```

The following workflow runs these same commands in parallel so that they all start copying at the same time. Only after they are all copied is the completion message displayed.

```
Workflow Copy-Files
{
    Parallel
    {
        Copy-Item -Path "C:\LocalPath\file1.txt" -Destination "\\NetworkPath"
        Copy-Item -Path "C:\LocalPath\file2.txt" -Destination "\\NetworkPath"
        Copy-Item -Path "C:\LocalPath\file3.txt" -Destination "\\NetworkPath"
    }

    Write-Output "Files copied."
}
```

You can use the **ForEach -Parallel** construct to process commands for each item in a collection concurrently. The items in the collection are processed in parallel while the commands in the script block run sequentially. This uses the following syntax shown below. In this case, Activity1 starts at the same time for all items in the collection. For each item, Activity2 starts after Activity1 is complete. Activity3 starts only after both Activity1 and Activity2 have completed for all items.

```
ForEach -Parallel ($<item> in $<collection>)
{
    <Activity1>
    <Activity2>
}
<Activity3>
```

The following example is similar to the previous example copying files in parallel. In this case, a message is displayed for each file after it copies. Only after they are all completely copied is the final completion message displayed.

```

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

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

    Write-Output "All files copied."
}

```

NOTE

We do not recommend running child runbooks in parallel since this has been shown to give unreliable results. The output from the child runbook sometimes does not show up, and settings in one child runbook can affect the other parallel child runbooks. Variables such as \$VerbosePreference, \$WarningPreference, and others may not be propagated to the child runbooks. And if the child runbook changes these values, they may not be properly restored after invocation.

Checkpoints

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.

In the following sample code, an exception occurs after Activity2 causing the workflow to end. When the workflow is run again, it starts by running Activity2 since this was just after the last checkpoint set.

```

<Activity1>
Checkpoint-Workflow
<Activity2>
<Exception>
<Activity3>

```

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. For example, your workflow may create a virtual machine. You could set a checkpoint both before and after the commands to create the virtual machine. If the creation fails, then the commands would be repeated if the workflow is started again. If the workflow fails after the creation succeeds, then the virtual machine will not be created again when the workflow is resumed.

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."
}

```

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. Otherwise, you may receive the following error message: *The workflow job cannot be resumed, either because persistence data could not be saved completely, or saved persistence data has been corrupted. You must restart the workflow.*

The following same code demonstrates how to handle this in your PowerShell Workflow runbooks.

```

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

    $VmsToCreate = Get-AzureAutomationVariable -Name "VmsToCreate"

    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
        $Cred = Get-AzureAutomationCredential -Name "MyCredential"
        $null = Connect-AzureRmAccount -Credential $Cred
    }
}

```

IMPORTANT

Add-AzureRmAccount is now an alias for **Connect-AzureRMAccount**. When searching your library items, if you do not see **Connect-AzureRMAccount**, you can use **Add-AzureRmAccount**, or you can update your modules in your Automation Account.

This is not required if you are authenticating using a Run As account configured with a service principal.

For more information about checkpoints, see [Adding Checkpoints to a Script Workflow](#).

Next steps

- To get started with PowerShell workflow runbooks, see [My first PowerShell workflow runbook](#)

Child runbooks in Azure Automation

6/14/2018 • 5 minutes to read • [Edit Online](#)

It is a best practice in Azure Automation to write reusable, modular runbooks with a discrete function that can be used by other runbooks. A parent runbook will often call one or more child runbooks to perform required functionality. There are two ways to call a child runbook, and each has distinct differences that you should understand so that you can determine which will be best for your different scenarios.

Invoking a child runbook using inline execution

To invoke a runbook inline from another runbook, you use the name of the runbook and provide values for its parameters exactly like you would use an activity or cmdlet. All runbooks in the same Automation account are available to all others to be used in this manner. The parent runbook will wait for the child runbook to complete before moving to the next line, and any output is returned directly to the parent.

When you invoke a runbook inline, it runs in the same job as the parent runbook. There will be no indication in the job history of the child runbook that it ran. Any exceptions and any stream output from the child runbook will be associated with the parent. This results in fewer jobs and makes them easier to track and to troubleshoot since any exceptions thrown by the child runbook and any of its stream output are associated with the parent job.

When a runbook is published, any child runbooks that it calls must already be published. This is because Azure Automation builds an association with any child runbooks when a runbook is compiled. If they aren't, the parent runbook will appear to publish properly, but will generate an exception when it's started. If this happens, you can republish the parent runbook in order to properly reference the child runbooks. You do not need to republish the parent runbook if any of the child runbooks are changed because the association will have already been created.

The parameters of a child runbook called inline can be any data type including complex objects, and there is no [JSON serialization](#) as there is when you start the runbook using the Azure portal or with the `Start-AzureRmAutomationRunbook` cmdlet.

Runbook types

Which types can call each other:

- A [PowerShell runbook](#) and [Graphical runbooks](#) can call each other inline (both are PowerShell based).
- A [PowerShell Workflow runbook](#) and Graphical PowerShell Workflow runbooks can call each other inline (both are PowerShell Workflow based)
- The PowerShell types and the PowerShell Workflow types can't call each other inline, and must use `Start-AzureRmAutomationRunbook`.

When does publish order matter:

- The publish order of runbooks only matters for PowerShell Workflow and Graphical PowerShell Workflow runbooks.

When you call a Graphical or PowerShell Workflow child runbook using inline execution, you just use the name of the runbook. When you call a PowerShell child runbook, you must precede its name with `.\` to specify that the script is located in the local directory.

Example

The following example invokes a test child runbook that accepts three parameters, a complex object, an integer, and a boolean. The output of the child runbook is assigned to a variable. In this case, the child runbook is a PowerShell Workflow runbook.

```
$vm = Get-AzureRmVM -ResourceGroupName "LabRG" -Name "MyVM"
$output = PSWF-ChildRunbook -VM $vm -RepeatCount 2 -Restart $true
```

Following is the same example using a PowerShell runbook as the child.

```
$vm = Get-AzureRmVM -ResourceGroupName "LabRG" -Name "MyVM"
$output = .\PS-ChildRunbook.ps1 -VM $vm -RepeatCount 2 -Restart $true
```

Starting a child runbook using cmdlet

You can use the [Start-AzureRmAutomationRunbook](#) cmdlet to start a runbook as described in [To start a runbook with Windows PowerShell](#). There are two modes of use for this cmdlet. In one mode, the cmdlet returns the job id as soon as the child job is created for the child runbook. In the other mode, which you enable by specifying the **-Wait** parameter, the cmdlet will wait until the child job finishes and will return the output from the child runbook.

The job from a child runbook started with a cmdlet will run in a separate job from the parent runbook. This results in more jobs than invoking the runbook inline and makes them more difficult to track. The parent can start multiple child runbooks asynchronously without waiting for each to complete. For that same kind of parallel execution calling the child runbooks inline, the parent runbook would need to use the [parallel keyword](#).

The output of the child runbooks are not returned to the parent runbook reliably due to timing. Also certain variables like `$VerbosePreference`, `$WarningPreference`, and others may not be propagated to the child runbooks. In order to avoid these issues, you can invoke the child runbooks as separate Automation jobs using the `Start-AzureRmAutomationRunbook` cmdlet with the `-Wait` switch. This blocks the parent runbook until the child runbook is complete.

If you don't want the parent runbook to be blocked on waiting, you can invoke the child runbook using `Start-AzureRmAutomationRunbook` cmdlet without the `-Wait` switch. You would then need to use `Get-AzureRmAutomationJob` to wait for job completion, and `Get-AzureRmAutomationJobOutput` and `Get-AzureRmAutomationJobOutputRecord` to retrieve the results.

Parameters for a child runbook started with a cmdlet are provided as a hashtable as described in [Runbook Parameters](#). Only simple data types can be used. If the runbook has a parameter with a complex data type, then it must be called inline.

Example

The following example starts a child runbook with parameters and then waits for it to complete using the `Start-AzureRmAutomationRunbook -wait` parameter. Once completed, its output is collected from the child runbook.

```
$params = @{"VMName"="MyVM";"RepeatCount"=2;"Restart"=$true}
$joboutput = Start-AzureRmAutomationRunbook -AutomationAccountName "MyAutomationAccount" -Name "Test-ChildRunbook" -ResourceGroupName "LabRG" -Parameters $params -wait
```

Comparison of methods for calling a child runbook

The following table summarizes the differences between the two methods for calling a runbook from another runbook.

	INLINE	CMDLET
Job	Child runbooks run in the same job as the parent.	A separate job is created for the child runbook.

	INLINE	CMDLET
Execution	Parent runbook waits for the child runbook to complete before continuing.	Parent runbook continues immediately after child runbook is started <i>or</i> parent runbook waits for the child job to finish.
Output	Parent runbook can directly get output from child runbook.	Parent runbook must retrieve output from child runbook job <i>or</i> parent runbook can directly get output from child runbook.
Parameters	Values for the child runbook parameters are specified separately and can use any data type.	Values for the child runbook parameters must be combined into a single hashtable and can only include simple, array, and object data types that leverage JSON serialization.
Automation Account	Parent runbook can only use child runbook in the same automation account.	Parent runbook can use child runbook from any automation account from the same Azure subscription and even a different subscription if you have a connection to it.
Publishing	Child runbook must be published before parent runbook is published.	Child runbook must be published any time before parent runbook is started.

Next steps

- [Starting a runbook in Azure Automation](#)
- [Runbook output and messages in Azure Automation](#)

Runbook output and messages in Azure Automation

6/25/2018 • 12 minutes to read • [Edit Online](#)

Most Azure Automation runbooks have some form of output such as an error message to the user or a complex object intended to be consumed by another workflow. Windows PowerShell provides [multiple streams](#) to send output from a script or workflow. Azure Automation works with each of these streams differently, and you should follow best practices for how to use each when you are creating a runbook.

The following table provides a brief description of each of the streams and their behavior in the Azure portal both when running a published runbook and when [testing a runbook](#). Further details on each stream are provided in subsequent sections.

STREAM	DESCRIPTION	PUBLISHED	TEST
Output	Objects intended to be consumed by other runbooks.	Written to the job history.	Displayed in the Test Output Pane.
Warning	Warning message intended for the user.	Written to the job history.	Displayed in the Test Output Pane.
Error	Error message intended for the user. Unlike an exception, the runbook continues after an error message by default.	Written to the job history.	Displayed in the Test Output Pane.
Verbose	Messages providing general or debugging information.	Written to job history only if verbose logging is turned on for the runbook.	Displayed in the Test Output pane only if \$VerbosePreference is set to Continue in the runbook.
Progress	Records automatically generated before and after each activity in the runbook. The runbook should not attempt to create its own progress records since they are intended for an interactive user.	Written to job history only if progress logging is turned on for the runbook.	Not displayed in the Test Output Pane.
Debug	Messages intended for an interactive user. Should not be used in runbooks.	Not written to job history.	Not written to Test Output Pane.

Output stream

The Output stream is intended for output of objects created by a script or workflow when it runs correctly. In Azure Automation, this stream is primarily used for objects intended to be consumed by [parent runbooks that call the current runbook](#). When you [call a runbook inline](#) from a parent runbook, it returns data from the output stream to the parent. You should only use the output stream to communicate general information back to the user if you know the runbook is never called by another runbook. As a best practice, however, you should typically use the [Verbose Stream](#) to communicate general information to the user.

You can write data to the output stream using [Write-Output](#) or by putting the object on its own line in the runbook.

```
#The following lines both write an object to the output stream.  
Write-Output -InputObject $object  
$object
```

Output from a function

When you write to the output stream in a function that is included in your runbook, the output is passed back to the runbook. If the runbook assigns that output to a variable, then it is not written to the output stream. Writing to any other streams from within the function writes to the corresponding stream for the runbook.

Consider the following sample runbook:

```
Workflow Test-Runbook  
{  
    Write-Verbose "Verbose outside of function" -Verbose  
    Write-Output "Output outside of function"  
    $functionOutput = Test-Function  
    $functionOutput  
  
    Function Test-Function  
    {  
        Write-Verbose "Verbose inside of function" -Verbose  
        Write-Output "Output inside of function"  
    }  
}
```

The output stream for the runbook job would be:

```
Output inside of function  
Output outside of function
```

The verbose stream for the runbook job would be:

```
Verbose outside of function  
Verbose inside of function
```

Once you have published the runbook and before you start it, you must also turn on Verbose logging in the runbook settings in order to get the Verbose stream output.

Declaring output data type

A workflow can specify the data type of its output using the [OutputType attribute](#). This attribute has no effect during runtime, but it provides an indication to the runbook author at design time of the expected output of the runbook. As the toolset for runbooks continues to evolve, the importance of declaring output data types at design time increases in importance. As a result, it is a best practice to include this declaration in any runbooks that you create.

Here is a list of example output types:

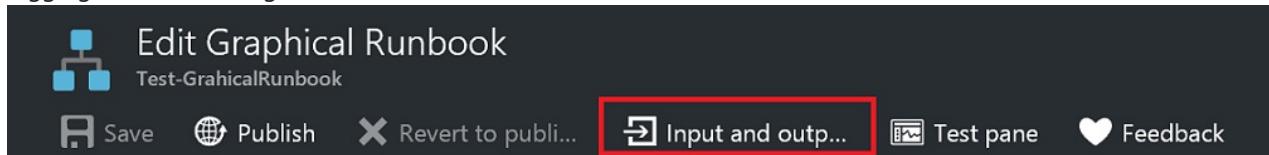
- System.String
- System.Int32
- System.Collections.Hashtable
- Microsoft.Azure.Commands.Compute.Models.PSVirtualMachine

The following sample runbook outputs a string object and includes a declaration of its output type. If your runbook outputs an array of a certain type, then you should still specify the type as opposed to an array of the type.

```
Workflow Test-Runbook
{
    [OutputType([string])]

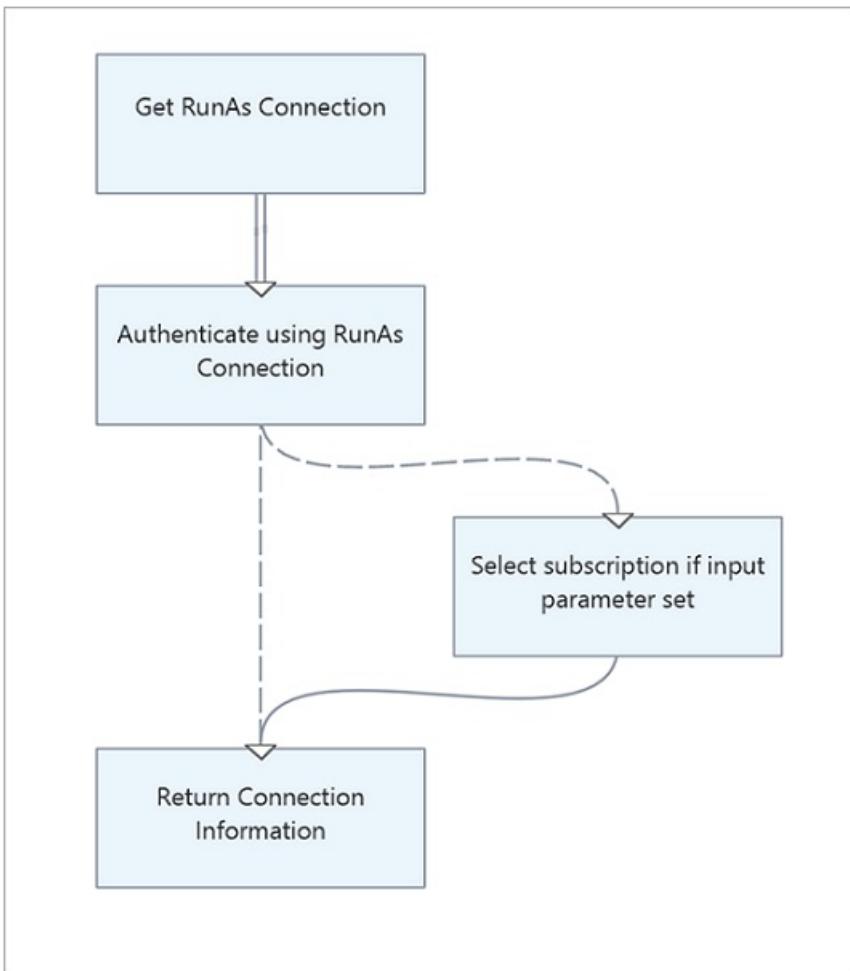
    $output = "This is some string output."
    Write-Output $output
}
```

To declare an output type in Graphical or Graphical PowerShell Workflow runbooks, you can select the **Input and Output** menu option and type in the name of the output type. It is recommended you use the full .NET class name to make it easily identifiable when referencing it from a parent runbook. This exposes all the properties of that class to the data bus in the runbook and provides much flexibility when using them for conditional logic, logging, and referencing as values for other activities in the runbook.

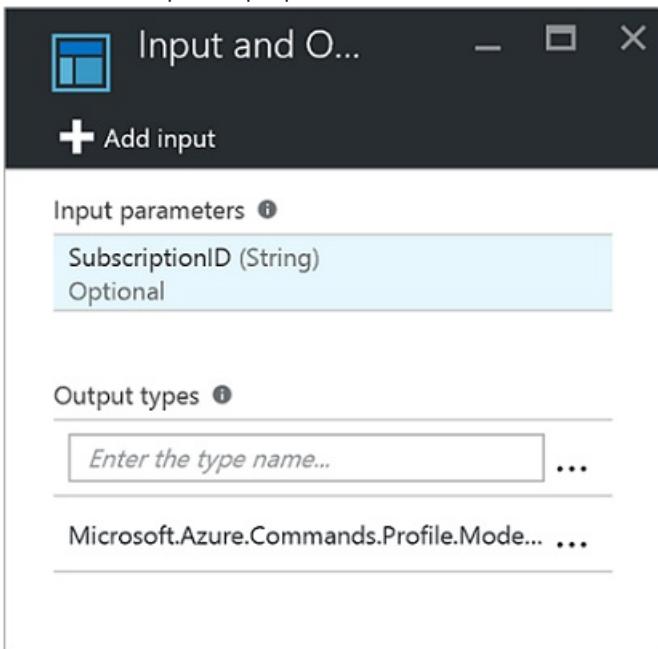


In the following example, you have two graphical runbooks to demonstrate this feature. If you apply the modular runbook design model, you have one runbook, which serves as the *Authentication Runbook template* managing authentication with Azure using the Run As account. Our second runbook, which would normally perform the core logic to automate a given scenario, in this case is going to execute the *Authentication Runbook template* and display the results to your **Test** output pane. Under normal circumstances, you would have this runbook do something against a resource leveraging the output from the child runbook.

Here is the basic logic of the **AuthenticateTo-Azure** runbook.

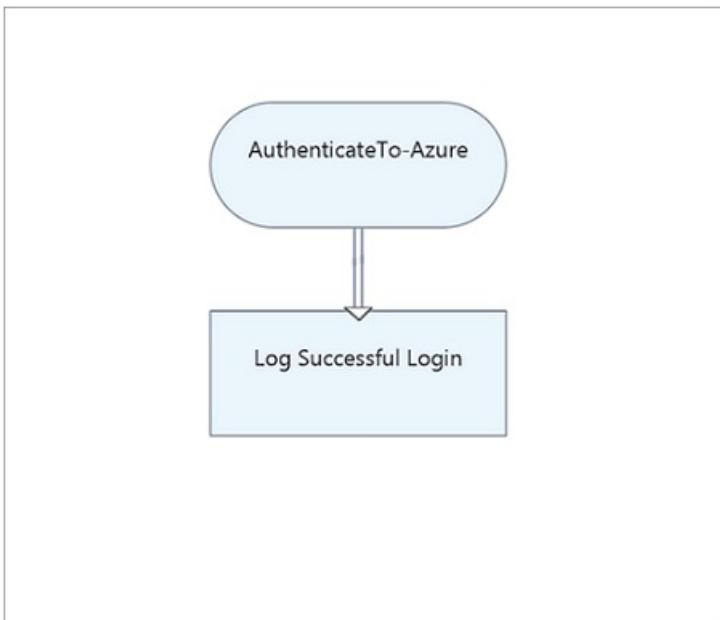


It includes the output type `Microsoft.Azure.Commands.Profile.Models.PSAzureContext`, which returns the authentication profile properties.



While this runbook is straight forward, there is one configuration item to call out here. The last activity is executing the **Write-Output** cmdlet and writes the profile data to a `$_` variable using a PowerShell expression for the **Inputobject** parameter, which is required for that cmdlet.

For the second runbook in this example, named `Test-ChildOutputType`, you simply have two activities.



The first activity calls the **AuthenticateTo-Azure** runbook and the second activity is running the **Write-Verbose** cmdlet with the **Data source of Activity output** and the value for **Field path** is **Context.Subscription.SubscriptionName**, which is specifying the context output from the **AuthenticateTo-Azure** runbook.

Activity Parameter Configuration	Parameter Value
<p>Name: Write-Verbose</p> <p>* Label: Log Successful Login</p> <p>Comment:</p> <p>Parameters: 1 of 1 configured</p> <p>Optional additional parameters: Configure parameters</p> <p>Retry behavior: Configure retry behavior</p>	<p>✓ MESSAGE ⓘ AuthenticateTo-Azure (Activity output) ></p> <p>Data source: Activity output</p> <p>Select data:</p> <ul style="list-style-type: none"> ▼ AuthenticateTo-Azure <ul style="list-style-type: none"> ▼ Output (PSAzureContext) <ul style="list-style-type: none"> ▶ Account (PSAzureRmAccount) ▶ Environment (PSAzureEnvironment) ▶ Subscription (PSAzureSubscription) <ul style="list-style-type: none"> CurrentStorageAccountName (String) SubscriptionId (String) SubscriptionName (String) TenantId (String) ▶ Tenant (PSAzureTenant) <p>Selected activity name: AuthenticateTo-Azure</p> <p>Field path: Context.Subscription.SubscriptionName</p>

The resulting output is the name of the subscription.

One note about the behavior of the Output Type control. When you type a value in the Output Type field on the Input and Output properties blade, you have to click outside of the control after you type it, in order for your entry

to be recognized by the control.

Message streams

Unlike the output stream, message streams are intended to communicate information to the user. There are multiple message streams for different kinds of information, and each is handled differently by Azure Automation.

Warning and error streams

The Warning and Error streams are intended to log problems that occur in a runbook. They are written to the job history when a runbook is executed, and are included in the Test Output Pane in the Azure portal when a runbook is tested. By default, the runbook will continue executing after a warning or error. You can specify that the runbook should be suspended on a warning or error by setting a [preference variable](#) in the runbook before creating the message. For example, to cause a runbook to suspend on an error as it would an exception, set **\$ErrorActionPreference** to Stop.

Create a warning or error message using the [Write-Warning](#) or [Write-Error](#) cmdlet. Activities may also write to these streams.

```
#The following lines create a warning message and then an error message that will suspend the runbook.

$ErrorActionPreference = "Stop"
Write-Warning -Message "This is a warning message."
Write-Error -Message "This is an error message that will stop the runbook because of the preference variable."
```

Verbose stream

The Verbose message stream is for general information about the runbook operation. Since the [Debug Stream](#) is not available in a runbook, verbose messages should be used for debug information. By default, verbose messages from published runbooks is not stored in the job history. To store verbose messages, configure published runbooks to Log Verbose Records on the Configure tab of the runbook in the Azure portal. In most cases, you should keep the default setting of not logging verbose records for a runbook for performance reasons. Turn on this option only to troubleshoot or debug a runbook.

When [testing a runbook](#), verbose messages are not displayed even if the runbook is configured to log verbose records. To display verbose messages while [testing a runbook](#), you must set the \$VerbosePreference variable to Continue. With that variable set, verbose messages are displayed in the Test Output Pane of the Azure portal.

Create a verbose message using the [Write-Verbose](#) cmdlet.

```
#The following line creates a verbose message.

Write-Verbose -Message "This is a verbose message."
```

Debug stream

The Debug stream is intended for use with an interactive user and should not be used in runbooks.

Progress records

If you configure a runbook to log progress records (on the Configure tab of the runbook in the Azure portal), then a record will be written to the job history before and after each activity is run. In most cases, you should keep the default setting of not logging progress records for a runbook in order to maximize performance. Turn on this option only to troubleshoot or debug a runbook. When testing a runbook, progress messages are not displayed even if the runbook is configured to log progress records.

The [Write-Progress](#) cmdlet is not valid in a runbook, since this is intended for use with an interactive user.

Preference variables

Windows PowerShell uses [preference variables](#) to determine how to respond to data sent to different output streams. You can set these variables in a runbook to control how it responds to data sent into different streams.

The following table lists the preference variables that can be used in runbooks with their valid and default values. This table only includes the values that are valid in a runbook. Additional values are valid for the preference variables when used in Windows PowerShell outside of Azure Automation.

VARIABLE	DEFAULT VALUE	VALID VALUES
WarningPreference	Continue	Stop Continue SilentlyContinue
ErrorActionPreference	Continue	Stop Continue SilentlyContinue
VerbosePreference	SilentlyContinue	Stop Continue SilentlyContinue

The following table lists the behavior for the preference variable values that are valid in runbooks.

VALUE	BEHAVIOR
Continue	Logs the message and continues executing the runbook.
SilentlyContinue	Continues executing the runbook without logging the message. This has the effect of ignoring the message.
Stop	Logs the message and suspends the runbook.

Retrieving runbook output and messages

Azure portal

You can view the details of a runbook job in the Azure portal from the Jobs tab of a runbook. The Summary of the job displays the input parameters and the [Output Stream](#) in addition to general information about the job and any exceptions if they occurred. The History includes messages from the [Output Stream](#) and [Warning and Error Streams](#) in addition to the [Verbose Stream](#) and [Progress Records](#) if the runbook is configured to log verbose and progress records.

Windows PowerShell

In Windows PowerShell, you can retrieve output and messages from a runbook using the [Get-AzureAutomationJobOutput](#) cmdlet. This cmdlet requires the ID of the job and has a parameter called Stream where you specify which stream to return. You can specify **Any** to return all streams for the job.

The following example starts a sample runbook and then waits for it to complete. Once completed, its output stream is collected from the job.

```

$job = Start-AzureRmAutomationRunbook -ResourceGroupName "ResourceGroup01" ` 
    -AutomationAccountName "MyAutomationAccount" -Name "Test-Runbook"

$doLoop = $true
While ($doLoop) {
    $job = Get-AzureRmAutomationJob -ResourceGroupName "ResourceGroup01" ` 
        -AutomationAccountName "MyAutomationAccount" -Id $job.JobId
    $status = $job.Status
    $doLoop = (($status -ne "Completed") -and ($status -ne "Failed") -and ($status -ne "Suspended") -and 
    ($status -ne "Stopped"))
}

Get-AzureRmAutomationJobOutput -ResourceGroupName "ResourceGroup01" ` 
    -AutomationAccountName "MyAutomationAccount" -Id $job.JobId -Stream Output

# For more detailed job output, pipe the output of Get-AzureRmAutomationJobOutput to Get-
# AzureRmAutomationJobOutputRecord
Get-AzureRmAutomationJobOutput -ResourceGroupName "ResourceGroup01" ` 
    -AutomationAccountName "MyAutomationAccount" -Id $job.JobId -Stream Any | Get-
    AzureRmAutomationJobOutputRecord

```

Graphical Authoring

For graphical runbooks, extra logging is available in the form of activity-level tracing. There are two levels of tracing: Basic and Detailed. In Basic tracing, you can see the start and end time of each activity in the runbook plus information related to any activity retries, such as number of attempts and start time of the activity. In Detailed tracing, you get Basic tracing plus input and output data for each activity. Currently the trace records are written using the verbose stream, so you must enable Verbose logging when you enable tracing. For graphical runbooks with tracing enabled, there is no need to log progress records, because the Basic tracing serves the same purpose and is more informative.



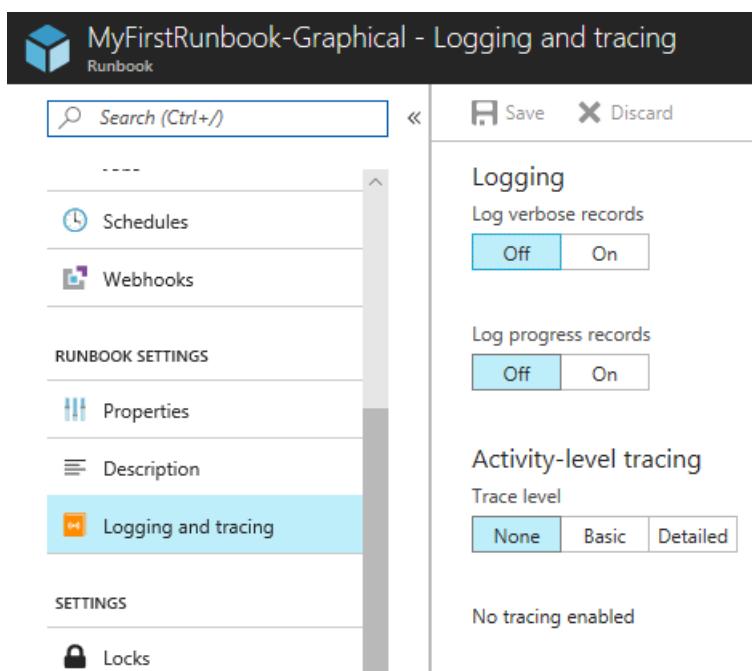
The screenshot shows a Windows application window titled 'Streams' with the sub-titles 'TestRunbookStreamsGraphical 2/7/2016, 6:55 PM'. The window contains a table with three columns: TIME, TYPE, and DETAILS. The table lists 18 rows of tracing events from 2/7/2016, 6:56 PM, categorized by severity: Verbose (14 rows), Warning (1 row), and Error (3 rows). The details column provides specific information about each event, such as activity names like 'Write-Verbose', 'ActivityStart', 'ActivityInput', etc., and their corresponding times and values.

TIME	TYPE	DETAILS
2/7/2016, 6:56 PM	Verbose	GraphTrace:[Activity:"Write-Verbose", Event:"ActivityStart", Time:"2016-02-08T02:56:20.1373078Z"]
2/7/2016, 6:56 PM	Verbose	GraphTrace:[Activity:"Write-Verbose", Event:"ActivityInput", Time:"2016-02-08T02:56:24.1372592Z", Val...
2/7/2016, 6:56 PM	Verbose	Verbose message
2/7/2016, 6:56 PM	Verbose	GraphTrace:[Activity:"Write-Verbose", Event:"ActivityOutput", Time:"2016-02-08T02:56:24.4653789Z", Val...
2/7/2016, 6:56 PM	Verbose	GraphTrace:[Activity:"Write-Verbose", Event:"ActivityEnd", Time:"2016-02-08T02:56:24.6528783Z", Dura...
2/7/2016, 6:56 PM	Verbose	GraphTrace:[Activity:"Write-Warning", Event:"ActivityStart", Time:"2016-02-08T02:56:24.7935039Z"]
2/7/2016, 6:56 PM	Verbose	GraphTrace:[Activity:"Write-Warning", Event:"ActivityInput", Time:"2016-02-08T02:56:24.8872483Z", Val...
2/7/2016, 6:56 PM	Warning	Warning message
2/7/2016, 6:56 PM	Verbose	GraphTrace:[Activity:"Write-Warning", Event:"ActivityOutput", Time:"2016-02-08T02:56:25.0903685Z", Val...
2/7/2016, 6:56 PM	Verbose	GraphTrace:[Activity:"Write-Warning", Event:"ActivityEnd", Time:"2016-02-08T02:56:25.1841205Z", Dur...
2/7/2016, 6:56 PM	Verbose	GraphTrace:[Activity:"Write-Error", Event:"ActivityStart", Time:"2016-02-08T02:56:25.2778702Z"]
2/7/2016, 6:56 PM	Verbose	GraphTrace:[Activity:"Write-Error", Event:"ActivityInput", Time:"2016-02-08T02:56:25.3716198Z", Values...
2/7/2016, 6:56 PM	Verbose	GraphTrace:[Activity:"Write-Error", Event:"ActivityOutput", Time:"2016-02-08T02:56:25.4966177Z", Valu...
2/7/2016, 6:56 PM	Error	Error message
2/7/2016, 6:56 PM	Verbose	GraphTrace:[Activity:"Write-Error", Event:"ActivityEnd", Time:"2016-02-08T02:56:26.7778651Z", Duratio...

You can see from the preceding screenshot that when you enable Verbose logging and tracing for Graphical runbooks, much more information is available in the production Job Streams view. This extra information can be essential for troubleshooting production problems with a runbook, and therefore you should only enable it for that purpose and not as a general practice. The Trace records can be especially numerous. With Graphical runbook tracing, you can get two to four records per activity depending on whether you have configured Basic or Detailed tracing. Unless you need this information to track the progress of a runbook for troubleshooting, you might want to keep Tracing turned off.

To enable activity-level tracing, perform the following steps:

1. In the Azure portal, open your Automation account.
2. Under **Process Automation**, select **Runbooks** to open the list of runbooks.
3. On the Runbooks page, click to select a graphical runbook from your list of runbooks.
4. Under **Settings**, click **Logging and tracing**.
5. On the Logging and Tracing page, under Log verbose records, click **On** to enable verbose logging and under Activity-level tracing, change the trace level to **Basic** or **Detailed** based on the level of tracing you require.



Microsoft Azure Log Analytics

Automation can send runbook job status and job streams to your Log Analytics workspace. With Log Analytics you can,

- Get insight on your Automation jobs
- Trigger an email or alert based on your runbook job status (for example, failed or suspended)
- Write advanced queries across your job streams
- Correlate jobs across Automation accounts
- Visualize your job history over time

For more information on how to configure integration with Log Analytics to collect, correlate and act on job data, see [Forward job status and job streams from Automation to Log Analytics](#).

Next steps

- To learn more about runbook execution, how to monitor runbook jobs, and other technical details, see [Track a runbook job](#)

- To understand how to design and use child runbooks, see [Child runbooks in Azure Automation](#)

Source control integration in Azure Automation

5/21/2018 • 6 minutes to read • [Edit Online](#)

Source control integration allows you to associate runbooks in your Automation account to a GitHub source control repository. Source control allows you to easily collaborate with your team, track changes, and roll back to earlier versions of your runbooks. For example, source control allows you to sync different branches in source control to your development, test or production Automation accounts, making it easy to promote code that has been tested in your development environment to your production Automation account.

Source control allows you to push code from Azure Automation to source control or pull your runbooks from source control to Azure Automation. This article describes how to set up source control in your Azure Automation environment. We start by configuring Azure Automation to access your GitHub repository and walk through different operations that can be done using source control integration.

NOTE

Source control supports pulling and pushing [PowerShell Workflow runbooks](#) as well as [PowerShell runbooks](#). [Graphical runbooks](#) are not yet supported.

There are two simple steps required to configure source control for your Automation account, and only one if you already have a GitHub account. They are:

Step 1 – Create a GitHub repository

If you already have a GitHub account and a repository that you want to link to Azure Automation, then sign into your existing account and start from step 2 below. Otherwise, navigate to [GitHub](#), sign up for a new account and [create a new repository](#).

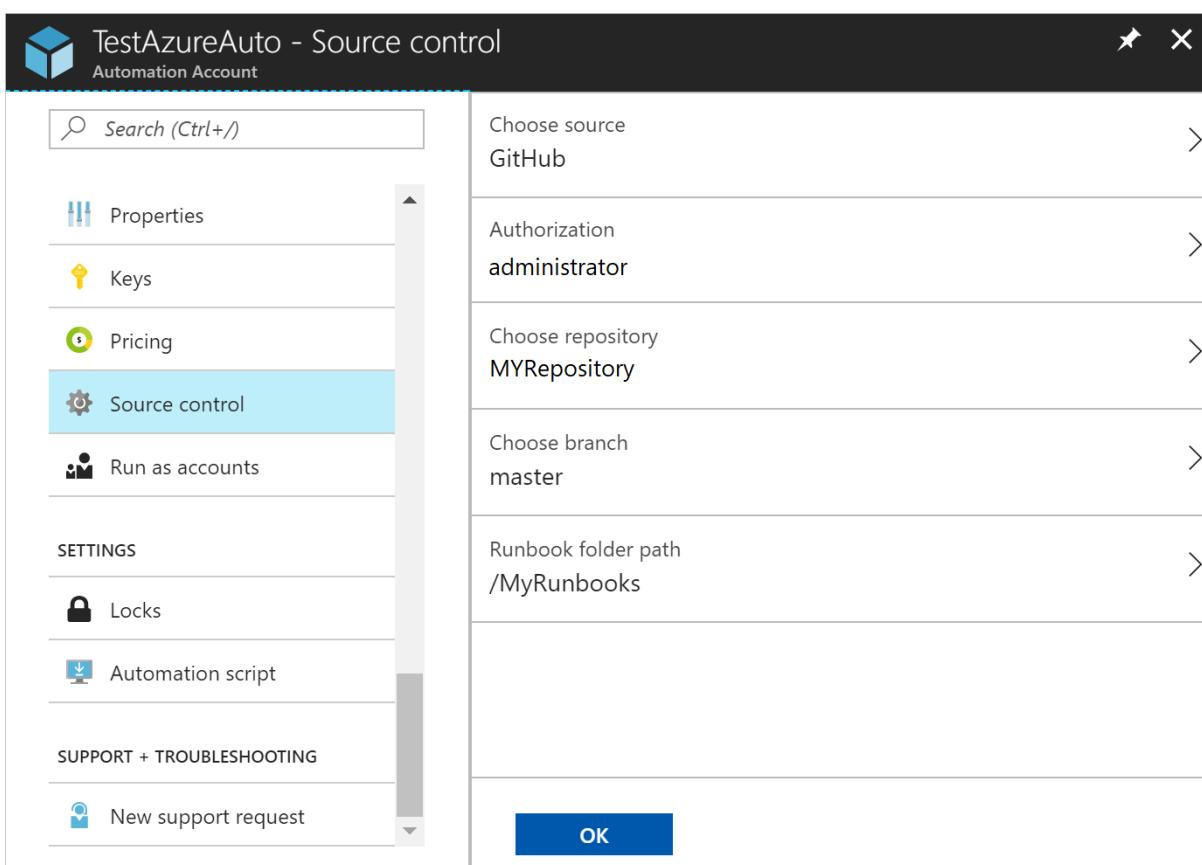
Step 2 – Set up source control in Azure Automation

1. From the Automation Account page in the Azure portal, under **Account Settings**, click **Source Control**.
2. The **Source Control** page opens, where you can configure your GitHub account details. Below is the list of parameters to configure:

PARAMETER	DESCRIPTION
Choose Source	Select the source. Currently, only GitHub is supported.
Authorization	Click the Authorize button to grant Azure Automation access to your GitHub repository. If you are already logged in to your GitHub account in a different window, then the credentials of that account are used. Once authorization is successful, the page will show your GitHub username under Authorization Property .
Choose repository	Select a GitHub repository from the list of available repositories.

PARAMETER	DESCRIPTION
Choose branch	Select a branch from the list of available branches. Only the master branch is shown if you haven't created any branches.
Runbook folder path	The runbook folder path specifies the path in the GitHub repository from which you want to push or pull your code. It must be entered in the format /foldername/subfoldername . Only runbooks in the runbook folder path will be synced to your Automation account. Runbooks in the subfolders of the runbook folder path will NOT be synced. Use / to sync all the runbooks under the repository.

3. For example, if you have a repository named **PowerShellScripts** that contains a folder named **RootFolder**, which contains a folder named **SubFolder**. You can use the following strings to sync each folder level:
 - a. To sync runbooks from **repository**, runbook folder path is **/**
 - b. To sync runbooks from **RootFolder**, runbook folder path is **/RootFolder**
 - c. To sync runbooks from **SubFolder**, runbook folder path is **/RootFolder/SubFolder**.
4. After you configure the parameters, they are displayed on the **Set Up Source Control** page.



5. Once you click **OK**, source control integration is now configured for your Automation account and should be updated with your GitHub information. You can now click on this part to view all of your source control sync job history.

Repository Synchronization

Source: GitHub

Repository: PowerShellScripts

Branch: master

Runbook folder path: /MyRunbooks



- After you set up source control, the following Automation resources will be created in your Automation account:

Two [variable assets](#) are created.

- The variable **Microsoft.Azure.Automation.SourceControl.Connection** contains the values of the connection string, as shown below.

PARAMETER	VALUE
Name	Microsoft.Azure.Automation.SourceControl.Connection
Type	String
Value	{"Branch":<Your branch name>,"RunbookFolderPath":<Runbook folder path>,"ProviderType":<has a value 1 for GitHub>,"Repository":<Name of your repository>,"Username":<Your GitHub user name>}

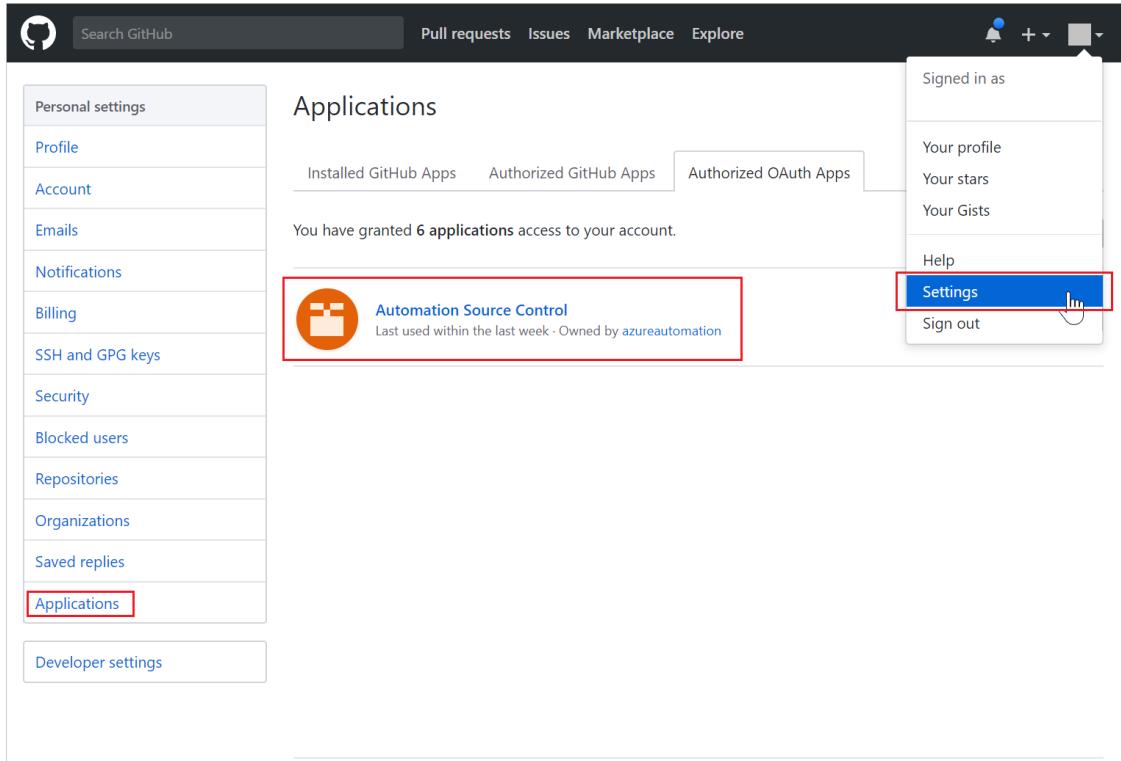
- The variable **Microsoft.Azure.Automation.SourceControl.OAuthToken**, contains the secure encrypted value of your OAuthToken.

PARAMETER	VALUE
Name	Microsoft.Azure.Automation.SourceControl.OAuthToke n
Type	Unknown(Encrypted)
Value	<Encrypted OAuthToken>

Variables			
NAME	TYPE	VALUE	LAST MODIFIED
Microsoft.Azure.Automati...	String	{"Branch":"master","RunbookFolderPath":"/My...	2/6/2018, 11:31 AM
Microsoft.Azure.Automati...	Unknown (encrypted)	*****	2/6/2018, 11:24 AM
VSToken	Unknown (encrypted)	*****	2/5/2018, 7:11 AM

- Automation Source Control** is added as an authorized application to your GitHub account. To

view the application: From your GitHub home page, navigate to your **profile > Settings > Applications**. This application allows Azure Automation to sync your GitHub repository to an Automation account.



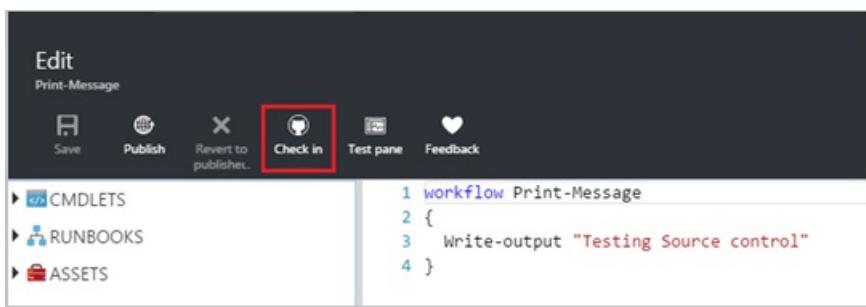
The screenshot shows the GitHub 'Applications' page. On the left is a sidebar with links like Personal settings, Profile, Account, Emails, Notifications, Billing, SSH and GPG keys, Security, Blocked users, Repositories, Organizations, Saved replies, Applications (which is selected and highlighted with a red box), and Developer settings. The main area is titled 'Applications' and shows three tabs: Installed GitHub Apps, Authorized GitHub Apps, and Authorized OAuth Apps. Under 'Authorized GitHub Apps', it says 'You have granted 6 applications access to your account.' A card for 'Automation Source Control' is shown, featuring an orange icon of a folder with files, the text 'Automation Source Control', and 'Last used within the last week - Owned by azureautomation'. In the top right corner, there's a 'Signed in as' dropdown menu with options: Your profile, Your stars, Your Gists, Help, Settings (which is highlighted with a red box), and Sign out.

Using Source Control in Automation

Check in a runbook from Azure Automation to source control

Runbook check in allows you to push the changes you have made to a runbook in Azure Automation into your source control repository. Below are the steps to check in a runbook:

1. From your Automation Account, [create a new textual runbook](#), or [edit an existing, textual runbook](#). This runbook can be either a PowerShell Workflow or a PowerShell script runbook.
2. After you edit your runbook, save it and click **check-in** from the **Edit** page.



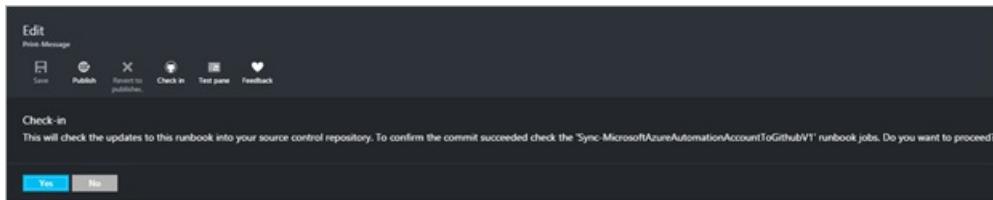
The screenshot shows the Azure Automation Runbook Editor. At the top, there's a toolbar with Save, Publish, Revert to publisher.., **Check in** (which is highlighted with a red box), Test pane, and Feedback. Below the toolbar, there's a navigation pane on the left with sections for CMDLETS, RUNBOOKS, and ASSETS. The main pane displays a PowerShell script for a 'Print-Message' workflow:

```
1 workflow Print-Message
2 {
3     Write-output "Testing Source control"
4 }
```

NOTE

Check-in from Azure Automation overwrites the code that currently exists in your source control. The Git equivalent command-line instruction to check-in is **git add + git commit + git push**

3. When you click **check-in**, you are prompted with a confirmation message, click **Yes** to continue.



4. Check-in starts the source control runbook: **Sync-MicrosoftAzureAutomationAccountToGitHubV1**.

This runbook connects to GitHub and pushes changes from Azure Automation to your repository. To view the checked in job history, go back to the **Source Control Integration** tab and click to open the Repository Synchronization page. This page shows all of your source control jobs. Select the job you want to view and click to view the details.

A screenshot of the Repository Synchronization page. It has a header 'Repository Synchronization' and a sub-header 'Jobs'. Below that are two buttons: 'Sync' and 'Disconnect'. A table lists a single job: STATUS: Completed, RUNBOOK: Sync-MicrosoftAzureAutomationAccountToGit..., CREATED: 10/27/2015, 9:14 PM, LAST UPDATED: 10/27/2015, 9:16 PM.

NOTE

Source control runbooks are special Automation runbooks that you cannot view or edit. While they do not show up on your runbook list, you see sync jobs showing in your jobs list.

5. The name of the modified runbook is sent as an input parameter for the checked in runbook. You can [view the job details](#) by expanding runbook in **Repository Synchronization** page.

A screenshot of the Job Overview page for the 'Sync-MicrosoftAzureAutomationAccountToGitHubV1' job. The job status is 'Completed'. On the right, there's a 'Runbook' section with a tooltip 'Runbook. This runbook is not available'. Below it, there's an 'INPUT' button highlighted with a blue box. To the right, there's a 'Input Parameters' panel with a table:

NAME	INPUT VALUE
RUNBOOKNAME	Print-Message

6. Refresh your GitHub repository once the job completes to view the changes. There should be a commit in your repository with a commit message: **Updated Runbook Name in Azure Automation**.

Sync runbooks from source control to Azure Automation

The sync button on the Repository Synchronization page allows you to pull all the runbooks in the runbook folder path of your repository to your Automation account. The same repository can be synced to more than one Automation account. Below are the steps to sync a runbook:

- From the Automation account where you set up source control, open the **Source Control Integration/Repository Synchronization** page and click **Sync**. You are prompted with a confirmation message, click **Yes** to continue.

A screenshot of the Repository Synchronization page. The 'Sync' button is highlighted with a red box. A message box at the bottom asks: 'Sync' This will sync all runbooks in the runbooks folder of your repository to this account. Do you want to proceed? There are 'Yes' and 'No' buttons at the bottom.

2. Sync starts the runbook: **Sync-MicrosoftAzureAutomationAccountFromGitHubV1**. This runbook connects to GitHub and pulls the changes from your repository to Azure Automation. You should see a new job on the **Repository Synchronization** page for this action. To view details about the sync job, click to open the job details page.

The screenshot shows the Azure Automation Source control interface. On the left, there's a navigation menu with options like Properties, Keys, Pricing, Source control (which is selected and highlighted in blue), and Run as accounts. Below that are sections for Settings (Locks, Automation script) and Support + Troubleshooting (New support request). The main area is titled "Repository Synchronization" and shows details: Source: GitHub, Repository: PowerShellScripts, Branch: master, and Runbook folder path: /MyRunbooks. It includes a GitHub logo icon. Below this, there's a table with two columns: STATUS and RUNBOOK. The STATUS row shows "Completed" with a green checkmark. The RUNBOOK row shows "Sync-MicrosoftAzureAutomationAccountFro..." followed by a "Load more" link. At the top right, there are "Sync" and "Disconnect" buttons.

NOTE

A sync from source control overwrites the draft version of the runbooks that currently exist in your Automation account for **ALL** runbooks that are currently in source control. The Git equivalent command-line instruction to sync is **git pull**

Troubleshooting source control problems

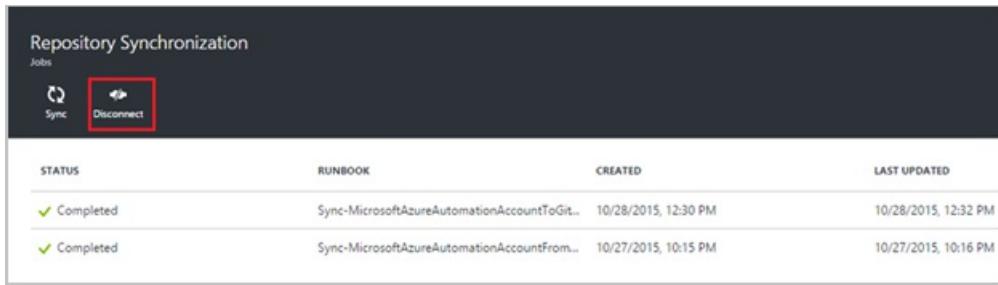
If there are any errors with a check in or sync job, the job status should be suspended and you can view more details about the error in the job page. The **All Logs** part shows you all the PowerShell streams associated with that job. This provides you with the details needed to help you fix any problems with your check in or sync. It also shows you the sequence of actions that occurred while syncing or checking in a runbook.

The screenshot shows the Azure Automation Job details page for the job "Sync-MicrosoftAzureAutomationAccountFromGitHubV1" from 10/23/2015. The left pane displays the "Overview" section with a summary of the job, including its ID, creation date, last update, and status (Suspended). It also shows the number of errors (0), warnings (1), and logs (All Logs). The right pane is titled "Streams" and shows a table of log entries with columns for TIME, TYPE, and DETAILS. The log entries detail the sync process, including connecting to GitHub, saving runbooks, and encountering an error where "File NewTestConfig cannot be saved to the Automation account. Make sure the definition of the file exists." The log also notes that 1 runbook was found and synchronized successfully, and that one runbook failed to sync due to the configuration issue.

Disconnecting source control

To disconnect from your GitHub account, open the Repository Synchronization page and click **Disconnect**. Once

you disconnect source control, runbooks that were synced earlier still remain in your Automation account but the Repository Synchronization page will not be enabled.



The screenshot shows the 'Repository Synchronization' page in the Azure portal. At the top, there are two buttons: 'Sync' and 'Disconnect'. The 'Disconnect' button is highlighted with a red box. Below the buttons is a table with four columns: STATUS, RUNBOOK, CREATED, and LAST UPDATED. There are two rows in the table, both of which are marked as 'Completed' with green checkmarks. The first row corresponds to 'Sync-MicrosoftAzureAutomationAccountToGit...' and was created on 10/28/2015, 12:30 PM, last updated on 10/28/2015, 12:32 PM. The second row corresponds to 'Sync-MicrosoftAzureAutomationAccountFrom...' and was created on 10/27/2015, 10:15 PM, last updated on 10/27/2015, 10:16 PM.

STATUS	RUNBOOK	CREATED	LAST UPDATED
✓ Completed	Sync-MicrosoftAzureAutomationAccountToGit...	10/28/2015, 12:30 PM	10/28/2015, 12:32 PM
✓ Completed	Sync-MicrosoftAzureAutomationAccountFrom...	10/27/2015, 10:15 PM	10/27/2015, 10:16 PM

Next steps

For more information about source control integration, see the following resources:

- [Azure Automation: Source Control Integration in Azure Automation](#)
- [Vote for your favorite source control system](#)
- [Azure Automation: Integrating Runbook Source Control using Visual Studio Team Services](#)

Starting a runbook in Azure Automation

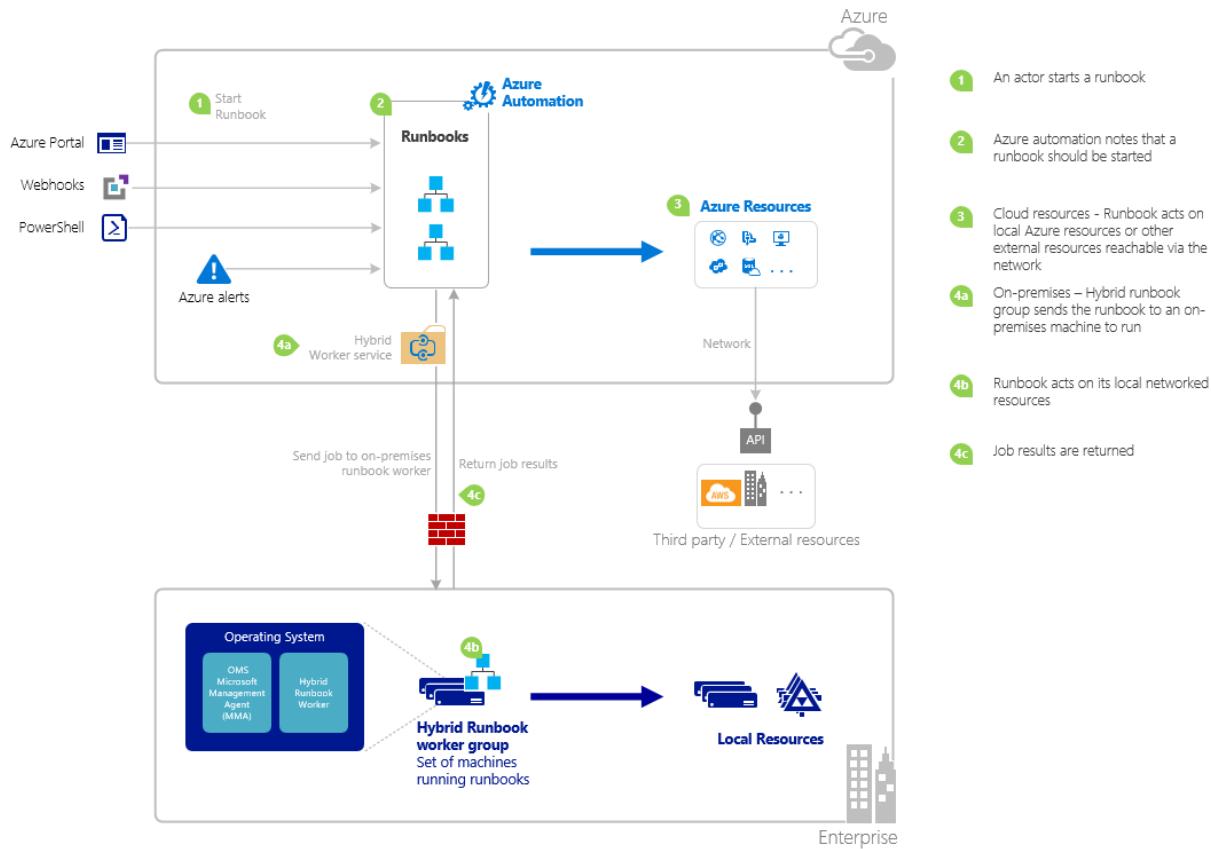
5/21/2018 • 5 minutes to read • [Edit Online](#)

The following table helps you determine the method to start a runbook in Azure Automation that is most suitable to your particular scenario. This article includes details on starting a runbook with the Azure portal and with Windows PowerShell. Details on the other methods are provided in other documentation that you can access from the links below.

METHOD	CHARACTERISTICS
Azure portal	<ul style="list-style-type: none">Simplest method with interactive user interface.Form to provide simple parameter values.Easily track job state.Access authenticated with Azure logon.
Windows PowerShell	<ul style="list-style-type: none">Call from command line with Windows PowerShell cmdlets.Can be included in automated solution with multiple steps.Request is authenticated with certificate or OAuth user principal / service principal.Provide simple and complex parameter values.Track job state.Client required to support PowerShell cmdlets.
Azure Automation API	<ul style="list-style-type: none">Most flexible method but also most complex.Call from any custom code that can make HTTP requests.Request authenticated with certificate, or OAuth user principal / service principal.Provide simple and complex parameter values. <i>If you are calling a Python runbook using the API, the JSON payload must be serialized.</i>Track job state.
Webhooks	<ul style="list-style-type: none">Start runbook from single HTTP request.Authenticated with security token in URL.Client cannot override parameter values specified when webhook created. Runbook can define single parameter that is populated with the HTTP request details.No ability to track job state through webhook URL.
Respond to Azure Alert	<ul style="list-style-type: none">Start a runbook in response to Azure alert.Configure webhook for runbook and link to alert.Authenticated with security token in URL.
Schedule	<ul style="list-style-type: none">Automatically start runbook on hourly, daily, weekly, or monthly schedule.Manipulate schedule through Azure portal, PowerShell cmdlets, or Azure API.Provide parameter values to be used with schedule.
From Another Runbook	<ul style="list-style-type: none">Use a runbook as an activity in another runbook.Useful for functionality used by multiple runbooks.Provide parameter values to child runbook and use output in parent runbook.

The following image illustrates detailed step-by-step process in the life cycle of a runbook. It includes different

ways a runbook is started in Azure Automation, components required for Hybrid Runbook Worker to execute Azure Automation runbooks and interactions between different components. To learn about executing Automation runbooks in your datacenter, refer to [hybrid runbook workers](#)



Starting a runbook with the Azure portal

- In the Azure portal, select **Automation** and then click the name of an automation account.
- On the Hub menu, select **Runbooks**.
- On the **Runbooks** page, select a runbook, and then click **Start**.
- If the runbook has parameters, you are prompted to provide values with a text box for each parameter. See [Runbook Parameters](#) below for further details on parameters.
- On the **Job** page, you can view the status of the runbook job.

Starting a runbook with Windows PowerShell

You can use the [Start-AzureRmAutomationRunbook](#) to start a runbook with Windows PowerShell. The following sample code starts a runbook called Test-Runbook.

```
Start-AzureRmAutomationRunbook -AutomationAccountName "MyAutomationAccount" -Name "Test-Runbook" -ResourceGroupName "ResourceGroup01"
```

Start-AzureRmAutomationRunbook returns a job object that you can use to track its status once the runbook is started. You can then use this job object with [Get-AzureRmAutomationJob](#) to determine the status of the job and [Get-AzureRmAutomationJobOutput](#) to get its output. The following sample code starts a runbook called Test-Runbook, waits until it has completed, and then displays its output.

```

$runbookName = "Test-Runbook"
$ResourceGroup = "ResourceGroup01"
$AutomationAcct = "MyAutomationAccount"

$job = Start-AzureRmAutomationRunbook -AutomationAccountName $AutomationAcct -Name $runbookName -
ResourceGroupName $ResourceGroup

$doLoop = $true
While ($doLoop) {
    $job = Get-AzureRmAutomationJob -AutomationAccountName $AutomationAcct -Id $job.JobId -ResourceGroupName
$ResourceGroup
    $status = $job.Status
    $doLoop = (($status -ne "Completed") -and ($status -ne "Failed") -and ($status -ne "Suspended") -and
($status -ne "Stopped"))
}

Get-AzureRmAutomationJobOutput -AutomationAccountName $AutomationAcct -Id $job.JobId -ResourceGroupName
$ResourceGroup -Stream Output

```

If the runbook requires parameters, then you must provide them as a [hashtable](#) where the key of the hashtable matches the parameter name and the value is the parameter value. The following example shows how to start a runbook with two string parameters named FirstName and LastName, an integer named RepeatCount, and a boolean parameter named Show. For more information on parameters, see [Runbook Parameters](#) below.

```

$params = @{"FirstName"="Joe";"LastName"="Smith";"RepeatCount"=2;"Show"=$true}
Start-AzureRmAutomationRunbook -AutomationAccountName "MyAutomationAccount" -Name "Test-Runbook" -
ResourceGroupName "ResourceGroup01" -Parameters $params

```

Runbook parameters

When you start a runbook from the Azure portal or Windows PowerShell, the instruction is sent through the Azure Automation web service. This service does not support parameters with complex data types. If you need to provide a value for a complex parameter, then you must call it inline from another runbook as described in [Child runbooks in Azure Automation](#).

The Azure Automation web service provides special functionality for parameters using certain data types as described in the following sections:

Named values

If the parameter is data type [object], then you can use the following JSON format to send it a list of named values: `{Name1:'Value1', Name2:'Value2', Name3:'Value3'}`. These values must be simple types. The runbook receives the parameter as a [PSCustomObject](#) with properties that correspond to each named value.

Consider the following test runbook that accepts a parameter called user.

```

Workflow Test-Parameters
{
    param (
        [Parameter(Mandatory=$true)][object]$user
    )
    $userObject = $user | ConvertFrom-JSON
    if ($userObject.Show) {
        foreach ($i in 1..$userObject.RepeatCount) {
            $userObject.FirstName
            $userObject.LastName
        }
    }
}

```

The following text could be used for the user parameter.

```
{FirstName:'Joe',LastName:'Smith',RepeatCount:'2',Show:'True'}
```

This results in the following output:

```
Joe  
Smith  
Joe  
Smith
```

Arrays

If the parameter is an array such as [array] or [string[]], then you can use the following JSON format to send it a list of values: *[Value1, Value2, Value3]*. These values must be simple types.

Consider the following test runbook that accepts a parameter called *user*.

```
Workflow Test-Parameters
{
    param (
        [Parameter(Mandatory=$true)][array]$user
    )
    if ($user[3]) {
        foreach ($i in 1..$user[2]) {
            $ user[0]
            $ user[1]
        }
    }
}
```

The following text could be used for the user parameter.

```
["Joe","Smith",2,true]
```

This results in the following output:

```
Joe  
Smith  
Joe  
Smith
```

Credentials

If the parameter is data type **PSCredential**, then you can provide the name of an Azure Automation [credential asset](#). The runbook retrieves the credential with the name that you specify.

Consider the following test runbook that accepts a parameter called *credential*.

```
Workflow Test-Parameters
{
    param (
        [Parameter(Mandatory=$true)][PSCredential]$credential
    )
    $credential.UserName
}
```

The following text could be used for the user parameter assuming that there was a credential asset called *My Credential*.

```
My Credential
```

Assuming the username in the credential was *jsmith*, this results in the following output:

```
jsmith
```

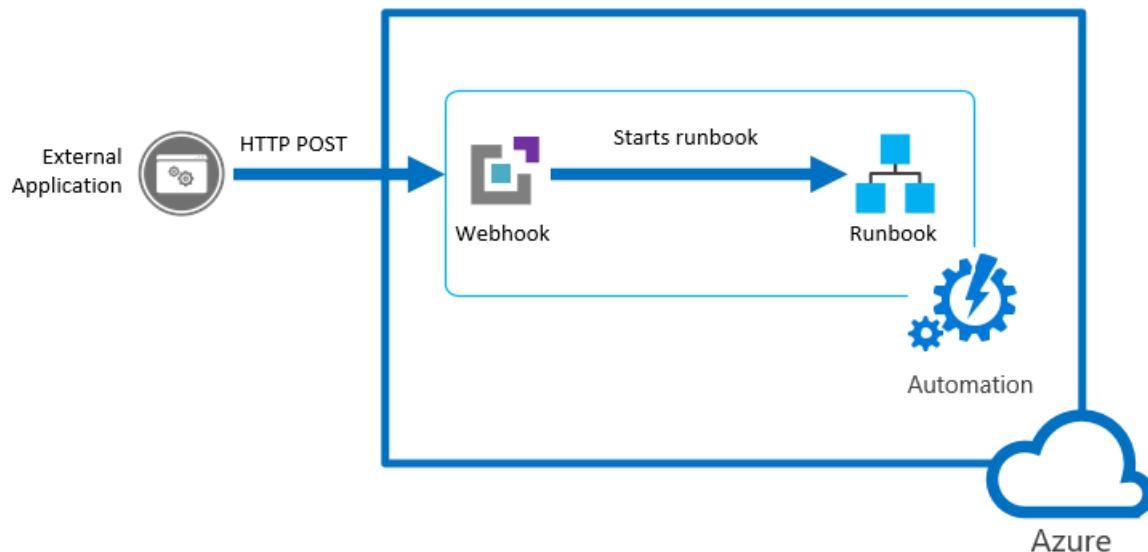
Next steps

- The runbook architecture in current article provides a high-level overview of runbooks managing resources in Azure and on-premises with the Hybrid Runbook Worker. To learn about executing Automation runbooks in your datacenter, refer to [Hybrid Runbook Workers](#).
- To learn more about the creating modular runbooks to be used by other runbooks for specific or common functions, refer to [Child Runbooks](#).

Starting an Azure Automation runbook with a webhook

7/2/2018 • 8 minutes to read • [Edit Online](#)

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



You can compare webhooks to other methods of starting a runbook in [Starting a runbook in Azure Automation](#)

Details of a webhook

The following table describes the properties that you must configure for a webhook.

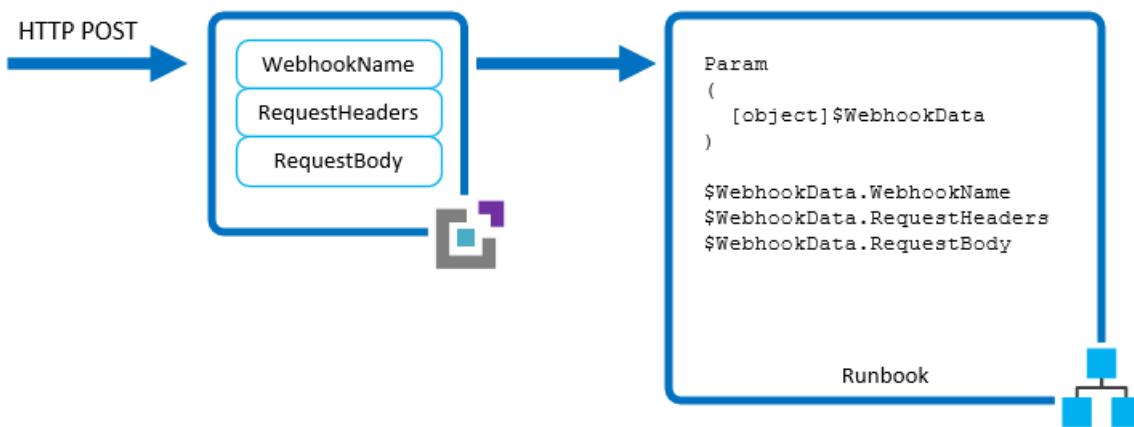
PROPERTY	DESCRIPTION
Name	You can provide any name you want for a webhook since this is not exposed to the client. It is only used for you to identify the runbook in Azure Automation. As a best practice, you should give the webhook a name related to the client that uses it.
URL	The URL of the webhook is the unique address that a client calls with an HTTP POST to start the runbook linked to the webhook. It is automatically generated when you create the webhook. You cannot specify a custom URL. The URL contains a security token that allows the runbook to be invoked by a third-party system with no further authentication. For this reason, it should be treated like a password. For security reasons, you can only view the URL in the Azure portal at the time the webhook is created. Note the URL in a secure location for future use.

PROPERTY	DESCRIPTION
Expiration date	Like a certificate, each webhook has an expiration date at which time it can no longer be used. This expiration date can be modified after the webhook is created.
Enabled	A webhook is enabled by default when it is created. If you set it to Disabled, then no client is able to use it. You can set the Enabled property when you create the webhook or anytime once it is created.

Parameters

A webhook can define values for runbook parameters that are used when the runbook is started by that webhook. The webhook must include values for any mandatory parameters of the runbook and may include values for optional parameters. A parameter value configured to a webhook can be modified even after creating the webhook. Multiple webhooks linked to a single runbook can each use different parameter values.

When a client starts a runbook using a webhook, it cannot override the parameter values defined in the webhook. To receive data from the client, the runbook can accept a single parameter called **\$WebhookData** of type [object] that contains data that the client includes in the POST request.



The **\$WebhookData** object has the following properties:

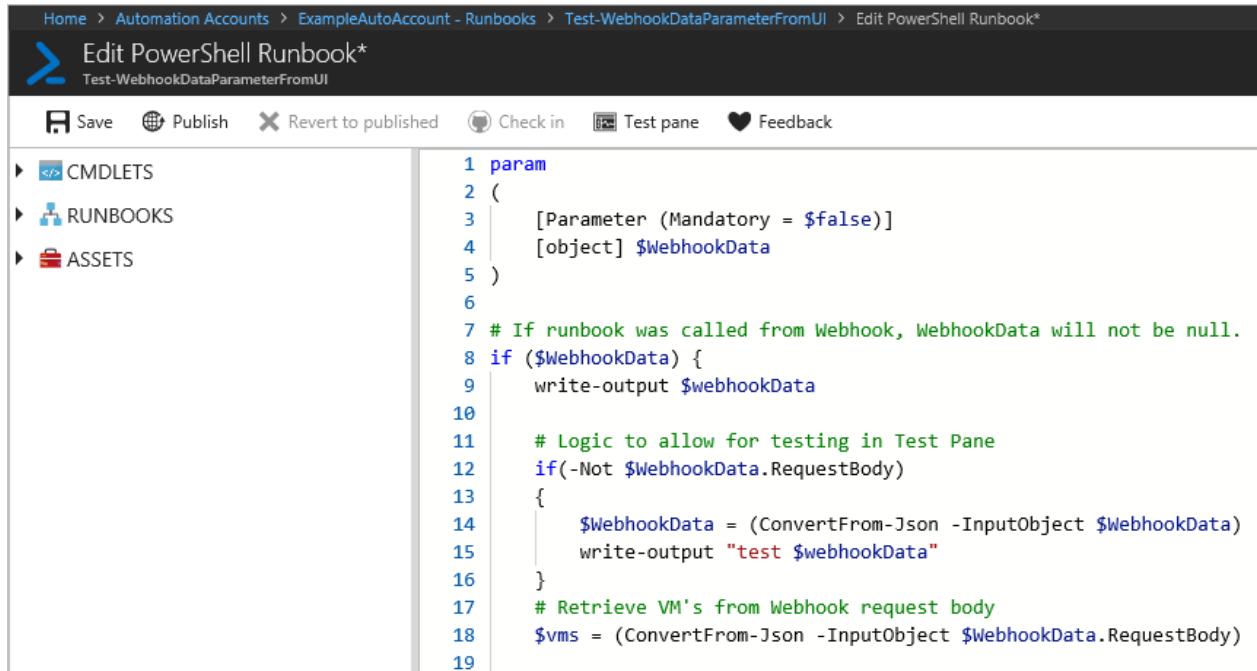
PROPERTY	DESCRIPTION
WebhookName	The name of the webhook.
RequestHeader	Hash table containing the headers of the incoming POST request.
RequestBody	The body of the incoming POST request. This retains any formatting such as string, JSON, XML, or form encoded data. The runbook must be written to work with the data format that is expected.

There is no configuration of the webhook required to support the **\$WebhookData** parameter, and the runbook is not required to accept it. If the runbook does not define the parameter, then any details of the request sent from the client is ignored.

If you specify a value for **\$WebhookData** when you create the webhook, that value is overridden when the

webhook starts the runbook with the data from the client POST request, even if the client does not include any data in the request body. If you start a runbook that has \$WebhookData using a method other than a webhook, you can provide a value for \$WebhookData that is recognized by the runbook. This value should be an object with the same [properties](#) as \$WebhookData so that the runbook can properly work with it as if it was working with actual WebhookData passed by a webhook.

For example, if you are starting the following runbook from the Azure portal and want to pass some sample WebhookData for testing, since WebhookData is an object, it should be passed as JSON in the UI.



```

Home > Automation Accounts > ExampleAutoAccount - Runbooks > Test-WebhookDataParameterFromUI > Edit PowerShell Runbook*
Edit PowerShell Runbook*
Save Publish Revert to published Check in Test pane Feedback
▶ CMDLETS
▶ RUNBOOKS
▶ ASSETS
1 param
2 (
3     [Parameter (Mandatory = $false)]
4     [object] $WebhookData
5 )
6
7 # If runbook was called from Webhook, WebhookData will not be null.
8 if ($WebhookData) {
9     write-output $WebhookData
10
11     # Logic to allow for testing in Test Pane
12     if(-Not $WebhookData.RequestBody)
13     {
14         $WebhookData = (ConvertFrom-Json -InputObject $WebhookData)
15         write-output "test $WebhookData"
16     }
17     # Retrieve VM's from Webhook request body
18     $vms = (ConvertFrom-Json -InputObject $WebhookData.RequestBody)
19

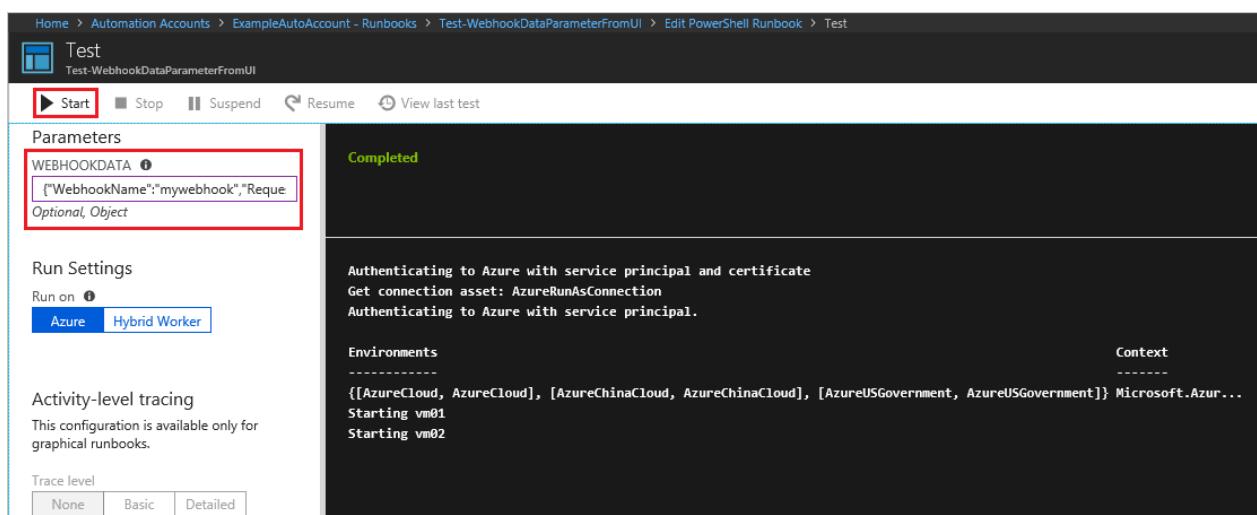
```

For the following runbook, if you have the following properties for the WebhookData parameter:

- WebhookName: *MyWebhook*
- RequestBody: `[{'ResourceGroup': 'myResourceGroup','Name': 'vm01'},{'ResourceGroup': 'myResourceGroup','Name': 'vm02'}]`

Then you would pass the following JSON value in the UI for the WebhookData parameter. The following example with the carriage returns and newline characters matches the format that is passed in from a webhook.

```
{"WebhookName": "mywebhook", "RequestBody": "[\r\n { \r\n \"ResourceGroup\": \"vm01\", \r\n \"Name\": \"vm01\" \r\n }, \r\n { \r\n \"ResourceGroup\": \"vm02\", \r\n \"Name\": \"vm02\" \r\n } \r\n ]"}
```



NOTE

The values of all input parameters are logged with the runbook job. This means that any input provided by the client in the webhook request will be logged and available to anyone with access to the automation job. For this reason, you should be cautious about including sensitive information in webhook calls.

Security

The security of a webhook relies on the privacy of its URL, which contains a security token that allows it to be invoked. Azure Automation does not perform any authentication on the request as long as it is made to the correct URL. For this reason, webhooks should not be used for runbooks that perform highly sensitive functions without using an alternate means of validating the request.

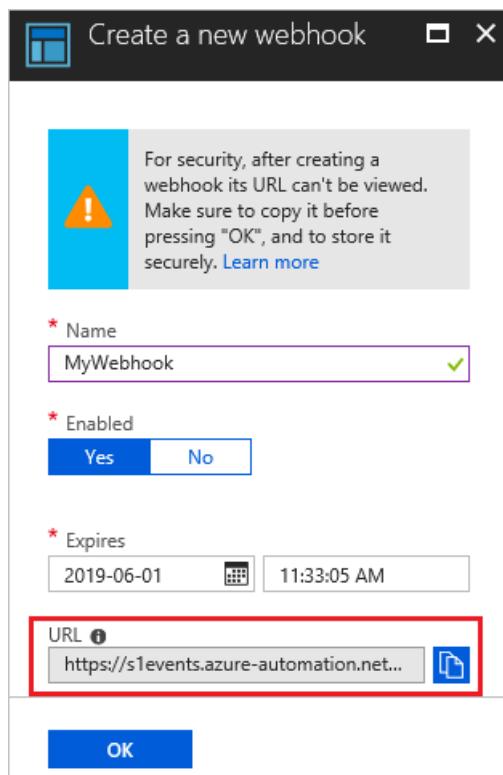
You can include logic within the runbook to determine that it was called by a webhook by checking the **WebhookName** property of the `$WebhookData` parameter. The runbook could perform further validation by looking for particular information in the **RequestHeader** or **RequestBody** properties.

Another strategy is to have the runbook perform some validation of an external condition when it received a webhook request. For example, consider a runbook that is called by GitHub whenever there is a new commit to a GitHub repository. The runbook might connect to GitHub to validate that a new commit had occurred before continuing.

Creating a webhook

Use the following procedure to create a new webhook linked to a runbook in the Azure portal.

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.
3. Click **Create new webhook** to open the **Create webhook page**.
4. Specify a **Name**, **Expiration Date** for the webhook and whether it should be enabled. See [Details of a webhook](#) for more information these properties.
5. Click the copy icon and press Ctrl+C to copy the URL of the webhook. Then record it in a safe place. **Once you create the webhook, you cannot retrieve the URL again.**



6. Click **Parameters** to provide values for the runbook parameters. If the runbook has mandatory parameters, then you are not able to create the webhook unless values are provided.
7. Click **Create** to create the webhook.

Using a webhook

To use a webhook after it has been created, your client application must issue an HTTP POST with the URL for the webhook. The syntax of the webhook is in the following format:

```
http://<Webhook Server>/token?=<Token Value>
```

The client receives one of the following return codes from the POST request.

CODE	TEXT	DESCRIPTION
202	Accepted	The request was accepted, and the runbook was successfully queued.
400	Bad Request	The request was not accepted for one of the following reasons: <ul style="list-style-type: none"> • The webhook has expired. • The webhook is disabled. • The token in the URL is invalid.
404	Not Found	The request was not accepted for one of the following reasons: <ul style="list-style-type: none"> • The webhook was not found. • The runbook was not found. • The account was not found.

Code	Text	Description
500	Internal Server Error	The URL was valid, but an error occurred. Please resubmit the request.

Assuming the request is successful, the webhook response contains the job id in JSON format as follows. It will contain a single job id, but the JSON format allows for potential future enhancements.

```
{"JobIds":["<JobId>"]}
```

The client cannot determine when the runbook job completes or its completion status from the webhook. It can determine this information using the job id with another method such as [Windows PowerShell](#) or the [Azure Automation API](#).

Sample runbook

The following sample runbook accepts the accepts webhook data and starts the virtual machines specified in the request body. To test this runbook, in your Automation Account under **Runbooks**, click **+ Add a runbook**. If you do not know how to create a runbook, see [Creating a runbook](#).

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

# If runbook was called from Webhook, WebhookData will not be null.
if ($WebhookData) {

    # 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"
    Write-Output "Get connection asset: $ConnectionAssetName"

    $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 | Write-Output

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

Test the example

The following example uses Windows PowerShell to start a runbook with a webhook. Any language that can make an HTTP request can use a webhook; Windows PowerShell is used here as an example.

The runbook is expecting a list of virtual machines formatted in JSON in the body of the request.

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

The following example shows the body of the request that is available to the runbook in the **RequestBody** property of **WebhookData**. This is formatted as JSON because that was the format that was included in the body of the request.

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

The following image shows the request being sent from Windows PowerShell and the resulting response. The job id is extracted from the response and converted to a string.

```
PS C:\> $response = Invoke-WebRequest -Method Post -Uri $uri -Headers $headers -Body $body  
PS C:\> $response  
  
StatusCode      : 202  
StatusDescription : Accepted  
Content         : {"JobIds":["308b5691-3b8c-4c73-a896-8b956ea849fa"]}  
RawContent      : HTTP/1.1 202 Accepted  
                Pragma: no-cache  
                Strict-Transport-Security: max-age=31536000; includeSubDomains  
                Content-Length: 51  
                Cache-Control: no-cache  
                Content-Type: application/json; charset=utf-8  
                Date...  
Forms           : {}  
Headers         : {[Pragma, no-cache], [Strict-Transport-Security, max-age=31536000; includeSubDomains], [Content-Length, 51], [Cache-Control, no-cache]...}  
Images          : {}  
InputFields     : {}  
Links           : {}  
ParsedHtml      : mshtml.HTMLDocumentClass  
RawContentLength : 51  
  
PS C:\> $jobid = (ConvertFrom-Json ($response.Content)).jobids[0]  
PS C:\> $jobid  
308b5691-3b8c-4c73-a896-8b956ea849fa  
PS C:\> |
```

Next steps

- To learn how to use Azure Automation to take action on Azure Alerts, see [Use an alert to trigger an Azure Automation runbook](#).

Runbook input parameters

5/21/2018 • 10 minutes to read • [Edit Online](#)

Runbook input parameters increase the flexibility of runbooks by allowing you to pass data to it when it is started. The parameters allow the runbook actions to be targeted for specific scenarios and environments. In this article, you walk through different scenarios where input parameters are used in runbooks.

Configure input parameters

Input parameters can be configured in PowerShell, PowerShell Workflow, Python, and graphical runbooks. A runbook can have multiple parameters with different data types, or no parameters at all. Input parameters can be mandatory or optional, and you can assign a default value for optional parameters. You can assign values to the input parameters for a runbook when you start it through one of the available methods. These methods include starting a runbook from the portal or a web service. You can also start one as a child runbook that is called inline in another runbook.

Configure input parameters in PowerShell and PowerShell Workflow runbooks

PowerShell and [PowerShell Workflow runbooks](#) in Azure Automation support input parameters that are defined through the following attributes:

PROPERTY	DESCRIPTION
Type	Required. The data type expected for the parameter value. Any .NET type is valid.
Name	Required. The name of the parameter. This must be unique within the runbook, and can contain only letters, numbers, or underscore characters. It must start with a letter.
Mandatory	Optional. Specifies whether a value must be provided for the parameter. If you set this to \$true , then a value must be provided when the runbook is started. If you set this to \$false , then a value is optional.
Default value	Optional. Specifies a value that is used for the parameter if a value is not passed in when the runbook is started. A default value can be set for any parameter and will automatically make the parameter optional regardless of the Mandatory setting.

Windows PowerShell supports more attributes of input parameters than those listed here, like validation, aliases, and parameter sets. However, Azure Automation currently supports only the preceding input parameters.

A parameter definition in PowerShell Workflow runbooks has the following general form, where multiple parameters are separated by commas.

```

Param
(
    [Parameter (Mandatory= $true/$false)]
    [Type] $Name1 = <Default value>,

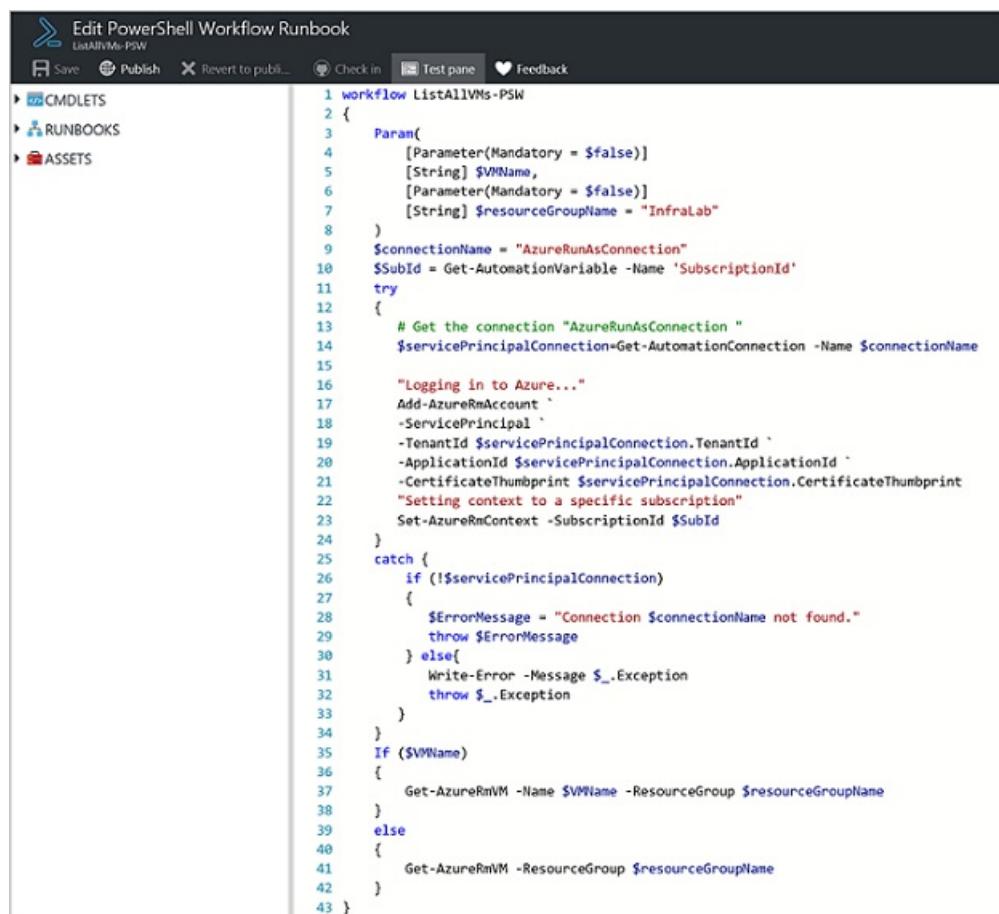
    [Parameter (Mandatory= $true/$false)]
    [Type] $Name2 = <Default value>
)

```

NOTE

When you're defining parameters, if you don't specify the **Mandatory** attribute, then by default, the parameter is considered optional. Also, if you set a default value for a parameter in PowerShell Workflow runbooks, it is treated by PowerShell as an optional parameter, regardless of the **Mandatory** attribute value.

As an example, let's configure the input parameters for a PowerShell Workflow runbook that outputs details about virtual machines, either a single VM or all VMs within a resource group. This runbook has two parameters as shown in the following screenshot: the name of virtual machine and the name of the resource group.



The screenshot shows the 'Edit PowerShell Workflow Runbook' interface. The left sidebar lists 'CMDLETS', 'RUNBOOKS', and 'ASSETS'. The main pane displays the PowerShell script for a runbook named 'ListAllVMs-PSW'. The script defines a workflow 'ListAllVMs-PSW' with a parameter block. It attempts to connect to Azure using a service principal, sets context to a specific subscription, and then retrieves either a single VM or all VMs in a specified resource group based on the input parameter '\$VMName'.

```

1 workflow ListAllVMs-PSW
2 {
3     Param(
4         [Parameter(Mandatory = $false)]
5         [String] $VMName,
6         [Parameter(Mandatory = $false)]
7         [String] $resourceGroupName = "InfraLab"
8     )
9     $connectionName = "AzureRunAsConnection"
10    $subId = Get-AutomationVariable -Name 'SubscriptionId'
11    try
12    {
13        # Get the connection "AzureRunAsConnection"
14        $servicePrincipalConnection = Get-AutomationConnection -Name $connectionName
15
16        "Logging in to Azure..."
17        Add-AzureRmAccount `
18            -ServicePrincipal `
19            -TenantId $servicePrincipalConnection.TenantId `
20            -ApplicationId $servicePrincipalConnection.ApplicationId `
21            -CertificateThumbprint $servicePrincipalConnection.CertificateThumbprint
22        "Setting context to a specific subscription"
23        Set-AzureRmContext -SubscriptionId $subId
24    }
25    catch {
26        if (!$servicePrincipalConnection)
27        {
28            $errorMessage = "Connection $connectionName not found."
29            throw $errorMessage
30        }
31        else{
32            Write-Error -Message $_.Exception
33            throw $_.Exception
34        }
35    }
36    IF ($VMName)
37    {
38        Get-AzureRmVM -Name $VMName -ResourceGroup $resourceGroupName
39    }
39    else
40    {
41        Get-AzureRmVM -ResourceGroup $resourceGroupName
42    }
43 }

```

In this parameter definition, the parameters **\$VMName** and **\$resourceGroupName** are simple parameters of type string. However, PowerShell and PowerShell Workflow runbooks support all simple types and complex types, such as **object** or **PSCredential** for input parameters.

If your runbook has an object type input parameter, then use a PowerShell hashtable with (name, value) pairs to pass in a value. For example, if you have the following parameter in a runbook:

```

[Parameter (Mandatory = $true)]
[object] $FullName

```

Then you can pass the following value to the parameter:

```
@{"FirstName"="Joe";"MiddleName"="Bob";"LastName"="Smith"}
```

Configure input parameters in graphical runbooks

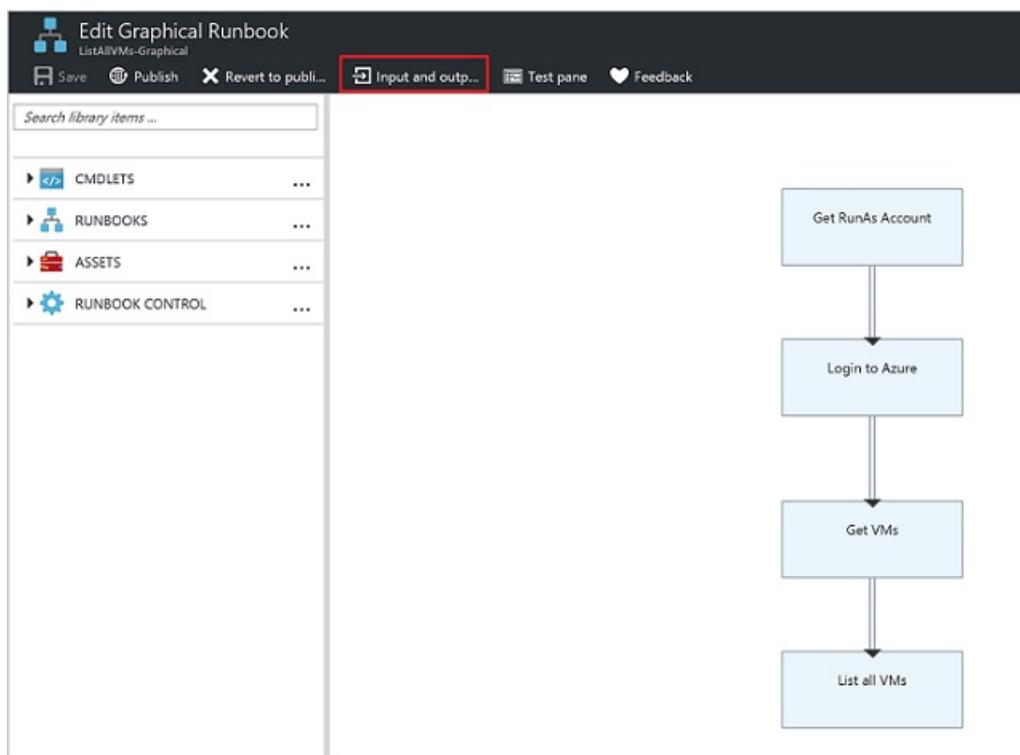
To [configure a graphical runbook](#) with input parameters, let's create a graphical runbook that outputs details about virtual machines, either a single VM or all VMs within a resource group. Configuring a runbook consists of two major activities, as described below.

Authenticate Runbooks with Azure Run As account to authenticate with Azure.

Get-AzureRmVm to get the properties of a virtual machine.

You can use the **Write-Output** activity to output the names of virtual machines. The activity **Get-AzureRmVm** accepts two parameters, the **virtual machine name** and the **resource group name**. Since these parameters could require different values each time you start the runbook, you can add input parameters to your runbook. Here are the steps to add input parameters:

1. Select the graphical runbook from the **Runbooks** blade and then click **Edit** it.
2. From the runbook editor, click **Input and output** to open the **Input and output** blade.



3. The **Input and output** blade displays a list of input parameters that are defined for the runbook. On this blade, you can either add a new input parameter or edit the configuration of an existing input parameter. To add a new parameter for the runbook, click **Add input** to open the **Runbook input parameter** blade.

There, you can configure the following parameters:

PROPERTY	DESCRIPTION
Name	Required. The name of the parameter. This must be unique within the runbook, and can contain only letters, numbers, or underscore characters. It must start with a letter.

PROPERTY	DESCRIPTION
Description	Optional. Description about the purpose of input parameter.
Type	Optional. The data type that's expected for the parameter value. Supported parameter types are String , Int32 , Int64 , Decimal , Boolean , DateTime , and Object . If a data type is not selected, it defaults to String .
Mandatory	Optional. Specifies whether a value must be provided for the parameter. If you choose yes , then a value must be provided when the runbook is started. If you choose no , then a value is not required when the runbook is started, and a default value may be set.
Default Value	Optional. Specifies a value that is used for the parameter if a value is not passed in when the runbook is started. A default value can be set for a parameter that's not mandatory. To set a default value, choose Custom . This value is used unless another value is provided when the runbook is started. Choose None if you don't want to provide any default value.

Runbook Input Param...

* Name ⓘ
resourceGroupName ✓

Description ⓘ

Type ⓘ
String

Mandatory ⓘ
Yes No

Default value ⓘ
None Custom

Custom default value
WSSC1

4. Create two parameters with the following properties that is used by the **Get-AzureRmVm** activity:

- **Parameter1:**

- Name - VMName
- Type - String
- Mandatory - No

- **Parameter2:**

- Name - resourceGroupName
- Type - String
- Mandatory - No
- Default value - Custom
- Custom default value - <Name of the resource group that contains the virtual machines>

5. Once you add the parameters, click **OK**. You can now view them in the **Input and output blade**. Click **OK** again, and then click **Save** and **Publish** your runbook.

Configure input parameters in Python runbooks

Unlike PowerShell, PowerShell Workflow, and Graphical runbooks, Python runbooks do not take named parameters. All input parameters are parsed as an array of argument values. You access the array by importing the `sys` module into your Python script, and then using the `sys.argv` array. It is important to note that the first element of the array, `sys.argv[0]`, is the name of the script, so the first actual input parameter is `sys.argv[1]`.

For an example of how to use input parameters in a Python runbook, see [My first Python runbook in Azure Automation](#).

Assign values to input parameters in runbooks

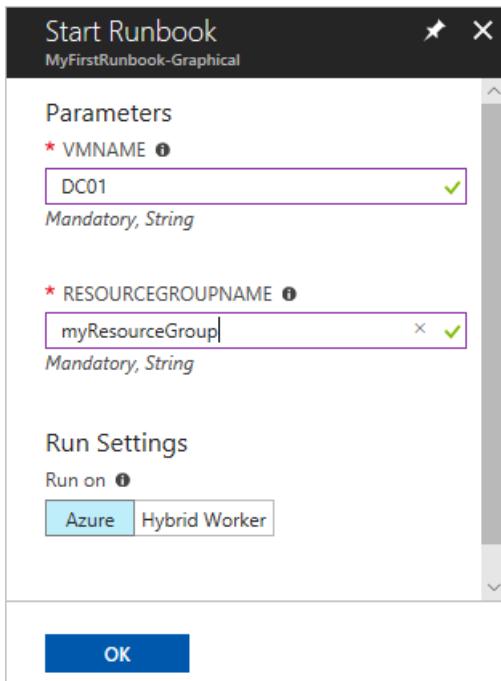
You can pass values to input parameters in runbooks in the following scenarios:

Start a runbook and assign parameters

A runbook can be started many ways: through the Azure portal, with a webhook, with PowerShell cmdlets, with the REST API, or with the SDK. Below we discuss different methods for starting a runbook and assigning parameters.

Start a published runbook by using the Azure portal and assign parameters

When you [start the runbook](#), the **Start Runbook** blade opens and you can enter values for the parameters that you created.



In the label beneath the input box, you can see the attributes that have been set for the parameter. Attributes include mandatory or optional, type, and default value. In the help balloon next to the parameter name, you can see all the key information you need to make decisions about parameter input values. This information includes whether a parameter is mandatory or optional. It also includes the type and default value (if any), and other helpful

notes.

NOTE

String type parameters support **Empty** String values. Entering [**EmptyString**] in the input parameter box passes an empty string to the parameter. Also, String type parameters don't support **Null** values being passed. If you don't pass any value to the String parameter, then PowerShell interprets it as null.

Start a published runbook by using PowerShell cmdlets and assign parameters

- **Azure Resource Manager cmdlets:** You can start an Automation runbook that was created in a resource group by using [Start-AzureRmAutomationRunbook](#).

Example:

```
$params = @{"VMName"="WSVMClassic";"resourceGroupName"="WSVMClassicSG"}  
  
Start-AzureRmAutomationRunbook -AutomationAccountName "TestAutomation" -Name "Get-AzureVMGraphical" -  
ResourceGroupName $resourceGroupName -Parameters $params
```

- **Azure classic deployment model cmdlets:** You can start an automation runbook that was created in a default resource group by using [Start-AzureAutomationRunbook](#).

Example:

```
$params = @{"VMName"="WSVMClassic"; "ServiceName"="WSVMClassicSG"}  
  
Start-AzureAutomationRunbook -AutomationAccountName "TestAutomation" -Name "Get-AzureVMGraphical" -  
Parameters $params
```

NOTE

When you start a runbook by using PowerShell cmdlets, a default parameter, **MicrosoftApplicationManagementStartedBy** is created with the value **PowerShell**. You can view this parameter in the **Job details** blade.

Start a runbook by using an SDK and assign parameters

- **Azure Resource Manager method:** You can start a runbook by using the SDK of a programming language. Below is a C# code snippet for starting a runbook in your Automation account. You can view all the code at our [GitHub repository](#).

```
public Job StartRunbook(string runbookName, IDictionary<string, string> parameters = null)  
{  
    var response = AutomationClient.Jobs.Create(resourceGroupName, automationAccount, new  
JobCreateParameters  
    {  
        Properties = new JobCreateProperties  
        {  
            Runbook = new RunbookAssociationProperty  
            {  
                Name = runbookName  
            },  
            Parameters = parameters  
        }  
    });  
    return response.Job;  
}
```

- **Azure classic deployment model method:** You can start a runbook by using the SDK of a programming language. Below is a C# code snippet for starting a runbook in your Automation account. You can view all the code at our [GitHub repository](#).

```
public Job StartRunbook(string runbookName, IDictionary<string, string> parameters = null)
{
    var response = AutomationClient.Jobs.Create(automationAccount, new JobCreateParameters
    {
        Properties = new JobCreateProperties
        {
            Runbook = new RunbookAssociationProperty
            {
                Name = runbookName
            },
            Parameters = parameters
        }
    });
    return response.Job;
}
```

To start this method, create a dictionary to store the runbook parameters, **VMName** and **resourceGroupName**, and their values. Then start the runbook. Below is the C# code snippet for calling the method that's defined above.

```
IDictionary<string, string> RunbookParameters = new Dictionary<string, string>();

// Add parameters to the dictionary.
RunbookParameters.Add("VMName", "WSVMClassic");
RunbookParameters.Add("resourceGroupName", "WSSC1");

//Call the StartRunbook method with parameters
StartRunbook("Get-AzureVMGraphical", RunbookParameters);
```

Start a runbook by using the REST API and assign parameters

A runbook job can be created and started with the Azure Automation REST API by using the **PUT** method with the following request URI:

```
https://management.core.windows.net/<subscription-id>/cloudServices/<cloud-service-name>/resources/automation/~/automationAccounts/<automation-account-name>/jobs/<job-id>?api-version=2014-12-08`
```

In the request URI, replace the following parameters:

- **subscription-id:** Your Azure subscription ID.
- **cloud-service-name:** The name of the cloud service to which the request should be sent.
- **automation-account-name:** The name of your automation account that's hosted within the specified cloud service.
- **job-id:** The GUID for the job. GUIDs in PowerShell can be created by using the **[GUID]::.NewGuid().ToString()** command.

In order to pass parameters to the runbook job, use the request body. It takes the following two properties provided in JSON format:

- **Runbook name:** Required. The name of the runbook for the job to start.
- **Runbook parameters:** Optional. A dictionary of the parameter list in (name, value) format where name should be of String type and value can be any valid JSON value.

If you want to start the **Get-AzureVMT textual** runbook that was created earlier with **VMName** and

resourceGroupName as parameters, use the following JSON format for the request body.

```
{  
  "properties":{  
    "runbook":{  
      "name":"Get-AzureVMTextual"},  
    "parameters":{  
      "VMName":"WSVMClassic",  
      "resourceGroupName":"WSCS1"}  
  }  
}
```

An HTTP status code 201 is returned if the job is successfully created. For more information on response headers and the response body, see the article about how to [create a runbook job by using the REST API](#).

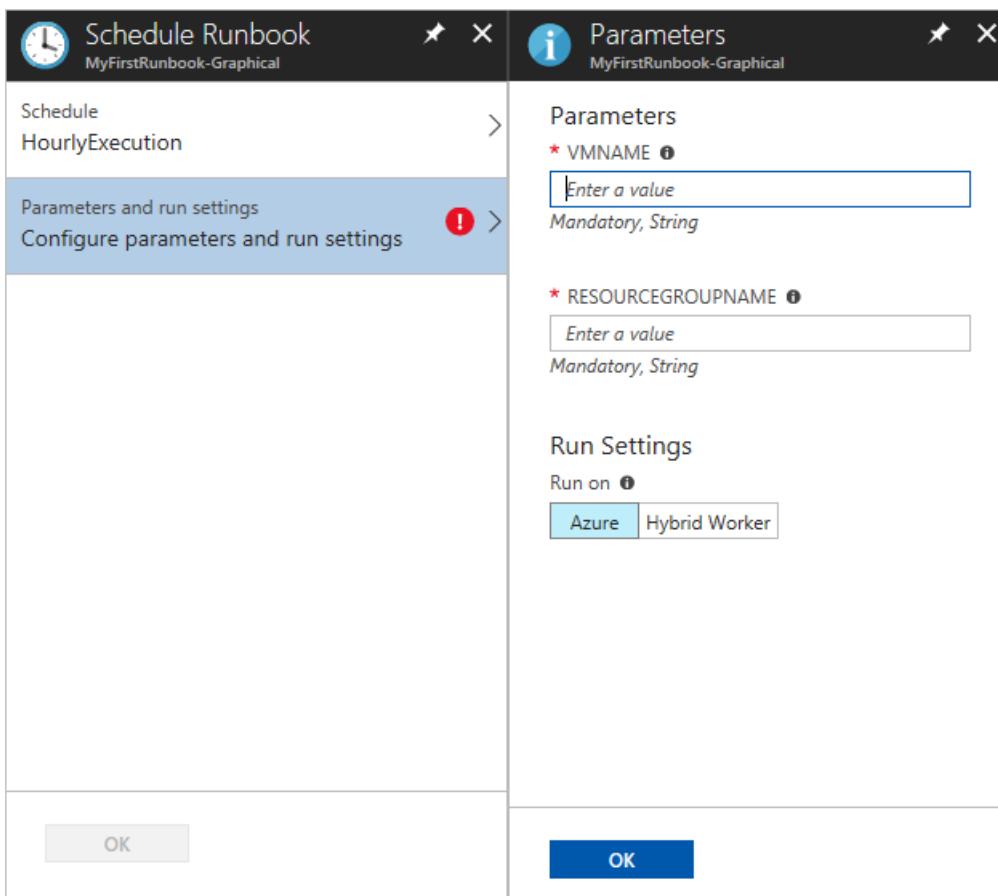
Test a runbook and assign parameters

When you [test the draft version of your runbook](#) by using the test option, the **Test** page opens and you can configure values for the parameters that you created.

The screenshot shows the 'Test' page for a runbook named 'MyFirstRunbook-Graphical'. The top navigation bar includes 'Start', 'Stop', 'Suspend', 'Resume', and 'View last test'. The main area is divided into sections: 'Parameters', 'Run Settings', and 'Activity-level tracing'. In the 'Parameters' section, there are two mandatory string parameters: 'VMNAME' and 'RESOURCEGROUPNAME', each with an 'Enter a value' input field. In the 'Run Settings' section, 'Run on' is set to 'Azure'. In the 'Activity-level tracing' section, 'Trace level' is set to 'None'.

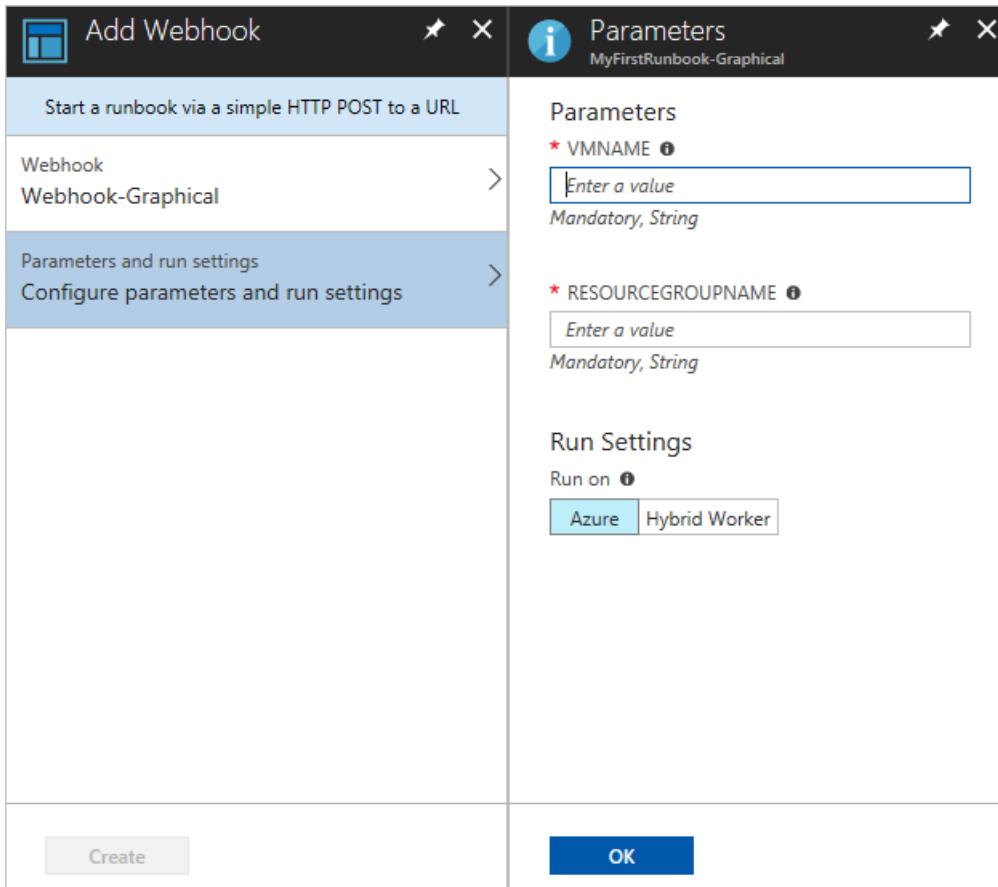
Link a schedule to a runbook and assign parameters

You can [link a schedule](#) to your runbook so that the runbook starts at a specific time. You assign input parameters when you create the schedule, and the runbook uses these values when it is started by the schedule. You can't save the schedule until all mandatory parameter values are provided.



Create a webhook for a runbook and assign parameters

You can create a [webhook](#) for your runbook and configure runbook input parameters. You can't save the webhook until all mandatory parameter values are provided.



When you execute a runbook by using a webhook, the predefined input parameter **Webhookdata** is sent, along with the input parameters that you defined. You can click to expand the **WebhookData** parameter for more details.

The screenshot shows two side-by-side windows. The left window is titled 'MyFirstRunbook-Graphical 1/31/2018 11:57 AM' and displays the 'Job' tab. It includes sections for 'Essentials' (Job Id, Created, Last Update, Run As User, Ran on Azure), 'Overview' (Input: 3, Output: 0, All Logs), and error/warning counts (1 Error, 0 Warnings). The right window is titled 'Input Parameters' and shows a table of parameters with their values. One parameter, 'WEBHOOKDATA', has its value ('{"WebhookName": "Alert1517428631059", "R...}') highlighted with a red border.

NAME	INPUT VALUE
VMNAME	"est"
RESOURCEGROUPNAME	"test"
WEBHOOKDATA	{"WebhookName": "Alert1517428631059", "R..."}

Next steps

- For more information on runbook input and output, see [Azure Automation: runbook input, output, and nested runbooks](#).
- For details about different ways to start a runbook, see [Starting a runbook](#).
- To edit a textual runbook, refer to [Editing textual runbooks](#).
- To edit a graphical runbook, refer to [Graphical authoring in Azure Automation](#).

Error handling in Azure Automation graphical runbooks

5/21/2018 • 3 minutes to read • [Edit Online](#)

A key runbook design principal to consider is identifying different issues that a runbook might experience. These issues can include success, expected error states, and unexpected error conditions.

Runbooks should include error handling. To validate the output of an activity or handle an error with graphical runbooks, you could use a Windows PowerShell code activity, define conditional logic on the output link of the activity, or apply another method.

Often, if there is a non-terminating error that occurs with a runbook activity, any activity that follows is processed regardless of the error. The error is likely to generate an exception, but the next activity is still allowed to run. This is the way that PowerShell is designed to handle errors.

The types of PowerShell errors that can occur during execution are terminating or non-terminating. The differences between terminating and non-terminating errors are as follows:

- **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.

Azure Automation graphical runbooks have been improved with the capability to include error handling. 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.

When to use error handling

Whenever there is a critical activity that throws an error or exception, it's important to prevent the next activity in your runbook from processing and to handle the error appropriately. This is especially critical when your runbooks are supporting a business or service operations process.

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.

In addition, the destination activity can also have outgoing links. These links can be regular links or error links. This means the runbook author can implement complex error-handling logic without resorting to a code activity. The recommended practice is to create a dedicated error-handling runbook with common functionality, but it's not mandatory. Error-handling logic in a PowerShell code activity it isn't the only option.

For 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.

One solution is to have an error link pointing to an activity that handles step one. For example, you could connect the **Write-Warning** cmdlet to an activity for step two, such as the **Start-AzureRmAutomationRunbook** cmdlet.

You could also generalize this behavior for use in many runbooks by putting these two activities in a separate error handling runbook and following the guidance suggested earlier. Before calling this error-handling runbook, you could construct a custom message from the data in the original runbook, and then pass it as a parameter to the error-handling runbook.

How to use error handling

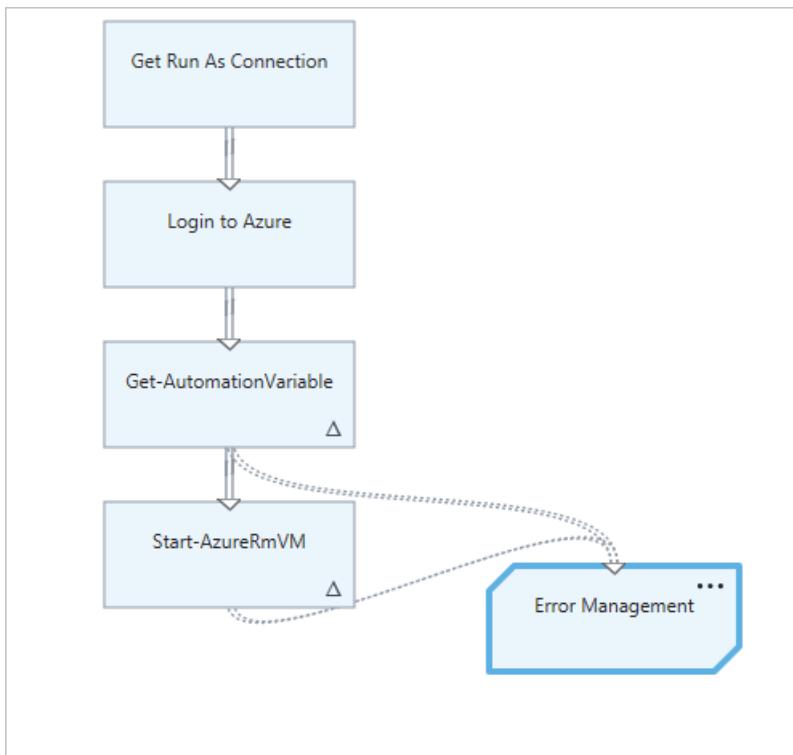
Each activity has a configuration setting that turns exceptions into non-terminating errors. By default, this setting is disabled. We recommend that you enable this setting on any activity where you want to handle errors.

By enabling this configuration, you are assuring that both terminating and non-terminating errors in the activity are handled as non-terminating errors, and can be handled with an error link.

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.



In the following example, a runbook retrieves a variable that contains the computer name of a virtual machine. It then attempts to start the virtual machine with the next activity.



The **Get-AutomationVariable** activity and **Start-AzureRmVm** are configured to convert exceptions to errors. If there are problems getting the variable or starting the VM, then errors are generated.

Name
Get-AutomationVariable

* Label ⓘ
Get-AutomationVariable ✓

Comment

Convert exceptions to errors ⓘ
Yes No

Parameters >
Configure parameter values

Optional additional parameters >
Configure parameters

Retry behavior >
Configure retry behavior

Notices
⚠ Exceptions converted to errors

Error links flow from these activities to a single **error management** activity (a code activity). This activity is configured with a simple PowerShell expression that uses the *Throw* keyword to stop processing, along with `$Error.Exception.Message` to get the message that describes the current exception.

Code Editor



PowerShell code ⓘ

```
Throw "Error occurred with an activity." + " Error returned is: " $Error.Exception.Message
```

OK

Next steps

- To learn more about links and link types in graphical runbooks, see [Graphical authoring in Azure Automation](#).
- To learn more about runbook execution, how to monitor runbook jobs, and other technical details, see [Track a runbook job](#).

Runbook execution in Azure Automation

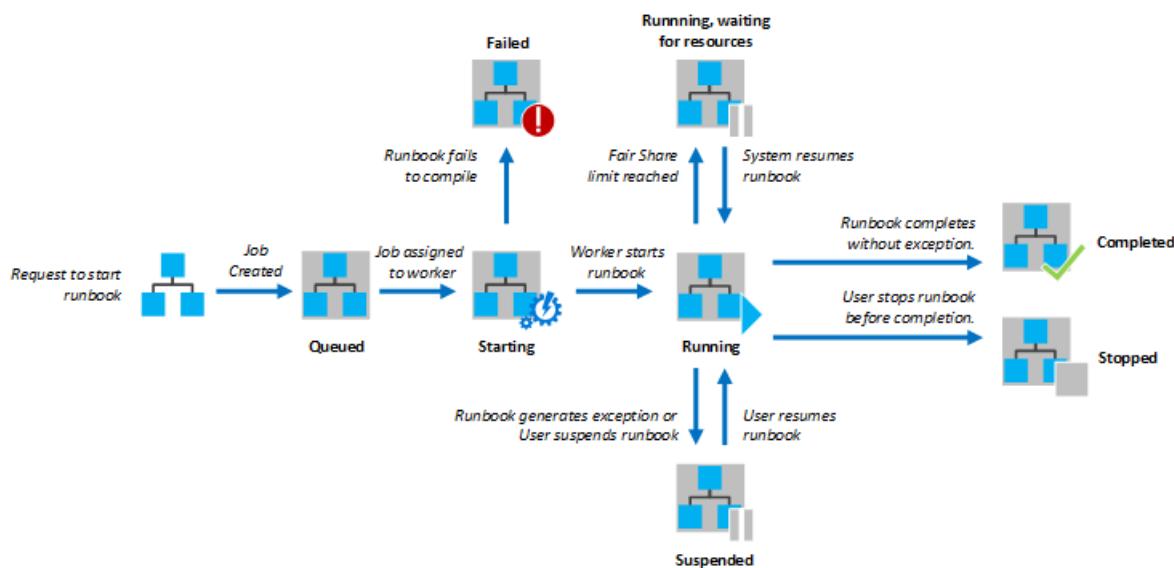
7/9/2018 • 8 minutes to read • [Edit Online](#)

When you start a runbook in Azure Automation, a job is created. A job is a single execution instance of a runbook. An Azure Automation worker is assigned to run each job. While workers are shared by multiple Azure accounts, jobs from different Automation accounts are isolated from one another. You do not have control over which worker services the request for your job. A single runbook can have multiple jobs running at one time. The execution environment for jobs from the same Automation Account may be reused. When you view the list of runbooks in the Azure portal, it lists the status of all jobs that were started for each runbook. You can view the list of jobs for each runbook in order to track the status of each. For a description of the different job statuses [Job Statuses](#).

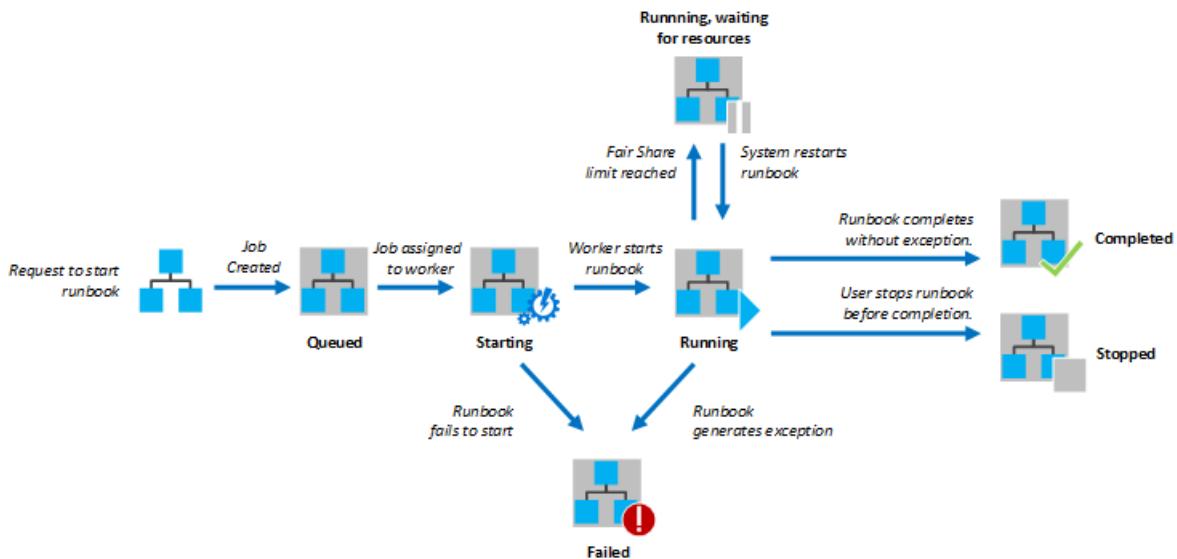
NOTE

If you're interested in viewing or deleting personal data, please see the [Azure Data Subject Requests for the GDPR](#) article. If you're looking for general info about GDPR, see the [GDPR section of the Service Trust portal](#).

The following diagram shows the lifecycle of a runbook job for [Graphical runbooks](#) and [PowerShell Workflow runbooks](#).



The following diagram shows the lifecycle of a runbook job for [PowerShell runbooks](#).



Your jobs have access to your Azure resources by making a connection to your Azure subscription. They only have access to resources in your data center if those resources are accessible from the public cloud.

Job statuses

The following table describes the different statuses that are possible for a job.

STATUS	DESCRIPTION
Completed	The job completed successfully.
Failed	For Graphical and PowerShell Workflow runbooks , the runbook failed to compile. For PowerShell Script runbooks , the runbook failed to start or the job encountered an exception.
Failed, waiting for resources	The job failed because it reached the fair share limit three times and started from the same checkpoint or from the start of the runbook each time.
Queued	The job is waiting for resources on an Automation worker to come available so that it can be started.
Starting	The job has been assigned to a worker, and the system is in the process of starting it.
Resuming	The system is in the process of resuming the job after it was suspended.
Running	The job is running.
Running, waiting for resources	The job has been unloaded because it reached the fair share limit. It resumes shortly from its last checkpoint.
Stopped	The job was stopped by the user before it was completed.
Stopping	The system is in the process of stopping the job.

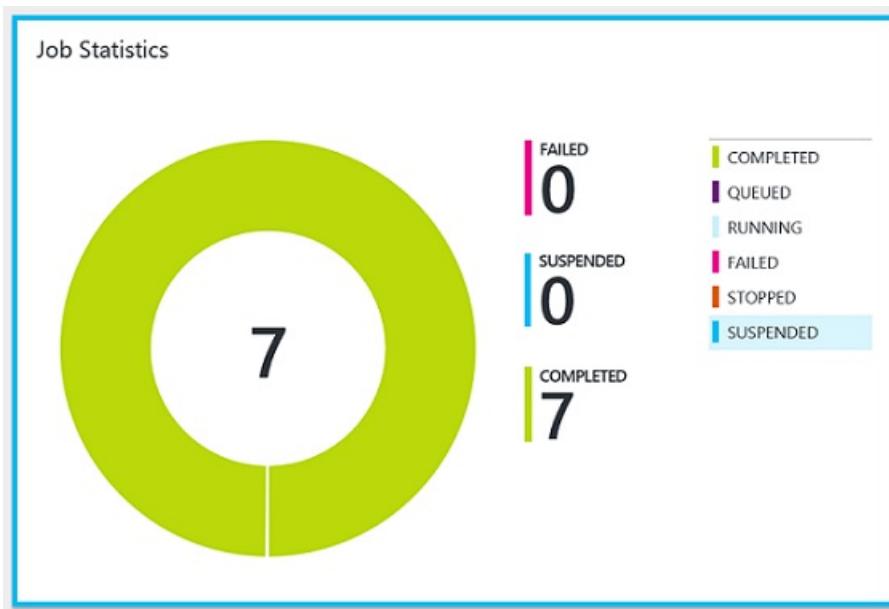
Status	Description
Suspended	<p>The job was suspended by the user, by the system, or by a command in the runbook. A job that is suspended can be started again and resume from its last checkpoint or from the beginning of the runbook if it doesn't have checkpoints. The runbook is only be suspended by the system when an exception occurs. By default, ErrorActionPreference is set to Continue, meaning that the job keeps running on an error. If this preference variable is set to Stop, then the job suspends on an error. Applies to Graphical and PowerShell Workflow runbooks only.</p>
Suspending	<p>The system is attempting to suspend the job at the request of the user. The runbook must reach its next checkpoint before it can be suspended. If it already passed its last checkpoint, then it completes before it can be suspended. Applies to Graphical and PowerShell Workflow runbooks only.</p>

Viewing job status from the Azure portal

You can view a summarized status of all runbook jobs or drill into details of a specific runbook job in the Azure portal or by configuring integration with your Log Analytics workspace to forward runbook job status and job streams. For more information about integrating with Log Analytics, see [Forward job status and job streams from Automation to Log Analytics](#).

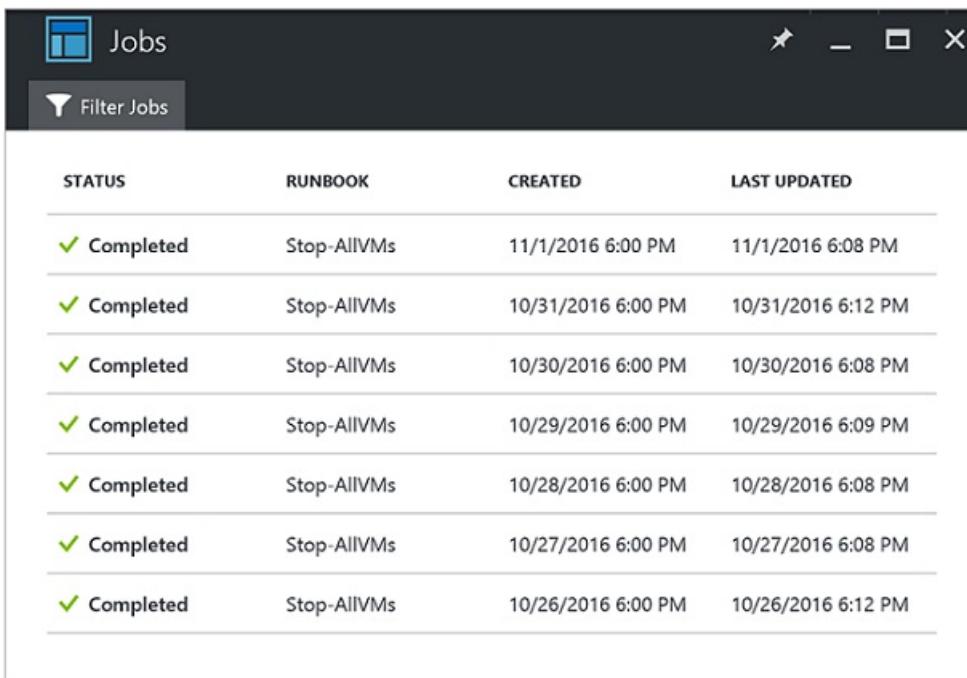
Automation runbook jobs summary

On the right of your selected Automation account, you can see a summary of all of the runbook jobs for a selected Automation account under **Job Statistics** tile.



This tile displays a count and graphical representation of the job status for all jobs executed.

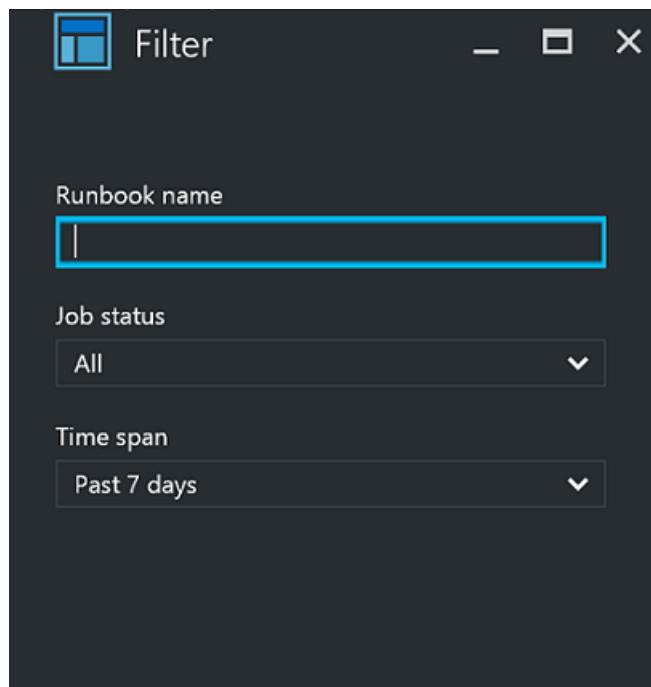
Clicking on the tile presents the **Jobs** blade, which includes a summarized list of all jobs executed, with status, job execution, and start and completion times.



The screenshot shows the 'Jobs' blade in the Azure portal. At the top, there's a 'Filter Jobs' button. Below it is a table with four columns: STATUS, RUNBOOK, CREATED, and LAST UPDATED. All seven listed jobs are in the 'Completed' status.

STATUS	RUNBOOK	CREATED	LAST UPDATED
✓ Completed	Stop-AllVMs	11/1/2016 6:00 PM	11/1/2016 6:08 PM
✓ Completed	Stop-AllVMs	10/31/2016 6:00 PM	10/31/2016 6:12 PM
✓ Completed	Stop-AllVMs	10/30/2016 6:00 PM	10/30/2016 6:08 PM
✓ Completed	Stop-AllVMs	10/29/2016 6:00 PM	10/29/2016 6:09 PM
✓ Completed	Stop-AllVMs	10/28/2016 6:00 PM	10/28/2016 6:08 PM
✓ Completed	Stop-AllVMs	10/27/2016 6:00 PM	10/27/2016 6:08 PM
✓ Completed	Stop-AllVMs	10/26/2016 6:00 PM	10/26/2016 6:12 PM

You can filter the list of jobs by selecting **Filter jobs** and filter on a specific runbook, job status, or from the drop-down list, the date/time range to search within.



Alternatively, you can view job summary details for a specific runbook by selecting that runbook from the **Runbooks** blade in your Automation account, and then select the **Jobs** tile. This presents the **Jobs** blade, and from there you can click on the job record to view its detail and output.

The screenshot shows the Azure portal interface for a runbook job. At the top, there's a navigation bar with icons for Stop, Suspend, and View source. Below that is an 'Overview' section with a 'Job Summary' card. The job ID is edf36252-6cc9-4c59-92d7-dcaeb2285cac, created on 11/1/2016 at 6:00 PM, and last updated on 11/1/2016 at 6:08 PM. It ran on Azure and is marked as 'Completed' with a green checkmark icon. To the right of the summary is a 'Runbook' card showing 0 INPUT and 0 Output. Below the summary is a 'Status' section with 'Errors' (1) and 'Warnings' (0). A link to 'All Logs' is also present. At the bottom is an 'Exception' section stating 'None'.

Job Summary

You can view a list of all of the jobs that have been created for a particular runbook and their most recent status. You can filter this list by job status and the range of dates for the last change to the job. To view its detailed information and output, click on the name of a job. The detailed view of the job includes the values for the runbook parameters that were provided to that job.

You can use the following steps to view the jobs for a runbook.

1. In the Azure portal, select **Automation** and then select the name of an Automation account.
2. From the hub, select **Runbooks** and then on the **Runbooks** blade select a runbook from the list.
3. On the blade for the selected runbook, click the **Jobs** tile.
4. Click on one of the jobs in the list and on the runbook job details blade you can view its detail and output.

Retrieving job status using Windows PowerShell

You can use the [Get-AzureRmAutomationJob](#) to retrieve the jobs created for a runbook and the details of a particular job. If you start a runbook with Windows PowerShell using [Start-AzureRmAutomationRunbook](#), then it returns the resulting job. Use [Get-AzureRmAutomationJobOutput](#) to get a job's output.

The following sample commands retrieve the last job for a sample runbook and display its status, the values provided for the runbook parameters, and the output from the job.

```
$job = (Get-AzureRmAutomationJob -AutomationAccountName "MyAutomationAccount" `  
-RunbookName "Test-Runbook" -ResourceGroupName "ResourceGroup01" | sort LastModifiedDate -desc)[0]  
$job.Status  
$job.JobParameters  
Get-AzureRmAutomationJobOutput -ResourceGroupName "ResourceGroup01" `  
-AutomationAccountName "MyAutomationAcct" -Id $job.JobId -Stream Output
```

The following sample retrieves the output for a specific job, and returns each record. In the case that there was an exception for one of the records, the exception is written out instead of the value. This is useful as exceptions can provide additional information which may not be logged normally during output.

```

$output = Get-AzureRmAutomationJobOutput -AutomationAccountName <AutomationAccountName> -Id <jobID> -
ResourceGroupName <ResourceGroupName> -Stream "Any"
foreach($item in $output)
{
    $fullRecord = Get-AzureRmAutomationJobOutputRecord -AutomationAccountName <AutomationAccountName> -
ResourceGroupName <ResourceGroupName> -JobId <jobID> -Id $item.StreamRecordId
    if ($fullRecord.Type -eq "Error")
    {
        $fullRecord.Value.Exception
    }
    else
    {
        $fullRecord.Value
    }
}

```

Get details from Activity log

Other details such as the person or account that started the runbook can be retrieved from the Activity log for the automation account. The following PowerShell example provides the last user to run the runbook in question:

```

$SubID = "00000000-0000-0000-0000-000000000000"
$rg = "ResourceGroup01"
$AutomationAccount = "MyAutomationAccount"
$RunbookName = "Test-Runbook"
$JobResourceID =
"/subscriptions/$subid/resourcegroups/$rg/providers/Microsoft.Automation/automationAccounts/$AutomationAccou
nt/jobs"

Get-AzureRmLog -ResourceId $JobResourceID -MaxRecord 1 | Select Caller

```

Fair share

In order to share resources among all runbooks in the cloud, Azure Automation will temporarily unload any job after it has been running for three hours. During this time, jobs for [PowerShell-based runbooks](#) are stopped and are not be restarted. The job status shows **Stopped**. This type of runbook is always restarted from the beginning since they don't support checkpoints.

[PowerShell-Workflow-based runbooks](#) are resumed from their last [checkpoint](#). After running three hours, the runbook job is suspended by the service and its status shows **Running, waiting for resources**. When a sandbox becomes available, the runbook is automatically restarted by the Automation service and resumes from the last checkpoint. This is normal PowerShell-Workflow behavior for suspend/restart. If the runbook again exceeds three hours of runtime, the process repeats, up to three times. After the third restart, if the runbook still has not completed in three hours, then the runbook job is failed, and the job status shows **Failed, waiting for resources**. In this case, you receive the following exception with the failure.

The job cannot continue running because it was repeatedly evicted from the same checkpoint. Please make sure your Runbook does not perform lengthy operations without persisting its state.

This is to protect the service from runbooks running indefinitely without completing, as they are not able to make it to the next checkpoint without being unloaded again.

If the runbook has no checkpoints or the job had not reached the first checkpoint before being unloaded, then it restarts from the beginning.

For long running tasks, it is recommended to use a [Hybrid Runbook Worker](#). Hybrid Runbook Workers are not limited by fair share, and do not have have a limitation on how long a runbook can execute.

If you are using a PowerShell Workflow runbook on Azure, when you create a runbook, you should ensure that the time to run any activities between two checkpoints does not exceed three hours. You may need to add checkpoints to your runbook to ensure that it does not reach this three-hour limit or break up long running operations. For example, your runbook might perform a reindex on a large SQL database. If this single operation does not complete within the fair share limit, then the job is unloaded and restarted from the beginning. In this case, you should break up the reindex operation into multiple steps, such as reindexing one table at a time, and then insert a checkpoint after each operation so that the job could resume after the last operation to complete.

Next steps

- To learn more about the different methods that can be used to start a runbook in Azure Automation, see [Starting a runbook in Azure Automation](#)

Runbook settings

5/21/2018 • 2 minutes to read • [Edit Online](#)

Each runbook in Azure Automation has multiple settings that help it to be identified and to change its logging behavior. Each of these settings is described below followed by procedures on how to modify them.

Settings

Name and 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. For example, when you submit a runbook to the [PowerShell Gallery](#), you specify particular tags to identify which categories the runbook should be listed in. You can specify multiple tags for a runbook by separating them with commas.

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. For more information on these records, see [Runbook Output and Messages](#).

Changing runbook settings

Changing runbook settings with the Azure portal

You can change settings for a runbook in the Azure portal from the **Settings** blade for the runbook.

1. In the Azure portal, select **Automation** and then click the name of an automation account.
2. Select the **Runbooks** tab.
3. Click the name of a runbook and you are directed to the settings blade for the runbook. From here you can specify or modify tags, the runbook description, configure logging and tracing settings, and access support tools to help you solve problems.

Changing runbook settings with Windows PowerShell

You can use the `Set-AzureRmAutomationRunbook` cmdlet to change the settings for a runbook. If you want to specify multiple tags, you can either provide an array or a single string with comma delimited values to the Tags parameter. You can get the current tags with the `Get-AzureRmAutomationRunbook`.

The following sample commands show how to set the properties for a runbook. This sample adds three tags to the existing tags and specifies that verbose records should be logged.

```
$automationAccountName = "MyAutomationAccount"
$runbookName = "Sample-TestRunbook"
$tags = (Get-AzureRmAutomationRunbook -ResourceGroupName "ResourceGroup01" ` 
-AutomationAccountName $automationAccountName -Name $runbookName).Tags
$tags += "Tag1,Tag2,Tag3"
Set-AzureRmAutomationRunbook -ResourceGroupName "ResourceGroup01" ` 
-AutomationAccountName $automationAccountName -Name $runbookName -LogVerbose $true -Tags $tags
```

Next steps

- To learn how to create and retrieve output and error messages from runbooks, see [Runbook Output and Messages](#)
- To understand how to add a runbook that was already developed by the community or other source, or to create your own runbook see [Creating or Importing a Runbook](#)

Managing Azure Automation data

5/21/2018 • 3 minutes to read • [Edit Online](#)

This article contains multiple topics for managing an Azure Automation environment.

Data retention

When you delete a resource in Azure Automation, it is retained for 90 days for auditing purposes before being removed permanently. You can't see or use the resource during this time. This policy also applies to resources that belong to an automation account that is deleted.

Azure Automation automatically deletes and permanently removes jobs older than 90 days.

The following table summarizes the retention policy for different resources.

DATA	POLICY
Accounts	Permanently removed 90 days after the account is deleted by a user.
Assets	Permanently removed 90 days after the asset is deleted by a user, or 90 days after the account that holds the asset is deleted by a user.
Modules	Permanently removed 90 days after the module is deleted by a user, or 90 days after the account that holds the module is deleted by a user.
Runbooks	Permanently removed 90 days after the resource is deleted by a user, or 90 days after the account that holds the resource is deleted by a user.
Jobs	Deleted and permanently removed 90 days after last being modified. This could be after the job completes, is stopped, or is suspended.
Node Configurations/MOF Files	Old node configuration is permanently removed 90 days after a new node configuration is generated.
DSC Nodes	Permanently removed 90 days after the node is unregistered from Automation Account using Azure portal or the Unregister-AzureRMAutomationDscNode cmdlet in Windows PowerShell. Nodes are also permanently removed 90 days after the account that holds the node is deleted by a user.
Node Reports	Permanently removed 90 days after a new report is generated for that node

The retention policy applies to all users and currently cannot be customized.

However, if you need to retain data for a longer period of time, you can forward runbook job logs to Log Analytics. For further information, review [forward Azure Automation job data to Log Analytics](#).

Backing up Azure Automation

When you delete an automation account in Microsoft Azure, all objects in the account are deleted including runbooks, modules, configurations, settings, jobs, and assets. The objects cannot be recovered after the account is deleted. You can use the following information to backup the contents of your automation account before deleting it.

Runbooks

You can export your runbooks to script files using either the Azure portal or the [Get-AzureAutomationRunbookDefinition](#) cmdlet in Windows PowerShell. These script files can be imported into another automation account as discussed in [Creating or Importing a Runbook](#).

Integration modules

You cannot export integration modules from Azure Automation. You must ensure that they are available outside of the automation account.

Assets

You cannot export [assets](#) from Azure Automation. Using the Azure portal, you must note the details of variables, credentials, certificates, connections, and schedules. You must then manually create any assets that are used by runbooks that you import into another automation.

You can use [Azure cmdlets](#) to retrieve details of unencrypted assets and either save them for future reference or create equivalent assets in another automation account.

You cannot retrieve the value for encrypted variables or the password field of credentials using cmdlets. If you don't know these values, then you can retrieve them from a runbook using the [Get-AutomationVariable](#) and [Get-AutomationPSCredential](#) activities.

You cannot export certificates from Azure Automation. You must ensure that any certificates are available outside of Azure.

DSC configurations

You can export your configurations to script files using either the Azure portal or the [Export-AzureRmAutomationDscConfiguration](#) cmdlet in Windows PowerShell. These configurations can be imported and used in another automation account.

Geo-replication in Azure Automation

Geo-replication, standard in Azure Automation accounts, backs up account data to a different geographical region for redundancy. You can choose a primary region when setting up your account, and then a secondary region is assigned to it automatically. The secondary data, copied from the primary region, is continuously updated in case of data loss.

The following table shows the available primary and secondary region pairings.

PRIMARY	SECONDARY
South Central US	North Central US
US East 2	Central US
West Europe	North Europe
South East Asia	East Asia

PRIMARY	SECONDARY
Japan East	Japan West

In the unlikely event that a primary region data is lost, Microsoft attempts to recover it. If the primary data cannot be recovered, then geo-failover is performed and the affected customers will be notified about this through their subscription.

Call an Azure Automation runbook from a Log Analytics alert

6/20/2018 • 4 minutes to read • [Edit Online](#)

You can configure an alert in Azure Log Analytics to create an alert record when results match your criteria. That alert can then automatically run an Azure Automation runbook in an attempt to auto-remediate the issue.

For example, an alert might indicate a prolonged spike in processor utilization. Or it might indicate when an application process that's critical to the functionality of a business application fails. A runbook can then write a corresponding event in the Windows event log.

There are two options to call a runbook in the alert configuration:

- Use a webhook.
 - This is the only option available if your Log Analytics workspace is not linked to an Automation account.
 - If you already have an Automation account linked to a Log Analytics workspace, this option is still available.
- Select a runbook directly.
 - This option is available only if the Log Analytics workspace is linked to an Automation account.

Calling a runbook by using a webhook

You can use a webhook to start a particular runbook in Azure Automation through a single HTTP request. Before you configure the [Webhook action for log alerts](#) to call the runbook by using a webhook as an alert action, you need to [create a webhook](#) for the runbook that's called through this method. Remember to record the webhook URL so you can reference it while configuring the alert rule.

Calling a runbook directly

You can install and configure the Automation and Control offering in your Log Analytics workspace. While you're configuring the runbook actions option for the alert, you can view all runbooks from the **Select a runbook** dropdown list and select the specific runbook that you want to run in response to the alert. The selected runbook can run in an Azure workspace or on a hybrid runbook worker.

After you create the alert by using the runbook option, a webhook is created for the runbook. You can see the webhook if you go to the Automation account and open the webhook pane of the selected runbook.

If you delete the alert, the webhook is not deleted. This is not a problem. The webhook just becomes an orphaned item that you should eventually delete manually, to maintain an organized Automation account.

Characteristics of a runbook

Both methods for calling the runbook from the Log Analytics alert have characteristics that you need to understand before you configure your alert rules.

The alert data is in JSON format in a single property called **SearchResult**. This format is for runbook and webhook actions with a standard payload. For webhook actions with custom payloads (including **IncludeSearchResults:True** in **RequestBody**), the property is **SearchResults**.

You must have a runbook input parameter called **WebhookData** that is an **Object** type. It can be mandatory or

optional. The alert passes the search results to the runbook by using this input parameter.

```
param
(
    [Parameter (Mandatory=$true)]
    [object] $WebhookData
)
```

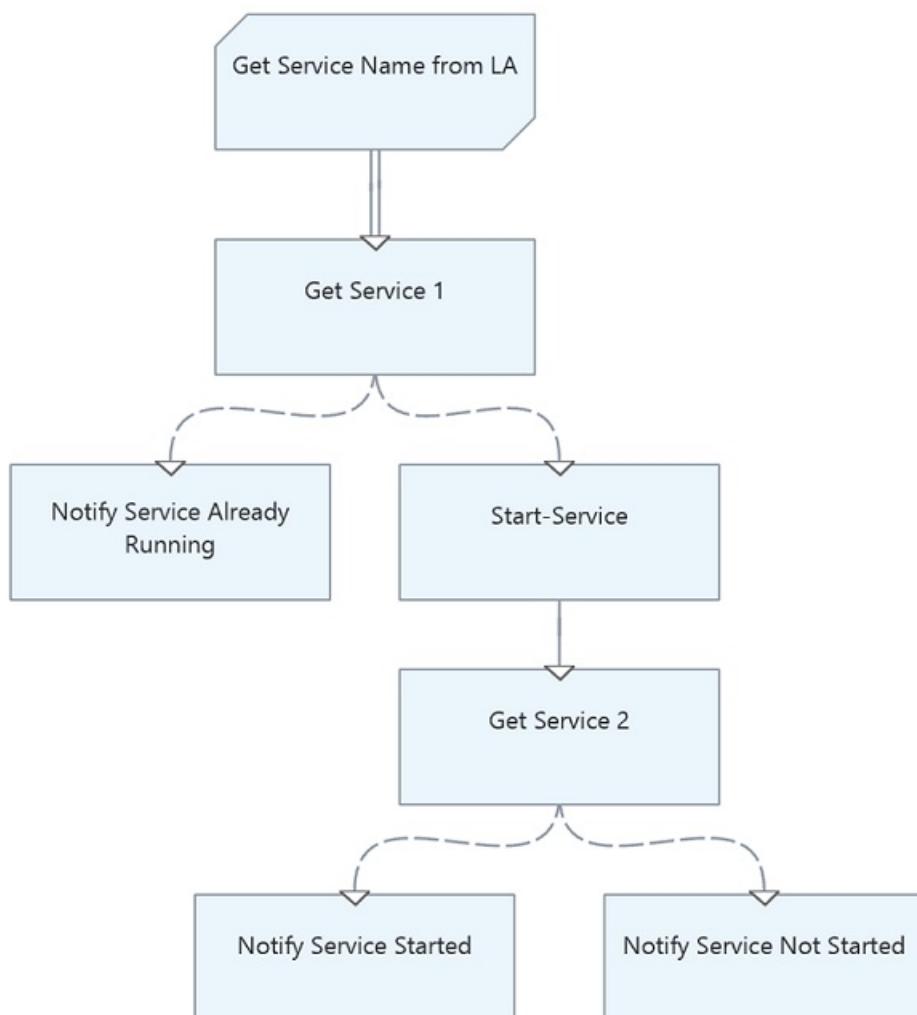
You must also have code to convert **WebhookData** to a PowerShell object.

```
$SearchResult = (ConvertFrom-Json $WebhookData.RequestBody).SearchResult.value
```

\$SearchResult is an array of objects. Each object contains the fields with values from one search result.

Example walkthrough

The following example of a graphical runbook demonstrates how this process works. It starts a Windows service.



The runbook has one input parameter of type **Object** that is called **WebhookData**. It includes the webhook data passed from the alert that contains **SearchResult**.

* Name !
WebhookData

Description !

Type !
Object

Mandatory !
 Yes No

Default value
Mandatory parameters can't have a default value

For this example, we created two custom fields in Log Analytics: **SvcDisplayName_CF** and **SvcState_CF**. These fields extract the service display name and the state of the service (that is, running or stopped) from the event that's written to the system event log. We then created an alert rule with the following search query, so that we can detect when the Print Spooler service is stopped on the Windows system:

```
Type=Event SvcDisplayName_CF="Print Spooler" SvcState_CF="stopped"
```

It can be any service of interest. For this example, we're referencing one of the pre-existing services that are included with the Windows OS. The alert action is configured to execute the runbook used in this example and run on the hybrid runbook worker, which is enabled on the target system.

The runbook code activity **Get Service Name from LA** converts the JSON-formatted string into an object type and filters on the item **SvcDisplayName_CF**. It extracts the display name of the Windows service and passes this value to the next activity, which verifies that the service is stopped before attempting to restart it.

SvcDisplayName_CF is a [custom field](#) that we created in Log Analytics to demonstrate this example.

```
$SearchResult = (ConvertFrom-Json $WebhookData.RequestBody).SearchResult.value
$SearchResult.SvcDisplayName_CF
```

When the service stops, the alert rule in Log Analytics detects a match, triggers the runbook, and sends the alert context to the runbook. The runbook tries to verify that the service is stopped. If so, the runbook attempts to restart the service, verify that it started correctly, and display the results.

Alternatively, if you don't have your Automation account linked to your Log Analytics workspace, you can configure the alert rule with a webhook action. The webhook action triggers the runbook. It also configures the runbook to convert the JSON-formatted string and filter on **SearchResult** by following the guidance mentioned earlier.

NOTE

If your workspace has been upgraded to the [new Log Analytics query language](#), the webhook payload has changed. Details of the format are in the [Azure Log Analytics REST API](#).

Next steps

- To learn more about creating an Azure Alert using a log search, see [Log alerts in Azure](#).
- To understand how to trigger runbooks by using a webhook, see [Azure Automation webhooks](#).

Pass a JSON object to an Azure Automation runbook

7/3/2018 • 2 minutes to read • [Edit Online](#)

It can be useful to store data that you want to pass to a runbook in a JSON file. For example, you might create a JSON file that contains all of the parameters you want to pass to a runbook. To do this, you have to convert the JSON to a string and then convert the string to a PowerShell object before passing its contents to the runbook.

In this example, we'll create a PowerShell script that calls [Start-AzureRmAutomationRunbook](#) to start a PowerShell runbook, passing the contents of the JSON to the runbook. The PowerShell runbook starts an Azure VM, getting the parameters for the VM from the JSON that was passed in.

Prerequisites

To complete this tutorial, you need the following:

- Azure subscription. If you don't have one yet, you can [activate your MSDN subscriber benefits](#) or [\[sign up for a free account\]\(https://azure.microsoft.com/free/\)](#).
- [Automation account](#) to hold the runbook and authenticate to Azure resources. This account must have permission to start and stop the virtual machine.
- An Azure virtual machine. We stop and start this machine so it should not be a production VM.
- Azure Powershell installed on a local machine. See [Install and configure Azure Powershell](#) for information about how to get Azure PowerShell.

Create the JSON file

Type the following test in a text file, and save it as `test.json` somewhere on your local computer.

```
{  
    "VmName" : "TestVM",  
    "ResourceGroup" : "AzureAutomationTest"  
}
```

Create the runbook

Create a new PowerShell runbook named "Test-Json" in Azure Automation. To learn how to create a new PowerShell runbook, see [My first PowerShell runbook](#).

To accept the JSON data, the runbook must take an object as an input parameter.

The runbook can then use the properties defined in the JSON.

```

Param(
    [parameter(Mandatory=$true)]
    [object]$json
)

# Connect to Azure account
$Conn = Get-AutomationConnection -Name AzureRunAsConnection
Connect-AzureRmAccount -ServicePrincipal -Tenant $Conn.TenantID ` 
    -ApplicationID $Conn.ApplicationID -CertificateThumbprint $Conn.CertificateThumbprint

# Convert object to actual JSON
$json = $json | ConvertFrom-Json

# Use the values from the JSON object as the parameters for your command
Start-AzureRmVM -Name $json.VMName -ResourceGroupName $json.ResourceGroup

```

Save and publish this runbook in your Automation account.

Call the runbook from PowerShell

Now you can call the runbook from your local machine by using Azure PowerShell. Run the following PowerShell commands:

1. Log in to Azure:

```
Connect-AzureRmAccount
```

You are prompted to enter your Azure credentials.

IMPORTANT

Add-AzureRmAccount is now an alias for **Connect-AzureRMAccount**. When searching your library items, if you do not see **Connect-AzureRMAccount**, you can use **Add-AzureRmAccount**, or you can update your modules in your Automation Account.

2. Get the contents of the JSON file and convert it to a string:

```
$json = (Get-content -path 'JsonPath\test.json' -Raw) | Out-string
```

JsonPath is the path where you saved the JSON file.

3. Convert the string contents of **\$json** to a PowerShell object: `powershell $JsonParams = @{"json"=$json}`

4. Create a hashtable for the parameters for `Start-AzureRmAutomationRunbook`:

```
powershell $RBParams = @{ AutomationAccountName = 'AATest' ResourceGroupName = 'RGTest' Name = 'Test-Json' Parameters = $JsonParams }
```

Notice that you are setting the value of **Parameters** to the PowerShell object that contains the values from the JSON file.

5. Start the runbook `powershell $job = Start-AzureRmAutomationRunbook @RBParams`

The runbook uses the values from the JSON file to start a VM.

Next steps

- To learn more about editing PowerShell and PowerShell Workflow runbooks with a textual editor, see [Editing textual runbooks in Azure Automation](#)

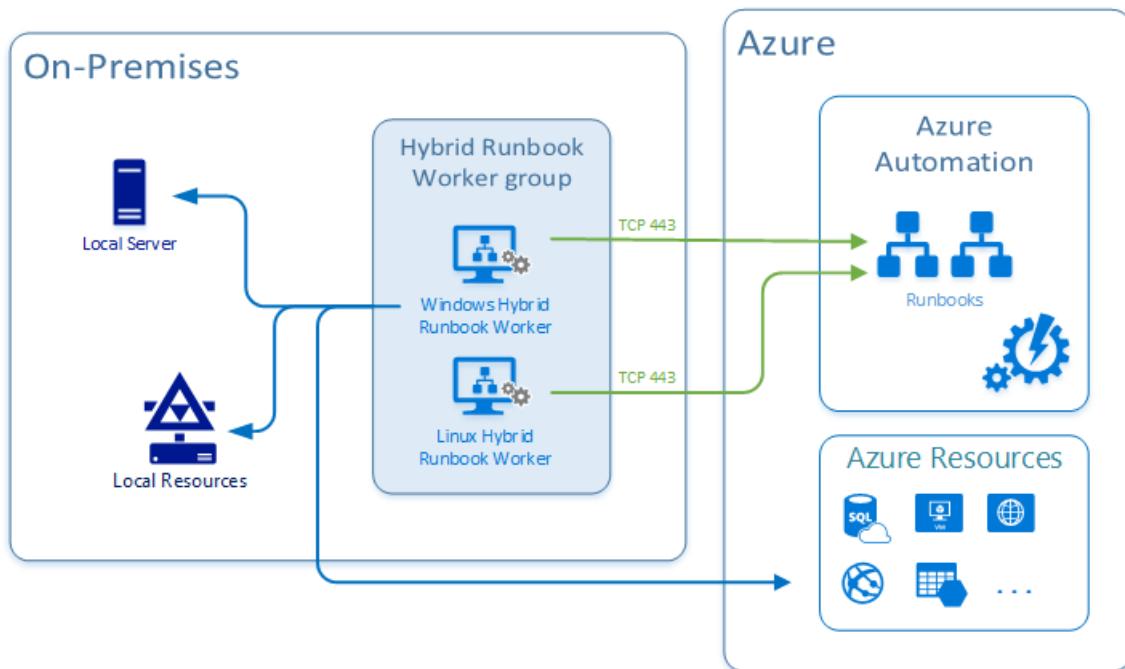
- To learn more about creating and importing runbooks, see [Creating or importing a runbook in Azure Automation](#)

Automate resources in your datacenter or cloud by using Hybrid Runbook Worker

7/10/2018 • 6 minutes to read • [Edit Online](#)

Runbooks in Azure Automation might not be able to access resources in other clouds or in your on-premises environment because they run on the Azure cloud platform. You can use the Hybrid Runbook Worker feature of Azure Automation to run runbooks directly on the computer that's hosting the role and against resources in the environment to manage those local resources. Runbooks are stored and managed in Azure Automation and then delivered to one or more designated computers.

The following image illustrates this functionality:



Each Hybrid Runbook Worker is a member of a Hybrid Runbook Worker group that you specify when you install the agent. A group can include a single agent, but you can install multiple agents in a group for high availability.

When you start a runbook on a Hybrid Runbook Worker, you specify the group that it runs on. Each worker in the group polls Azure Automation to see if any jobs are available. If a job is available, the first worker to get the job takes it. You can't specify a particular worker.

Install a Hybrid Runbook Worker

The process to install a Hybrid Runbook Worker depends on the OS. The following table contains links to the methods that you can use for the installation.

To install and configure a Windows Hybrid Runbook Worker, you can use two methods. The recommended method is using an Automation runbook to completely automate the process of configuring a Windows computer. The second method is following a step-by-step procedure to manually install and configure the role. For Linux machines, you run a Python script to install the agent on the machine.

OS

DEPLOYMENT TYPES

OS	Deployment Types
Windows	PowerShell Manual
Linux	Python

NOTE

To manage the configuration of your servers that support the Hybrid Runbook Worker role with Desired State Configuration (DSC), you need to add them as DSC nodes. For more information about onboarding them for management with DSC, see [Onboarding machines for management by Azure Automation DSC](#).

If you enable the [Update Management solution](#), any computer that's connected to your Azure Log Analytics workspace is automatically configured as a Hybrid Runbook Worker to support runbooks included in this solution. However, the computer is not registered with any Hybrid Worker groups already defined in your Automation account. The computer can be added to a Hybrid Runbook Worker group in your Automation account to support Automation runbooks as long as you're using the same account for both the solution and the Hybrid Runbook Worker group membership. This functionality has been added to version 7.2.12024.0 of Hybrid Runbook Worker.

Review the [information for planning your network](#) before you begin deploying a Hybrid Runbook Worker. After you successfully deploy the worker, review [Run runbooks on a Hybrid Runbook Worker](#) to learn how to configure your runbooks to automate processes in your on-premises datacenter or other cloud environment.

Remove a Hybrid Runbook Worker

You can remove one or more Hybrid Runbook Workers from a group, or you can remove the group, depending on your requirements. To remove a Hybrid Runbook Worker from an on-premises computer, perform the following steps:

1. In the Azure portal, go to your Automation account.
2. Under **Settings**, select **Keys** and note the values for **URL** and **Primary Access Key**. You need this information for the next step.

Windows

Open a PowerShell session in Administrator mode and run the following command. Use the **-Verbose** switch for a detailed log of the removal process.

```
Remove-HybridRunbookWorker -url <URL> -key <PrimaryAccessKey>
```

To remove stale machines from your Hybrid Worker group, use the optional `<machineName>` parameter.

```
Remove-HybridRunbookWorker -url <URL> -key <PrimaryAccessKey> -machineName <ComputerName>
```

Linux

```
sudo python onboarding.py --deregister --endpoint=<URL> --key=<PrimaryAccessKey> --groupname="Example" --workspaceid=<workspaceId>"
```

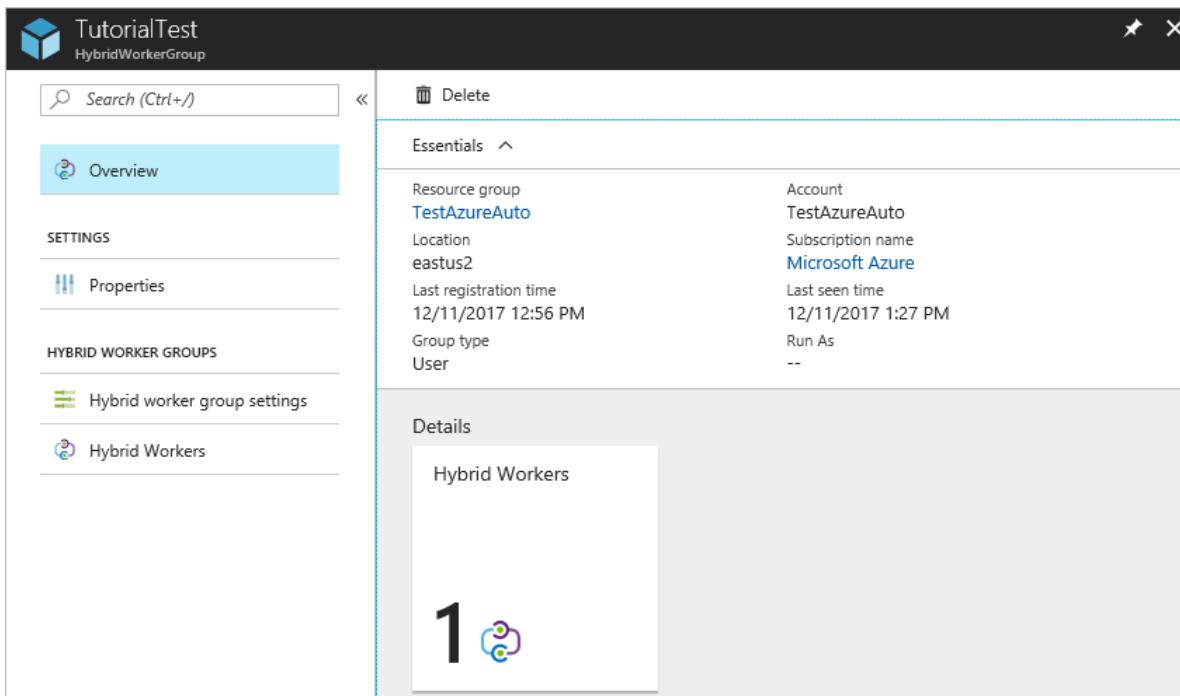
NOTE

This code does not remove the Microsoft Monitoring Agent from the computer, only the functionality and configuration of the Hybrid Runbook Worker role.

Remove a Hybrid Worker group

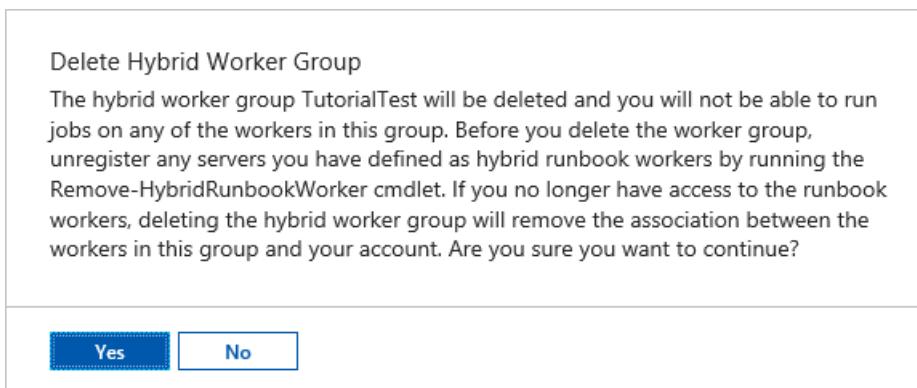
To remove a group, you first need to remove the Hybrid Runbook Worker from every computer that is a member of the group by using the procedure shown earlier. Then, perform the following steps to remove the group:

1. Open the Automation account in the Azure portal.
2. Under **Process Automation**, select **Hybrid worker groups**. Select the group that you want to delete. The properties page for that group appears.



The screenshot shows the Azure portal interface for managing a Hybrid Worker group. The title bar says 'TutorialTest HybridWorkerGroup'. The left sidebar has sections for 'SETTINGS' (Properties) and 'HYBRID WORKER GROUPS' (Hybrid worker group settings, Hybrid Workers). The main content area is titled 'Overview'. It shows the 'Essentials' section with details: Resource group - TestAzureAuto, Location - eastus2, Last registration time - 12/11/2017 12:56 PM, Group type - User. It also shows the 'Details' section with 'Hybrid Workers' and a count of 1, accompanied by a small icon.

3. On the properties page for the selected group, select **Delete**. A message asks you to confirm this action. Select **Yes** if you're sure that you want to proceed.



Delete Hybrid Worker Group
The hybrid worker group TutorialTest will be deleted and you will not be able to run jobs on any of the workers in this group. Before you delete the worker group, unregister any servers you have defined as hybrid runbook workers by running the Remove-HybridRunbookWorker cmdlet. If you no longer have access to the runbook workers, deleting the hybrid worker group will remove the association between the workers in this group and your account. Are you sure you want to continue?

Yes **No**

This process can take several seconds to finish. You can track its progress under **Notifications** from the menu.

Configure your network

Hybrid Worker role

For the Hybrid Runbook Worker to connect to and register with Log Analytics, it must have access to the port

number and the URLs that are described in this section. This access is in addition to the [ports and URLs required for Microsoft Monitoring Agent](#) to connect to Log Analytics.

If you use a proxy server for communication between the agent and the Log Analytics service, ensure that the appropriate resources are accessible. If you use a firewall to restrict access to the internet, you must configure your firewall to permit access. If you use the OMS gateway as a proxy, ensure it is configured for hybrid workers. For instructions on how to do this, see [Configure the OMS Gateway for Automation Hybrid Workers](#).

The following port and URLs are required for the Hybrid Runbook Worker role to communicate with Automation:

- Port: Only TCP 443 is required for outbound internet access.
- Global URL: *.azure-automation.net
- Global URL of US Gov Virginia: *.azure-automation.us
- Agent service: <https://<workspaceId>.agentsvc.azure-automation.net>

It is recommended to use the addresses listed when defining exceptions. For IP addresses you can download the [Microsoft Azure Datacenter IP Ranges](#). This file is updated weekly, and reflects the currently deployed ranges and any upcoming changes to the IP ranges.

If you have an Automation account that's defined for a specific region, you can restrict communication to that regional datacenter. The following table provides the DNS record for each region:

REGION	DNS RECORD
West Central US	wcus-jobruntimedata-prod-su1.azure-automation.net wcuagentservice-prod-1.azure-automation.net
South Central US	scus-jobruntimedata-prod-su1.azure-automation.net scus-agentservice-prod-1.azure-automation.net
East US 2	eus2-jobruntimedata-prod-su1.azure-automation.net eus2-agentservice-prod-1.azure-automation.net
Canada Central	cc-jobruntimedata-prod-su1.azure-automation.net cc-agentservice-prod-1.azure-automation.net
West Europe	we-jobruntimedata-prod-su1.azure-automation.net we-agentservice-prod-1.azure-automation.net
North Europe	ne-jobruntimedata-prod-su1.azure-automation.net ne-agentservice-prod-1.azure-automation.net
South East Asia	sea-jobruntimedata-prod-su1.azure-automation.net sea-agentservice-prod-1.azure-automation.net
Central India	cid-jobruntimedata-prod-su1.azure-automation.net cid-agentservice-prod-1.azure-automation.net
Japan East	jpe-jobruntimedata-prod-su1.azure-automation.net jpe-agentservice-prod-1.azure-automation.net
Australia South East	ase-jobruntimedata-prod-su1.azure-automation.net ase-agentservice-prod-1.azure-automation.net

REGION	DNS RECORD
UK South	uks-jobruntimedata-prod-su1.azure-automation.net uks-agentservice-prod-1.azure-automation.net
US Gov Virginia	usge-jobruntimedata-prod-su1.azure-automation.us usge-agentservice-prod-1.azure-automation.us

For a list of region IP addresses instead of region names, download the [Azure Datacenter IP address](#) XML file from the Microsoft Download Center.

NOTE

The Azure Datacenter IP address XML file lists the IP address ranges that are used in the Microsoft Azure datacenters. The file includes compute, SQL, and storage ranges.

An updated file is posted weekly. The file reflects the currently deployed ranges and any upcoming changes to the IP ranges. New ranges that appear in the file aren't used in the datacenters for at least one week.

It's a good idea to download the new XML file every week. Then, update your site to correctly identify services running in Azure. Azure ExpressRoute users should note that this file is used to update the Border Gateway Protocol (BGP) advertisement of Azure space in the first week of each month.

Update Management

In addition to the standard addresses and ports that the Hybrid Runbook Worker requires, the following addresses are required specifically for Update Management. Communication to these addresses is done over port 443.

AZURE PUBLIC	AZURE GOVERNMENT
*.ods.opinsights.azure.com	*.ods.opinsights.azure.us
*.oms.opinsights.azure.com	*.oms.opinsights.azure.us
*.blob.core.windows.net	*.blob.core.usgovcloudapi.net

Troubleshoot

To learn how to troubleshoot your Hybrid Runbook Workers, see [Troubleshooting Hybrid Runbook Workers](#)

Next steps

To learn how to configure your runbooks to automate processes in your on-premises datacenter or other cloud environment, see [Run runbooks on a Hybrid Runbook Worker](#).

Deploy a Windows Hybrid Runbook Worker

6/27/2018 • 7 minutes to read • [Edit Online](#)

You can use the Hybrid Runbook Worker feature of Azure Automation to run runbooks directly on the computer that's hosting the role and against resources in the environment to manage those local resources. Runbooks are stored and managed in Azure Automation and then delivered to one or more designated computers. This article describes how to install the Hybrid Runbook Worker on a Windows machine.

Installing the Windows Hybrid Runbook Worker

To install and configure a Windows Hybrid Runbook Worker, you can use two methods. The recommended method is using an Automation runbook to completely automate the process of configuring a Windows computer. The second method is following a step-by-step procedure to manually install and configure the role.

NOTE

To manage the configuration of your servers that support the Hybrid Runbook Worker role with Desired State Configuration (DSC), you need to add them as DSC nodes.

The minimum requirements for a Windows Hybrid Runbook Worker are:

- Windows Server 2012 or later.
- Windows PowerShell 4.0 or later ([download WMF 4.0](#)). We recommend Windows PowerShell 5.1 ([download WMF 5.1](#)) for increased reliability.
- .NET Framework 4.6.2 or later.
- Two cores.
- 4 GB of RAM.
- Port 443 (outbound).

To get more networking requirements for the Hybrid Runbook Worker, see [Configuring your network](#).

For more information about onboarding servers for management with DSC, see [Onboarding machines for management by Azure Automation DSC](#). If you enable the [Update Management solution](#), any Windows computer that's connected to your Azure Log Analytics workspace is automatically configured as a Hybrid Runbook Worker to support runbooks included in this solution. However, it isn't registered with any Hybrid Worker groups already defined in your Automation account.

The computer can be added to a Hybrid Runbook Worker group in your Automation account to support Automation runbooks as long as you're using the same account for both the solution and the Hybrid Runbook Worker group membership. This functionality has been added to version 7.2.12024.0 of the Hybrid Runbook Worker.

After you successfully deploy a runbook worker, review [Run runbooks on a Hybrid Runbook Worker](#) to learn how to configure your runbooks to automate processes in your on-premises datacenter or other cloud environment.

Automated deployment

Perform the following steps to automate the installation and configuration of the Windows Hybrid Worker role:

1. Download the New-OnPremiseHybridWorker.ps1 script from the [PowerShell Gallery](#) directly from the computer running the Hybrid Runbook Worker role or from another computer in your environment. Copy the script to the worker.

The New-OnPremiseHybridWorker.ps1 script requires the following parameters during execution:

- *AutomationAccountName* (mandatory): The name of your Automation account.
- *AAResourceGroupName* (mandatory): The name of the resource group that's associated with your Automation account.
- *OMSResourceGroupName* (optional): The name of the resource group for the Operations Management Suite workspace. If this resource group is not specified, *AAResourceGroupName* is used.
- *HybridGroupName* (mandatory): The name of a Hybrid Runbook Worker group that you specify as a target for the runbooks that support this scenario.
- *SubscriptionID* (mandatory): The Azure subscription ID that your Automation account is in.
- *WorkspaceName* (optional): The Log Analytics workspace name. If you don't have a Log Analytics workspace, the script creates and configures one.

NOTE

Currently, the only Automation regions supported for integration with Log Analytics are **Australia Southeast**, **East US 2**, **Southeast Asia**, and **West Europe**. If your Automation account is not in one of those regions, the script creates a Log Analytics workspace but warns you that it can't link them together.

2. On your computer, open **Windows PowerShell** from the **Start** screen in Administrator mode.
3. From the PowerShell command-line shell, browse to the folder that contains the script that you downloaded. Change the values for the parameters *-AutomationAccountName*, *-AAResourceGroupName*, *-OMSResourceGroupName*, *-HybridGroupName*, *-SubscriptionId*, and *-WorkspaceName*. Then run the script.

NOTE

You're prompted to authenticate with Azure after you run the script. You *must* sign in with an account that's a member of the Subscription Admins role and co-administrator of the subscription.

```
.\\New-OnPremiseHybridWorker.ps1 -AutomationAccountName <NameofAutomationAccount> -AAResourceGroupName  
<NameofResourceGroup>  
-OMSResourceGroupName <NameofOResourceGroup> -HybridGroupName <NameofHRWGroup> `  
-SubscriptionId <AzureSubscriptionId> -WorkspaceName <NameOfLogAnalyticsWorkspace>
```

4. You're prompted to agree to install NuGet, and you're prompted to authenticate with your Azure credentials.
5. After the script is finished, the **Hybrid Worker Groups** page shows the new group and the number of members. If it's an existing group, the number of members is incremented. You can select the group from the list on the **Hybrid Worker Groups** page and select the **Hybrid Workers** tile. On the **Hybrid Workers** page, you see each member of the group listed.

Manual deployment

Perform the first two steps once for your Automation environment, and then repeat the remaining steps for each worker computer.

1. Create a Log Analytics workspace

If you don't already have a Log Analytics workspace, create one by using the instructions at [Manage your workspace](#). You can use an existing workspace if you already have one.

2. Add the Automation solution to the Log Analytics workspace

Solutions add functionality to Log Analytics. The Automation solution adds functionality for Azure Automation, including support for Hybrid Runbook Worker. When you add the solution to your workspace, it automatically

pushes worker components to the agent computer that you will install in the next step.

To add the **Automation** solution to your Log Analytics workspace, follow the instructions at [To add a solution using the Solutions Gallery](#).

3. Install the Microsoft Monitoring Agent

The Microsoft Monitoring Agent connects computers to Log Analytics. When you install the agent on your on-premises computer and connect it to your workspace, it automatically downloads the components that are required for Hybrid Runbook Worker.

To install the agent on the on-premises computer, follow the instructions at [Connect Windows computers to Log Analytics](#). You can repeat this process for multiple computers to add multiple workers to your environment.

When the agent has successfully connected to Log Analytics, it's listed on the **Connected Sources** tab of the Log Analytics **Settings** page. You can verify that the agent has correctly downloaded the Automation solution when it has a folder called **AzureAutomationFiles** in C:\Program Files\Microsoft Monitoring Agent\Agent. To confirm the version of the Hybrid Runbook Worker, you can browse to C:\Program Files\Microsoft Monitoring Agent\Agent\AzureAutomation\ and note the \version subfolder.

4. Install the runbook environment and connect to Azure Automation

When you add an agent to Log Analytics, the Automation solution pushes down the **HybridRegistration** PowerShell module, which contains the **Add-HybridRunbookWorker** cmdlet. You use this cmdlet to install the runbook environment on the computer and register it with Azure Automation.

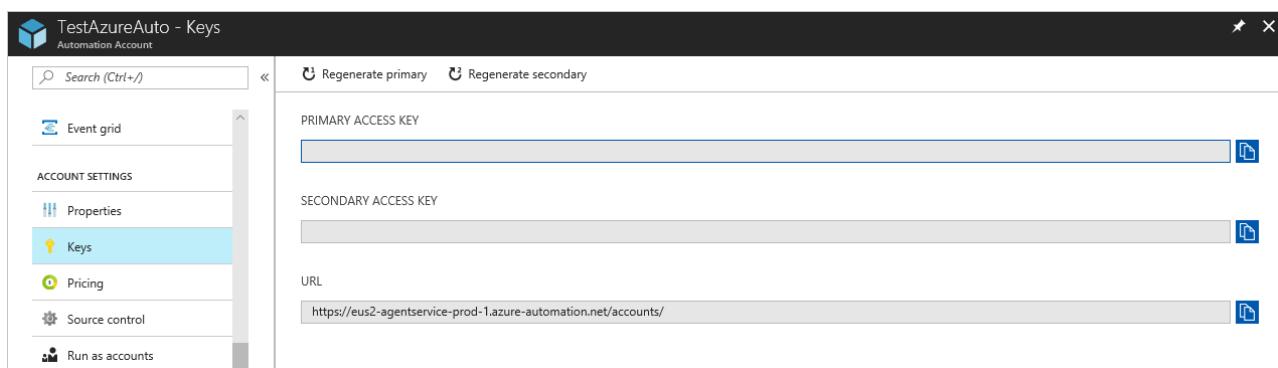
Open a PowerShell session in Administrator mode and run the following commands to import the module:

```
cd "C:\Program Files\Microsoft Monitoring Agent\Agent\AzureAutomation\<version>\HybridRegistration"
Import-Module HybridRegistration.psd1
```

Then run the **Add-HybridRunbookWorker** cmdlet by using the following syntax:

```
Add-HybridRunbookWorker -GroupName <String> -EndPoint <Url> -Token <String>
```

You can get the information required for this cmdlet from the **Manage Keys** page in the Azure portal. Open this page by selecting the **Keys** option from the **Settings** page in your Automation account.



- **GroupName** is the name of the Hybrid Runbook Worker group. If this group already exists in the Automation account, the current computer is added to it. If this group doesn't exist, it's added.
- **EndPoint** is the **URL** entry on the **Manage Keys** page.
- **Token** is the **PRIMARY ACCESS KEY** entry on the **Manage Keys** page.

To receive detailed information about the installation, use the **-Verbose** switch with **Add-HybridRunbookWorker**.

5. Install PowerShell modules

Runbooks can use any of the activities and cmdlets defined in the modules that are installed in your Azure

Automation environment. These modules are not automatically deployed to on-premises computers, so you must install them manually. The exception is the Azure module, which is installed by default and provides access to cmdlets for all Azure services and activities for Azure Automation.

Because the primary purpose of the Hybrid Runbook Worker feature is to manage local resources, you most likely need to install the modules that support these resources. For information on installing Windows PowerShell modules, see [Installing Modules](#).

Modules that are installed must be in a location referenced by the **PSModulePath** environment variable so that the hybrid worker can automatically import them. For more information, see [Modifying the PSModulePath Installation Path](#).

Troubleshoot

To learn how to troubleshoot your Hybrid Runbook Workers, see [Troubleshooting Windows Hybrid Runbook Workers](#)

For additional steps on how to troubleshoot issues with Update Management, see [Update Management: troubleshooting](#).

Next steps

- To learn how to configure your runbooks to automate processes in your on-premises datacenter or other cloud environment, see [Run runbooks on a Hybrid Runbook Worker](#).
- For instructions on how to remove Hybrid Runbook Workers, see [Remove Azure Automation Hybrid Runbook Workers](#).

Deploy a Linux Hybrid Runbook Worker

6/28/2018 • 3 minutes to read • [Edit Online](#)

You can use the Hybrid Runbook Worker feature of Azure Automation to run runbooks directly on the computer that's hosting the role and against resources in the environment to manage those local resources. The Linux Hybrid Runbook Worker executes runbooks as a special user that can be elevated for running commands that need elevation. Runbooks are stored and managed in Azure Automation and then delivered to one or more designated computers.

This article describes how to install the Hybrid Runbook Worker on a Linux machine.

Supported Linux operating systems

The Hybrid Runbook Worker feature supports the following distributions:

- Amazon Linux 2012.09 to 2015.09 (x86/x64)
- CentOS Linux 5, 6, and 7 (x86/x64)
- Oracle Linux 5, 6, and 7 (x86/x64)
- Red Hat Enterprise Linux Server 5, 6, and 7 (x86/x64)
- Debian GNU/Linux 6, 7, and 8 (x86/x64)
- Ubuntu 12.04 LTS, 14.04 LTS, and 16.04 LTS (x86/x64)
- SUSE Linux Enterprise Server 11 and 12 (x86/x64)

Installing a Linux Hybrid Runbook Worker

To install and configure a Hybrid Runbook Worker on your Linux computer, you follow a straightforward process to manually install and configure the role. It requires enabling the **Automation Hybrid Worker** solution in your Azure Log Analytics workspace and then running a set of commands to register the computer as a worker and add it to a group.

The minimum requirements for a Linux Hybrid Runbook Worker are:

- Two cores
- 4 GB of RAM
- Port 443 (outbound)

Package requirements

REQUIRED PACKAGE	DESCRIPTION	MINIMUM VERSION
Glibc	GNU C Library	2.5-12
Openssl	OpenSSL Libraries	1.0 (TLS 1.1 and TLS 1.2 are supported)
Curl	cURL web client	7.15.5
Python-ctypes		
PAM	Pluggable Authentication Modules	

REQUIRED PACKAGE	DESCRIPTION	MINIMUM VERSION
Optional package	Description	Minimum version
PowerShell Core	To run PowerShell runbooks, PowerShell needs to be installed, see Installing PowerShell Core on Linux to learn how to install it.	6.0.0

Installation

Before you proceed, note the Log Analytics workspace that your Automation account is linked to. Also note the primary key for your Automation account. You can find both from the Azure portal by selecting your Automation account, selecting **Workspace** for the workspace ID, and selecting **Keys** for the primary key. For information on ports and addresses that you need for the Hybrid Runbook Worker, see [Configuring your network](#).

1. Enable the **Automation Hybrid Worker** solution in Azure by using one of the following methods:

- Add the **Automation Hybrid Worker** solution to your subscription by using the procedure at [Add Log Analytics management solutions to your workspace](#).
- Run the following cmdlet:

```
Set-AzureRmOperationalInsightsIntelligencePack -ResourceGroupName <ResourceGroupName> - WorkspaceName <WorkspaceName> -IntelligencePackName "AzureAutomation" -Enabled $true
```

2. Install the OMS Agent for Linux by running the following command. Replace <WorkspaceID> and <WorkspaceKey> with the appropriate values from your workspace.

```
wget https://raw.githubusercontent.com/Microsoft/OMS-Agent-for-Linux/master/installer/scripts/onboard_agent.sh && sh onboard_agent.sh -w <WorkspaceID> -s <WorkspaceKey>
```

3. Run the following command, changing the values for the parameters *-w*, *-k*, *-g*, and *-e*. For the *-g* parameter, replace the value with the name of the Hybrid Runbook Worker group that the new Linux Hybrid Runbook Worker should join. If the name doesn't exist in your Automation account, a new Hybrid Runbook Worker group is made with that name.

```
sudo python /opt/microsoft/omsconfig/modules/nxOMSAutomationWorker/DSCResources/MSFT_nxOMSAutomationWorkerResource/automationworker/scripts/onboarding.py --register -w <LogAnalyticsworkspaceId> -k <AutomationSharedKey> -g <hybridgroupname> -e <automationendpoint>
```

4. After the command is completed, the **Hybrid Worker Groups** page in the Azure portal shows the new group and the number of members. If this is an existing group, the number of members is incremented. You can select the group from the list on the **Hybrid Worker Groups** page and select the **Hybrid Workers** tile. On the **Hybrid Workers** page, you see each member of the group listed.

Turning off signature validation

By default, Linux Hybrid Runbook Workers require signature validation. If you run an unsigned runbook against a worker, you see an error that says "Signature validation failed." To turn off signature validation, run the following command. Replace the second parameter with your Log Analytics workspace ID.

```
sudo python  
/opt/microsoft/omsconfig/modules/nxOMSAutomationWorker/DSCResources/MSFT_nxOMSAutomationWorkerResource/automationworker/scripts/require_runbook_signature.py --false <LogAnalyticsworkspaceId>
```

Supported runbook types

Linux Hybrid Runbook Workers don't support the full set of runbook types in Azure Automation.

The following runbook types work on a Linux Hybrid Worker:

- Python 2
- PowerShell

NOTE

PowerShell runbooks require PowerShell Core to be installed on the Linux machine. See [Installing PowerShell Core on Linux](#) to learn how to install it.

The following runbook types don't work on a Linux Hybrid Worker:

- PowerShell Workflow
- Graphical
- Graphical PowerShell Workflow

Troubleshoot

To learn how to troubleshoot your Hybrid Runbook Workers, see [Troubleshooting Linux Hybrid Runbook Workers](#)

Next steps

- To learn how to configure your runbooks to automate processes in your on-premises datacenter or other cloud environment, see [Run runbooks on a Hybrid Runbook Worker](#).
- For instructions on how to remove Hybrid Runbook Workers, see [Remove Azure Automation Hybrid Runbook Workers](#).

Running runbooks on a Hybrid Runbook Worker

7/9/2018 • 7 minutes to read • [Edit Online](#)

There is no difference in the structure of runbooks that run in Azure Automation and those that run on a Hybrid Runbook Worker. Runbooks that you use with each most likely differ significantly though since runbooks targeting a Hybrid Runbook Worker typically manage resources on the local computer itself or against resources in the local environment where it is deployed, while runbooks in Azure Automation typically manage resources in the Azure cloud.

When you author runbooks to run on a Hybrid Runbook Worker, you should edit and test the runbooks within the machine that hosts the Hybrid worker. The host machine has all of the PowerShell modules and network access you need to manage and access the local resources. Once a runbook has been edited and tested on the Hybrid worker machine, you can then upload it to the Azure Automation environment where it is available to run in the Hybrid worker. It is important to know that jobs run under the Local System account for windows or a special user account **nxautomation** on Linux, which can introduce subtle differences when authoring runbooks for a Hybrid Runbook Worker this should be taken into account.

Starting a runbook on Hybrid Runbook Worker

[Starting a Runbook in Azure Automation](#) describes different methods for starting a runbook. Hybrid Runbook Worker adds a **RunOn** option where you can specify the name of a Hybrid Runbook Worker Group. If a group is specified, then the runbook is retrieved and run by one of the workers in that group. If this option is not specified, then it is run in Azure Automation as normal.

When you start a runbook in the Azure portal, you are presented with a **Run on** option where you can select **Azure** or **Hybrid Worker**. If you select **Hybrid Worker**, then you can select the group from a dropdown.

Use the **RunOn** parameter. You can use the following command to start a runbook named Test-Runbook on a Hybrid Runbook Worker Group named MyHybridGroup using Windows PowerShell.

```
Start-AzureRmAutomationRunbook -AutomationAccountName "MyAutomationAccount" -Name "Test-Runbook" -RunOn "MyHybridGroup"
```

NOTE

The **RunOn** parameter was added to the **Start-AzureAutomationRunbook** cmdlet in version 0.9.1 of Microsoft Azure PowerShell. You should [download the latest version](#) if you have an earlier one installed. You only need to install this version on a workstation where you are starting the runbook from PowerShell. You do not need to install it on the worker computer unless you intend to start runbooks from that computer”

Runbook permissions

Runbooks running on a Hybrid Runbook Worker cannot use the same method that is typically used for runbooks authenticating to Azure resources, since they are accessing resources outside of Azure. The runbook can either provide its own authentication to local resources, or you can specify a RunAs account to provide a user context for all runbooks.

Runbook authentication

By default, runbooks run in the context of the Local System account for Windows and a special user account **nxautomation** for Linux on the on-premises computer, so they must provide their own authentication to

resources that they access.

You can use [Credential](#) and [Certificate](#) assets in your runbook with cmdlets that allow you to specify credentials so you can authenticate to different resources. The following example shows a portion of a runbook that restarts a computer. It retrieves credentials from a credential asset and the name of the computer from a variable asset and then uses these values with the `Restart-Computer` cmdlet.

```
$Cred = Get-AzureRmAutomationCredential -ResourceGroupName "ResourceGroup01" -Name "MyCredential"
$Computer = Get-AzureRmAutomationVariable -ResourceGroupName "ResourceGroup01" -Name "ComputerName"

Restart-Computer -ComputerName $Computer -Credential $Cred
```

You can also leverage [InlineScript](#), which allows you to run blocks of code on another computer with credentials specified by the [PSCredential common parameter](#).

RunAs account

By default the Hybrid Runbook Worker uses Local System for Windows and a special user account `nxautomation` for Linux to execute runbooks. Instead of having runbooks provide their own authentication to local resources, you can specify a **RunAs** account for a Hybrid worker group. You specify a [credential asset](#) that has access to local resources, and all runbooks run under these credentials when running on a Hybrid Runbook Worker in the group.

The user name for the credential must be in one of the following formats:

- domain\username
- username@domain
- username (for accounts local to the on-premises computer)

Use the following procedure to specify a RunAs account for a Hybrid worker group:

1. Create a [credential asset](#) with access to local resources.
2. Open the Automation account in the Azure portal.
3. Select the **Hybrid Worker Groups** tile, and then select the group.
4. Select **All settings** and then **Hybrid worker group settings**.
5. Change **Run As** from **Default** to **Custom**.
6. Select the credential and click **Save**.

Automation Run As account

As part of your automated build process for deploying resources in Azure, you may require access to on-premises systems to support a task or set of steps in your deployment sequence. To support authentication against Azure using the Run As account, you need to install the Run As account certificate.

The following PowerShell runbook, `Export-RunAsCertificateToHybridWorker`, exports the Run As certificate from your Azure Automation account and downloads and imports it into the local machine certificate store on a Hybrid worker connected to the same account. Once that step is completed, it verifies the worker can successfully authenticate to Azure using the Run As account.

```
<#PSScriptInfo
.VERSION 1.0
.GUID 3a796b9a-623d-499d-86c8-c249f10a6986
.AUTHOR Azure Automation Team
.COMPANYNAME Microsoft
.COPYRIGHT
.TAGS Azure Automation
.LICENSEURI
.PROJECTURI
.ICONURI
```

```

.EXTERNALMODULEDEPENDENCIES
.REQUIREDSCRIPTS
.EXTERNALSCRIPTDEPENDENCIES
.RELEASENOTES
#>

<#
.SYNOPSIS
Exports the Run As certificate from an Azure Automation account to a hybrid worker in that account.

.DESCRIPTION
This runbook exports the Run As certificate from an Azure Automation account to a hybrid worker in that account.
Run this runbook in the hybrid worker where you want the certificate installed.
This allows the use of the AzureRunAsConnection to authenticate to Azure and manage Azure resources from runbooks running in the hybrid worker.

.EXAMPLE
.\Export-RunAsCertificateToHybridWorker

.NOTES
AUTHOR: Azure Automation Team
LASTEDIT: 2016.10.13
#>

# Generate the password used for this certificate
Add-Type -AssemblyName System.Web -ErrorAction SilentlyContinue | Out-Null
$Password = [System.Web.Security.Membership]::GeneratePassword(25, 10)

# Stop on errors
$ErrorActionPreference = 'stop'

# Get the management certificate that will be used to make calls into Azure Service Management resources
$RunAsCert = Get-AutomationCertificate -Name "AzureRunAsCertificate"

# location to store temporary certificate in the Automation service host
$CertPath = Join-Path $env:temp "AzureRunAsCertificate.pfx"

# Save the certificate
$Cert = $RunAsCert.Export("pfx",$Password)
Set-Content -Value $Cert -Path $CertPath -Force -Encoding Byte | Write-Verbose

Write-Output ("Importing certificate into $env:computername local machine root store from " + $CertPath)
$SecurePassword = ConvertTo-SecureString $Password -AsPlainText -Force
Import-PfxCertificate -FilePath $CertPath -CertStoreLocation Cert:\LocalMachine\My -Password $SecurePassword
-Exportable | Write-Verbose

# Test that authentication to Azure Resource Manager is working
$RunAsConnection = Get-AutomationConnection -Name "AzureRunAsConnection"

Connect-AzureRmAccount ` 
    -ServicePrincipal ` 
    -TenantId $RunAsConnection.TenantId ` 
    -ApplicationId $RunAsConnection.ApplicationId ` 
    -CertificateThumbprint $RunAsConnection.CertificateThumbprint | Write-Verbose

Set-AzureRmContext -SubscriptionId $RunAsConnection.SubscriptionID | Write-Verbose

# List automation accounts to confirm Azure Resource Manager calls are working
Get-AzureRmAutomationAccount | Select-Object AutomationAccountName

```

IMPORTANT

Add-AzureRmAccount is now an alias for **Connect-AzureRMAccount**. When searching your library items, if you do not see **Connect-AzureRMAccount**, you can use **Add-AzureRmAccount**, or you can update your modules in your Automation Account.

Save the *Export-RunAsCertificateToHybridWorker* runbook to your computer with a `.ps1` extension. Import it into your Automation account and edit the runbook, changing the value of the variable `$Password` with your own password. Publish and then run the runbook targeting the Hybrid Worker group that run and authenticate runbooks using the Run As account. The job stream reports the attempt to import the certificate into the local machine store, and follows with multiple lines depending on how many Automation accounts are defined in your subscription and if authentication is successful.

Job behavior

Jobs are handled slightly different on Hybrid Runbook Workers than they are when they run on Azure sandboxes. One key difference is that there is no limit on job duration on Hybrid Runbook Workers. Runbooks ran in Azure sandboxes are limited to 3 hours due to [fair share](#). If you have a long-running runbook you want to ensure that it is resilient to possible restart, for example if the machine that hosts the Hybrid worker reboots. If the Hybrid worker host machine reboots, then any running runbook job restarts from the beginning, or from the last checkpoint for PowerShell Workflow runbooks. If a runbook job is restarted more than 3 times, then it is suspended.

Troubleshoot

If your runbooks are not completing successfully and the job summary shows a status of **Suspended**, review the troubleshooting guide on [runbook execution failures](#).

Next steps

- To learn more about the different methods that can be used to start a runbook, see [Starting a Runbook in Azure Automation](#).
- To understand the different procedures for working with PowerShell and PowerShell Workflow runbooks in Azure Automation using the textual editor, see [Editing a Runbook in Azure Automation](#)

Azure Automation DSC Overview

5/21/2018 • 2 minutes to read • [Edit Online](#)

Azure Automation DSC is an Azure service that allows you to write, manage, and compile PowerShell Desired State Configuration (DSC) [configurations](#), import [DSC Resources](#), and assign configurations to target nodes, all in the cloud.

Why use Azure Automation DSC

Azure Automation DSC provides several advantages over using DSC outside of Azure.

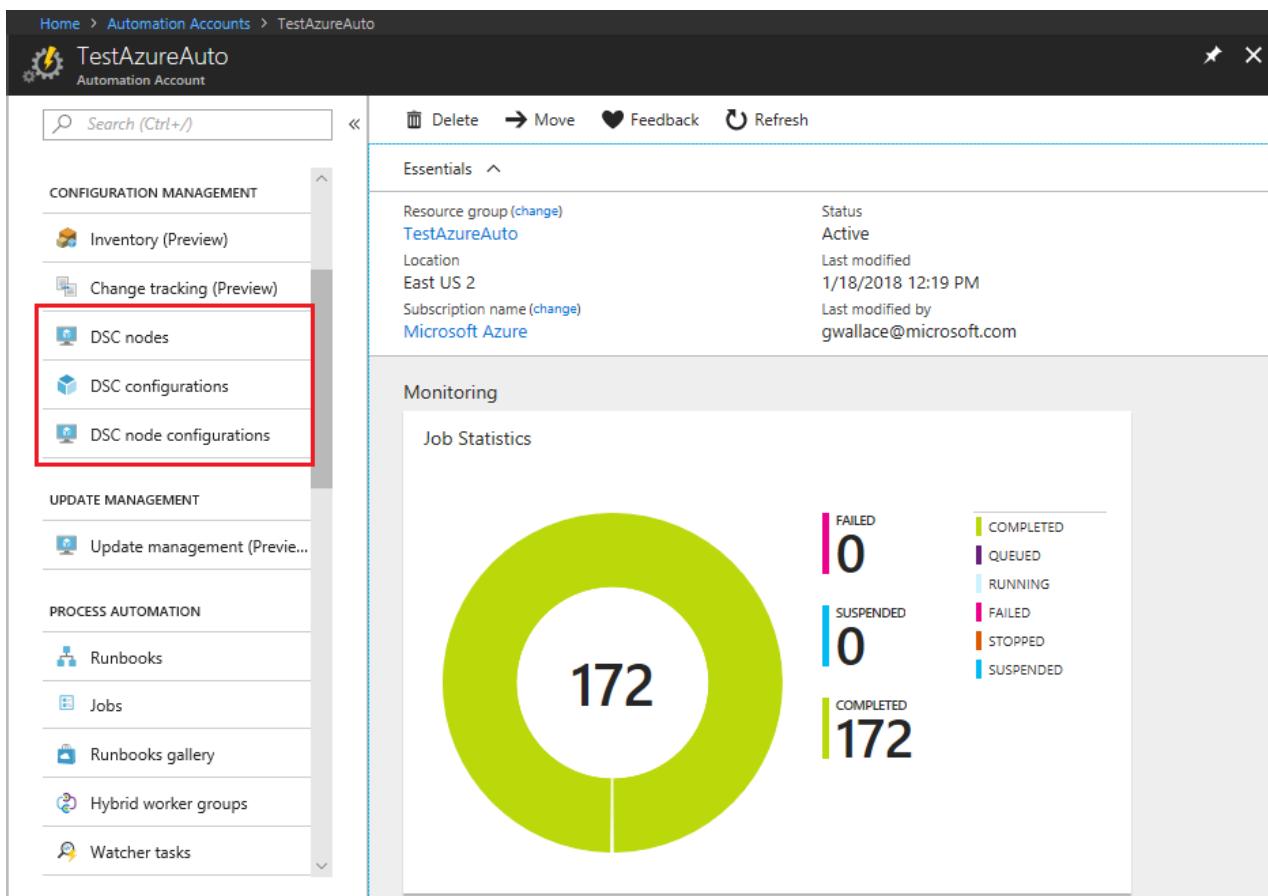
Built-in pull server

Azure Automation provides a [DSC pull server](#) so that target nodes automatically receive configurations, conform to the desired state, and report back on their compliance. The built-in pull server in Azure Automation eliminates the need to set up and maintain your own pull server. Azure Automation can target virtual or physical Windows or Linux machines, in the cloud or on-premises.

Management of all your DSC artifacts

Azure Automation DSC brings the same management layer to [PowerShell Desired State Configuration](#) as Azure Automation offers for PowerShell scripting.

From the Azure portal, or from PowerShell, you can manage all your DSC configurations, resources, and target nodes.



Import reporting data into Log Analytics

Nodes that are managed with Azure Automation DSC send detailed reporting status data to the built-in pull server. You can configure Azure Automation DSC to send this data to your Log Analytics workspace. To learn how

to send DSC status data to your Log Analytics workspace, see [Forward Azure Automation DSC reporting data to Log Analytics](#).

Introduction video

Prefer watching to reading? Have a look at the following video from May 2015, when Azure Automation DSC was first announced.

NOTE

While the concepts and life cycle discussed in this video are correct, Azure Automation DSC has progressed a lot since this video was recorded. It is now generally available, has a much more extensive UI in the Azure portal, and supports many additional capabilities.

Next steps

- To learn how to onboard nodes to be managed with Azure Automation DSC, see [Onboarding machines for management by Azure Automation DSC](#)
- To get started using Azure Automation DSC, see [Getting started with Azure Automation DSC](#)
- To learn about compiling DSC configurations so that you can assign them to target nodes, see [Compiling configurations in Azure Automation DSC](#)
- For PowerShell cmdlet reference for Azure Automation DSC, see [Azure Automation DSC cmdlets](#)
- For pricing information, see [Azure Automation DSC pricing](#)
- To see an example of using Azure Automation DSC in a continuous deployment pipeline, see [Continuous Deployment to IaaS VMs Using Azure Automation DSC and Chocolatey](#)

Getting started with Azure Automation DSC

5/21/2018 • 8 minutes to read • [Edit Online](#)

This article explains how to do the most common tasks with Azure Automation Desired State Configuration (DSC), such as creating, importing, and compiling configurations, onboarding machines to manage, and viewing reports. For an overview of what Azure Automation DSC is, see [Azure Automation DSC Overview](#). For DSC documentation, see [Windows PowerShell Desired State Configuration Overview](#).

This article provides a step-by-step guide to using Azure Automation DSC. If you want a sample environment that is already set up without following the steps described in this article, you can use the following Resource Manager template: This template sets up a completed Azure Automation DSC environment, including an Azure VM that is managed by Azure Automation DSC.

Prerequisites

To complete the examples in this article, the following are required:

- An Azure Automation account. For instructions on creating an Azure Automation Run As account, see [Azure Run As Account](#).
- An Azure Resource Manager VM (not Classic) running Windows Server 2008 R2 or later. For instructions on creating a VM, see [Create your first Windows virtual machine in the Azure portal](#)

Creating a DSC configuration

You create a simple [DSC configuration](#) that ensures either the presence or absence of the **Web-Server** Windows Feature (IIS), depending on how you assign nodes.

1. Start the Windows PowerShell ISE (or any text editor).
2. Type the following text:

```
configuration TestConfig
{
    Node IsWebServer
    {
        WindowsFeature IIS
        {
            Ensure          = 'Present'
            Name           = 'Web-Server'
            IncludeAllSubFeature = $true
        }
    }

    Node NotWebServer
    {
        WindowsFeature IIS
        {
            Ensure          = 'Absent'
            Name           = 'Web-Server'
        }
    }
}
```

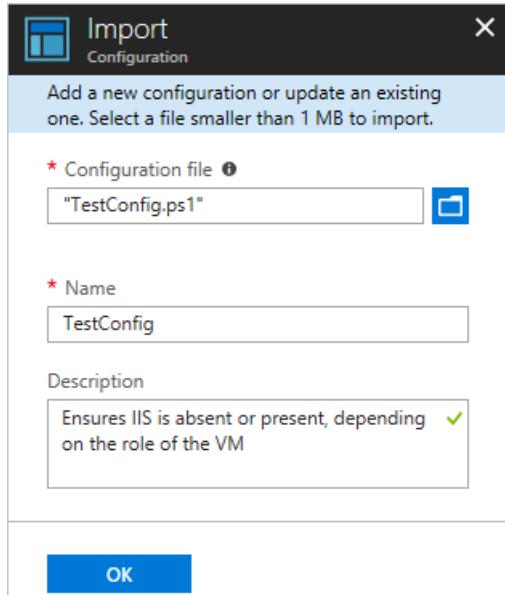
3. Save the file as `TestConfig.ps1`.

This configuration calls one resource in each node block, the [WindowsFeature](#) resource, that ensures either the presence or absence of the **Web-Server** feature.

Importing a configuration into Azure Automation

Next, you import the configuration into the Automation account.

1. Sign in to the [Azure portal](#).
2. On the left, click **All resources** and then the name of your Automation account.
3. On the **Automation account** page, select **DSC Configurations** under **Configuration Management**.
4. On the **DSC Configurations** page, click **+ Add a configuration**.
5. On the **Import Configuration** page, browse to the `TestConfig.ps1` file on your computer.



6. Click **OK**.

Viewing a configuration in Azure Automation

After you have imported a configuration, you can view it in the Azure portal.

1. Sign in to the [Azure portal](#).
2. On the left, click **All resources** and then the name of your Automation account.
3. On the **Automation account** page, select **DSC Configurations** under **Configuration Management**.
4. On the **DSC Configurations** page, click **TestConfig** (this is the name of the configuration you imported in the previous procedure).
5. On the **TestConfig Configuration** page, click **View configuration source**.

The screenshot shows the Azure portal interface for a configuration named 'TestConfig'. At the top, there are buttons for 'Compile', 'Export', and 'Delete'. Below that, a section titled 'Essentials' contains details about the configuration's resource group ('TestAzureAuto'), location ('eastus2'), subscription ID, and last published date ('1/23/2018 9:53 AM'). A 'View configuration source' link is highlighted with a red box. Below this, a section titled 'Deployments to Pull Server' shows a table for 'Compilation jobs' with columns for STATUS, CREATED, and LAST UPDATED. The table displays the message 'No compilation jobs found.'

A **TestConfig Configuration source** blade opens, displaying the PowerShell code for the configuration.

Compiling a configuration in Azure Automation

Before you can apply a desired state to a node, a DSC configuration defining that state must be compiled into one or more node configurations (MOF document), and placed on the Automation DSC Pull Server. For a more detailed description of compiling configurations in Azure Automation DSC, see [Compiling configurations in Azure Automation DSC](#). For more information about compiling configurations, see [DSC Configurations](#).

1. Sign in to the [Azure portal](#).
2. On the left, click **All resources** and then the name of your Automation account.
3. On the **Automation account** page, click **DSC Configurations** under **Configuration Management**.
4. On the **DSC Configurations** page, click **TestConfig** (the name of the previously imported configuration).
5. On the **TestConfig Configuration** page, click **Compile**, and then click **Yes**. This starts a compilation job.

The screenshot shows the Azure portal interface for a configuration named 'TestConfig'. At the top, there are buttons for 'Compile' (which is highlighted with a red box), 'Export', and 'Delete'. Below this is a section titled 'Essentials' with the following details:

Resource group	Account
TestAzureAuto	TestAzureAuto
Location	Subscription name
eastus2	Microsoft Azure
Subscription ID	Status
Last published	Published
1/23/2018 9:53 AM	Configuration source View configuration source

Below the essentials section is a 'Deployments to Pull Server' area. It contains a 'Compilation jobs' tile with the following structure:

STATUS	CREATED	LAST UPDATED
No compilation jobs found.		

NOTE

When you compile a configuration in Azure Automation, it automatically deploys any created node configuration MOFs to the pull server.

Viewing a compilation job

After you start a compilation, you can view it in the **Compilation jobs** tile in the **Configuration** blade. The **Compilation jobs** tile shows currently running, completed, and failed jobs. When you open a compilation job blade, it shows information about that job including any errors or warnings encountered, input parameters used in the configuration, and compilation logs.

1. Sign in to the [Azure portal](#).
2. On the left, click **All resources** and then the name of your Automation account.
3. On the **Automation account** page, click **DSC Configurations** under **Configuration Management**.
4. On the **DSC Configurations** page, click **TestConfig** (the name of the previously imported configuration).
5. Under **Compilation jobs**, select the compilation job you want to view. A **Compilation Job** page opens, labeled with the date that the compilation job was started.

1/23/2018 10:08 AM
Compilation Job

Essentials

Resource group	Account
TestAzureAuto	TestAzureAuto
Location	Subscription name
eastus2	Microsoft Azure
State	Start time
Completed	1/23/2018 10:10 AM
Last Update	Total runtime
1/23/2018 10:10 AM	Less than one minute
Job id	Configuration
83d2f2f3-f87e-4f0d-b10d-973150d44b24	TestConfig

Details

Input	<code></></code>
0 ➔	Configuration source snapshot

Monitoring

Errors	0 ✖
Warnings	2 ⚠
All Logs	<code>Logs</code>

- Click on any tile in the **Compilation Job** page to see further details about the job.

Viewing node configurations

Successful completion of a compilation job creates one or more new node configurations. A node configuration is a MOF document that is deployed to the pull server and ready to be pulled and applied by one or more nodes. You can view the node configurations in your Automation account in the **DSC Node Configurations** blade. A node configuration has a name with the form *ConfigurationName.NodeName*.

- Sign in to the [Azure portal](#).
- On the Hub menu, click **All resources** and then the name of your Automation account.
- On the **Automation account** blade, click **DSC Node Configurations**.

TestAzureAuto - DSC node configurations
Automation Account

Search (Ctrl+ /)

+ Add a NodeConfiguration X Delete ⌂ Refresh

NAME	CREATED	LAST MODIFIED
TestConfig.lsWebServer	1/23/2018 10:19 AM	1/23/2018 10:19 AM
TestConfig.WebServer	1/23/2018 10:21 AM	1/23/2018 10:21 AM

Inventory (Preview)
Change tracking (Preview)
DSC nodes
DSC configurations
DSC node configurations

Onboarding an Azure VM for management with Azure Automation DSC

You can use Azure Automation DSC to manage Azure VMs (both Classic and Resource Manager), on-premises

VMs, Linux machines, AWS VMs, and on-premises physical machines. In this article, you learn how to onboard only Azure Resource Manager VMs. For information about onboarding other types of machines, see [Onboarding machines for management by Azure Automation DSC](#).

To onboard an Azure Resource Manager VM for management by Azure Automation DSC

1. Sign in to the [Azure portal](#).
2. On the left, click **All resources** and then the name of your Automation account.
3. On the **Automation account** page, click **DSC Nodes** under **Configuration Management**.
4. In the **DSC Nodes** page, click **Add Azure VM**.

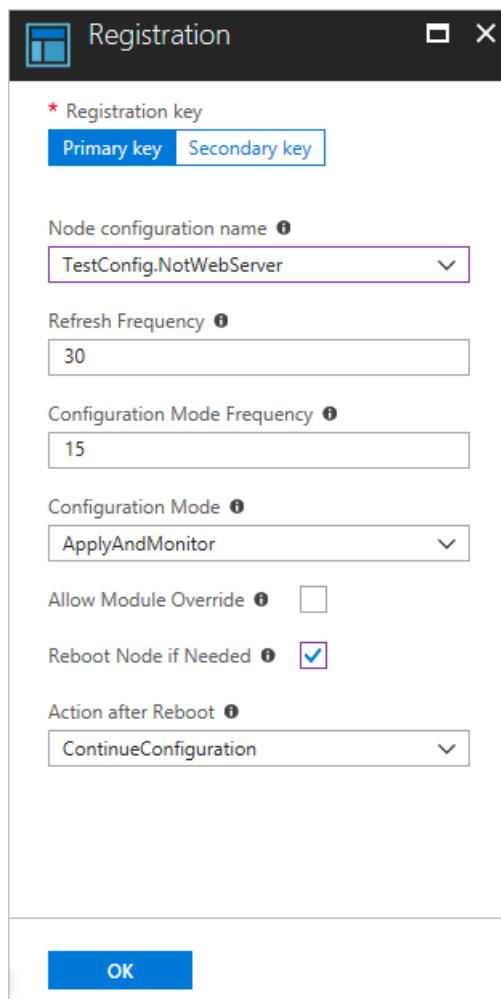
The screenshot shows the 'DSC nodes' page in the Azure portal. The title bar says 'TestAzureAuto - DSC nodes' and 'Automation Account'. On the left, there's a sidebar with 'Diagnose and solve problems', 'CONFIGURATION MANAGEMENT' section (Inventory (Preview), Change tracking (Preview), DSC nodes, DSC configurations, DSC node configurations), and a search bar. The main area has a header with 'Add Azure VM' (highlighted with a red box), 'Add Non-Azure', 'Learn more', 'Refresh', and 'Log search'. Below is an info message: 'Azure VMs managed by the configuration service do not incur a cost if they have the VM DSC extension v'. A table lists 'DSC nodes' with columns: NAME, STATUS, and NODE CONFIGURATION. It shows 7 selected nodes and 0 selected configurations. The status column shows 'No DSC nodes found.'

5. On the Virtual Machines page, select your VM. **Add Azure VMs** page, click **Select virtual machines to onboard**.
6. Click **Connect**.

IMPORTANT

This must be an Azure Resource Manager VM running Windows Server 2008 R2 or later.

7. In the **Registration** page, enter the name of the node configuration you want to apply to the VM in the **Node Configuration Name** box. This must exactly match the name of a node configuration in the Automation account. Providing a name at this point is optional. You can change the assigned node configuration after onboarding the node. Check **Reboot Node if Needed**, and then click **OK**.



The node configuration you specified are applied to the VM at intervals specified by the **Configuration Mode Frequency**, and the VM checks for updates to the node configuration at intervals specified by the **Refresh Frequency**. For more information about how these values are used, see [Configuring the Local Configuration Manager](#).

8. In the **Add Azure VMs** blade, click **Create**.

Azure starts the process of onboarding the VM. When it is complete, the VM shows up in the **DSC Nodes** blade in the Automation account.

Viewing the list of DSC nodes

You can view the list of all machines that have been onboarded for management in your Automation account in the **DSC Nodes** blade.

1. Sign in to the [Azure portal](#).
2. On the left, click **All resources** and then the name of your Automation account.
3. On the **Automation account** page, click **DSC Nodes**.

Viewing reports for DSC nodes

Each time Azure Automation DSC performs a consistency check on a managed node, the node sends a status report back to the pull server. You can view these reports on the page for that node.

1. Sign in to the [Azure portal](#).
2. On the left, click **All resources** and then the name of your Automation account.
3. On the **Automation account** page, click **DSC Nodes**.
4. In the **DSC nodes** list, select the node you want to view.

- On the **Node** page, click the report you want to view under **Reports**.

The screenshot shows two windows side-by-side. The left window is titled 'ContosoVM1' and displays node details. The right window is titled 'Report' and shows a list of consistency checks.

TYPE	STATUS	REPORT TIME
Consistency	✓ Compliant	12/7/2017 9:30 AM
Consistency	✓ Compliant	12/7/2017 9:14 AM
Consistency	✓ Compliant	12/7/2017 9:14 AM
Consistency	✓ Compliant	12/7/2017 8:59 AM
Consistency	✓ Compliant	12/7/2017 8:44 AM
Consistency	✓ Compliant	12/7/2017 8:44 AM
Consistency	✓ Compliant	12/7/2017 8:30 AM

On the blade for an individual report, you can see the following status information for the corresponding consistency check:

- The report status — whether the node is "Compliant", the configuration "Failed", or the node is "Not Compliant" (when the node is in **applyandmonitor** mode and the machine is not in the desired state).
- The start time for the consistency check.
- The total runtime for the consistency check.
- The type of consistency check.
- Any errors, including the error code and error message.
- Any DSC resources used in the configuration, and the state of each resource (whether the node is in the desired state for that resource) — you can click on each resource to get more detailed information for that resource.
- The name, IP address, and configuration mode of the node.

You can also click **View raw report** to see the actual data that the node sends to the server. For more information about using that data, see [Using a DSC report server](#).

It can take some time after a node is onboarded before the first report is available. You might need to wait up to 30 minutes for the first report after you onboard a node.

Reassigning a node to a different node configuration

You can assign a node to use a different node configuration than the one you initially assigned.

- Sign in to the [Azure portal](#).
- On the left, click **All resources** and then the name of your Automation account.
- On the **Automation account** page, click **DSC Nodes**.
- On the **DSC Nodes** page, click on the name of the node you want to reassign.
- On the page for that node, click **Assign node**.

The screenshot shows the Azure portal interface for a node named 'ContosoVM1'. At the top, there's a navigation bar with a blue square icon and the text 'ContosoVM1'. Below it is a header with two buttons: 'Assign node configuration' (highlighted with a red box) and 'Unregister'. The main content area is titled 'Essentials ^'. It contains several key details about the node:

Resource group	IP address
TestAzureAuto	10.1.0.4
Id	Account
Last seen time	TestAzureAuto
12/7/2017 9:30 AM	Virtual machine
Configuration	ContosoVM1
TestConfig	Node configuration
Registration time	TestConfig.NotWebServer
12/4/2017 9:54 AM	Status
	Unresponsive

6. On the **Assign Node Configuration** page, select the node configuration to which you want to assign the node, and then click **OK**.

This screenshot shows the 'Assign Node Configuration' dialog box. At the top, it says 'Assign Node Configuration' and 'ContosoVM1'. Below that is a message: 'Changing the node configuration assigned to a node will cause the node to change its configuration to match the node configuration next time it pulls.' A large table lists available configurations:

NAME	LAST MODIFIED
TestConfig.NotWebServer	1/23/2018 10:19 AM
TestConfig.IsWebServer	1/23/2018 10:21 AM

At the bottom is a blue 'OK' button.

Unregistering a node

If you no longer want a node to be managed by Azure Automation DSC, you can unregister it.

1. Sign in to the [Azure portal](#).
2. On the left, click **All resources** and then the name of your Automation account.
3. On the **Automation account** page, click **DSC Nodes**.
4. On the **DSC Nodes** page, click on the name of the node you want to unregister.
5. On the page for that node, click **Unregister**.

This screenshot shows a confirmation dialog box titled 'Unregister Node'. It contains the question 'Are you sure you want to unregister this node from Azure Automation DSC?'. At the bottom are two buttons: 'Yes' and 'No'.

Related Articles

- [Azure Automation DSC overview](#)

- [Onboarding machines for management by Azure Automation DSC](#)
- [Windows PowerShell Desired State Configuration Overview](#)
- [Azure Automation DSC cmdlets](#)
- [Azure Automation DSC pricing](#)

Configure servers to a desired state and manage drift

5/21/2018 • 4 minutes to read • [Edit Online](#)

Azure Automation Desired State Configuration (DSC) allows you to specify configurations for your servers and ensure that those servers are in the specified state over time.

- Onboard a VM to be managed by Azure Automation DSC
- Upload a configuration to Azure Automation
- Compile a configuration into a node configuration
- Assign a node configuration to a managed node
- Check the compliance status of a managed node

Prerequisites

To complete this tutorial, you will need:

- An Azure Automation account. For instructions on creating an Azure Automation Run As account, see [Azure Run As Account](#).
- An Azure Resource Manager VM (not Classic) running Windows Server 2008 R2 or later. For instructions on creating a VM, see [Create your first Windows virtual machine in the Azure portal](#)
- Azure PowerShell module version 3.6 or later. Run `Get-Module -ListAvailable AzureRM` to find the version. If you need to upgrade, see [Install Azure PowerShell module](#).
- Familiarity with DSC. For information about DSC, see [Windows PowerShell Desired State Configuration Overview](#)

Log in to Azure

Log in to your Azure subscription with the `Connect-AzureRmAccount` command and follow the on-screen directions.

```
Connect-AzureRmAccount
```

Create and upload a configuration to Azure Automation

For this tutorial, we will use a simple DSC configuration that ensures that IIS is installed on the VM.

For information about DSC configurations, see [DSC configurations](#).

In a text editor, type the following and save it locally as `TestConfig.ps1`.

```
configuration TestConfig {
    Node WebServer {
        WindowsFeature IIS {
            Ensure          = 'Present'
            Name            = 'Web-Server'
            IncludeAllSubFeature = $true
        }
    }
}
```

Call the `Import-AzureRmAutomationDscConfiguration` cmdlet to upload the configuration into your Automation account:

```
Import-AzureRmAutomationDscConfiguration -SourcePath 'C:\DscConfigs\TestConfig.ps1' -ResourceGroupName 'MyResourceGroup' -AutomationAccountName 'myAutomationAccount' -Published
```

Compile a configuration into a node configuration

A DSC configuration must be compiled into a node configuration before it can be assigned to a node.

For information about compiling configurations, see [DSC configurations](#).

Call the `Start-AzureRmAutomationDscCompilationJob` cmdlet to compile the `TestConfig` configuration into a node configuration:

```
Start-AzureRmAutomationDscCompilationJob -ConfigurationName 'TestConfig' -ResourceGroupName 'MyResourceGroup' -AutomationAccountName 'myAutomationAccount'
```

This creates a node configuration named `TestConfig.WebServer` in your Automation account.

Register a VM to be managed by DSC

You can use Azure Automation DSC to manage Azure VMs (both Classic and Resource Manager), on-premises VMs, Linux machines, AWS VMs, and on-premises physical machines. In this topic, we cover how to register only Azure Resource Manager VMs. For information about registering other types of machines, see [Onboarding machines for management by Azure Automation DSC](#).

Call the `Register-AzureRmAutomationDscNode` cmdlet to register your VM with Azure Automation DSC.

```
Register-AzureRmAutomationDscNode -ResourceGroupName "MyResourceGroup" -AutomationAccountName "myAutomationAccount" -AzureVMName "DscVm"
```

This registers the specified VM as a DSC node in your Azure Automation account.

Specify configuration mode settings

When you register a VM as a managed node, you can also specify properties of the configuration. For example, you can specify that the state of the machine is to be applied only once (DSC does not attempt to apply the configuration after the initial check) by specifying `ApplyOnly` as the value of the **ConfigurationMode** property:

```
Register-AzureRmAutomationDscNode -ResourceGroupName 'MyResourceGroup' -AutomationAccountName 'myAutomationAccount' -AzureVMName "DscVm" -ConfigurationMode 'ApplyOnly'
```

You can also specify how often DSC checks the configuration state by using the **ConfigurationModeFrequencyMins** property:

```
# Run a DSC check every 60 minutes
Register-AzureRmAutomationDscNode -ResourceGroupName 'MyResourceGroup' -AutomationAccountName 'myAutomationAccount' -AzureVMName "DscVm" -ConfigurationModeFrequencyMins 60
```

For more information about setting configuration properties for a managed node, see [Register-AzureRmAutomationDscNode](#).

For more information about DSC configuration settings, see [Configuring the Local Configuration Manager](#).

Assign a node configuration to a managed node

Now we can assign the compiled node configuration to the VM we want to configure.

```
# Get the ID of the DSC node
$node = Get-AzureRmAutomationDscNode -ResourceGroupName 'MyResourceGroup' -AutomationAccountName
'myAutomationAccount' -Name 'DscVm'

# Assign the node configuration to the DSC node
Set-AzureRmAutomationDscNode -ResourceGroupName 'MyResourceGroup' -AutomationAccountName 'myAutomationAccount'
-NodeConfigurationName 'TestConfig.WebServer' -Id $node.Id
```

This assigns the node configuration named `TestConfig.WebServer` to the registered DSC node named `DscVm`. By default, the DSC node is checked for compliance with the node configuration every 30 minutes. For information about how to change the compliance check interval, see [Configuring the Local Configuration Manager](#)

Check the compliance status of a managed node

You can get reports on the compliance status of a DSC node by calling the `Get-AzureRmAutomationDscNodeReport` cmdlet:

```
# Get the ID of the DSC node
$node = Get-AzureRmAutomationDscNode -ResourceGroupName 'MyResourceGroup' -AutomationAccountName
'myAutomationAccount' -Name 'DscVm'

# Get an array of status reports for the DSC node
$reports = Get-AzureRmAutomationDscNodeReport -ResourceGroupName 'MyResourceGroup' -AutomationAccountName
'myAutomationAccount' -Id $node.Id

# Display the most recent report
$reports[0]
```

Next steps

- To learn how to onboard nodes to be managed with Azure Automation DSC, see [Onboarding machines for management by Azure Automation DSC](#)
- To learn how to use the Azure portal to use Automation DSC, see [Getting started with Azure Automation DSC](#)
- To learn about compiling DSC configurations so that you can assign them to target nodes, see [Compiling configurations in Azure Automation DSC](#)
- For PowerShell cmdlet reference for Azure Automation DSC, see [Azure Automation DSC cmdlets](#)
- For pricing information, see [Azure Automation DSC pricing](#)
- To see an example of using Azure Automation DSC in a continuous deployment pipeline, see [Continuous Deployment to IaaS VMs Using Azure Automation DSC and Chocolatey](#)

Onboarding machines for management by Azure Automation DSC

5/21/2018 • 14 minutes to read • [Edit Online](#)

Why manage machines with Azure Automation DSC?

Like [PowerShell Desired State Configuration](#), Azure Automation Desired State Configuration is a simple, yet powerful, configuration management service for DSC nodes (physical and virtual machines) in any cloud or on-premises datacenter. It enables scalability across thousands of machines quickly and easily from a central, secure location. You can easily onboard machines, assign them declarative configurations, and view reports showing each machine's compliance to the desired state you specified. The Azure Automation DSC management layer is to DSC what the Azure Automation management layer is to PowerShell scripting. In other words, in the same way that Azure Automation helps you manage PowerShell scripts, it also helps you manage DSC configurations. To learn more about the benefits of using Azure Automation DSC, see [Azure Automation DSC overview](#).

Azure Automation DSC can be used to manage a variety of machines:

- Azure virtual machines (classic)
- Azure virtual machines
- Amazon Web Services (AWS) virtual machines
- Physical/virtual Windows machines on-premises, or in a cloud other than Azure/AWS
- Physical/virtual Linux machines on-premises, in Azure, or in a cloud other than Azure

In addition, if you are not ready to manage machine configuration from the cloud, Azure Automation DSC can also be used as a report-only endpoint. This allows you to set (push) desired configuration through DSC on-premises and view rich reporting details on node compliance with the desired state in Azure Automation.

NOTE

Managing Azure VMs with DSC is included at no extra charge if the virtual machine DSC extension installed is greater than 2.70. Refer to the [Automation pricing page](#) for more details.

The following sections outline how you can onboard each type of machine to Azure Automation DSC.

Azure virtual machines (classic)

With Azure Automation DSC, you can easily onboard Azure virtual machines (classic) for configuration management using either the Azure portal, or PowerShell. Under the hood, and without an administrator having to remote into the VM, the Azure VM Desired State Configuration extension registers the VM with Azure Automation DSC. Since the Azure VM Desired State Configuration extension runs asynchronously, steps to track its progress or troubleshoot it are provided in the following [Troubleshooting Azure virtual machine onboarding](#) section.

Azure portal

In the [Azure portal](#), click **Browse -> Virtual machines (classic)**. Select the Windows VM you want to onboard. On the virtual machine's dashboard blade, click **All settings -> Extensions -> Add -> Azure Automation DSC -> Create**. Enter the [PowerShell DSC Local Configuration Manager values](#) required for your use case, your Automation account's registration key and registration URL, and optionally a node configuration to assign to the VM.

The screenshot shows two windows side-by-side. On the left is the 'Azure Automation DSC' extension page from Microsoft Corp., which includes a brief description, social sharing links, publisher information, useful links, and a 'Create' button. On the right is the 'Add Extension' configuration dialog, which contains fields for Registration URL, Registration Key, Node Configuration Name, Refresh Frequency, Configuration Mode Frequency, Configuration Mode, and Action after Reboot, along with an 'OK' button at the bottom.

Azure Automation DSC

Microsoft Corp.

Azure Automation DSC brings the same management layer to PowerShell Desired State Configuration as Azure Automation offers for PowerShell Workflows today.

Azure Automation DSC allows you to author and manage PowerShell Desired State Configurations, import DSC Resources, and generate DSC Node Configurations (MOF documents), all in the cloud. These DSC items will be placed on the Azure Automation DSC pull server so that target nodes in the cloud or on-premises can pick them up, automatically conform to the desired state they specify, and report back on their compliance with the desired state to Azure Automation.

Adding the Azure Automation DSC extension to your Azure VM will automatically onboard the VM for configuration management using Azure Automation DSC.

PUBLISHER Microsoft Corp.

USEFUL LINKS [Learn more about Azure Automation DSC](#) [Learn more about PowerShell DSC](#)

Create

Add Extension

* Registration URL

* Registration Key

Node Configuration Name

Refresh Frequency

Configuration Mode Frequency

Configuration Mode

Action after Reboot

OK

To find the registration URL and key for the Automation account to onboard the machine to, see the following [Secure registration](#) section:

PowerShell

```

# log in to both Azure Service Management and Azure Resource Manager
Add-AzureAccount
Connect-AzureRmAccount

# fill in correct values for your VM/Automation account here
$VMName = ""
$ServiceName = ""
$AutomationAccountName = ""
$AutomationAccountResourceGroup = ""

# fill in the name of a Node Configuration in Azure Automation DSC, for this VM to conform to
# NOTE: DSC Node Configuration names are case sensitive in the portal.
$NodeConfigName = ""

# get Azure Automation DSC registration info
$Account = Get-AzureRmAutomationAccount -ResourceGroupName $AutomationAccountResourceGroup -Name
$AutomationAccountName
$RegistrationInfo = $Account | Get-AzureRmAutomationRegistrationInfo

# use the DSC extension to onboard the VM for management with Azure Automation DSC
$VM = Get-AzureVM -Name $VMName -ServiceName $ServiceName

$PublicConfiguration = ConvertTo-Json -Depth 8 @{
    SasToken = ""
    ModulesUrl =
"https://eus2oaasibizamarketprod1.blob.core.windows.net/automationdscpreview/RegistrationMetaConfigV2.zip"
    ConfigurationFunction = "RegistrationMetaConfigV2.ps1\RegistrationMetaConfigV2"

# update these PowerShell DSC Local Configuration Manager defaults if they do not match your use case.
# See https://technet.microsoft.com/library/dn249922.aspx?f=255&MSPPError=-2147217396 for more details
    Properties = @{
        RegistrationKey = @{
            UserName = 'notused'
            Password = 'PrivateSettingsRef:RegistrationKey'
        }
        RegistrationUrl = $RegistrationInfo.Endpoint
        NodeConfigurationName = $NodeConfigName
        ConfigurationMode = "ApplyAndMonitor"
        ConfigurationModeFrequencyMins = 15
        RefreshFrequencyMins = 30
        RebootNodeIfNeeded = $False
        ActionAfterReboot = "ContinueConfiguration"
        AllowModuleOverwrite = $False
    }
}

$PrivateConfiguration = ConvertTo-Json -Depth 8 @{
    Items = @{
        RegistrationKey = $RegistrationInfo.PrimaryKey
    }
}

$VM = Set-AzureVMEExtension `

-VM $vm `

-Publisher Microsoft.PowerShell `

-ExtensionName DSC `

-Version 2.19 `

-PublicConfiguration $PublicConfiguration `

-PrivateConfiguration $PrivateConfiguration `

-ForceUpdate

$VM | Update-AzureVM

```

NOTE

Dsc Node Configuration names are case sensitive in the portal. If the case is mismatched the node will not show up under DSC Nodes.

Azure virtual machines

Azure Automation DSC lets you easily onboard Azure virtual machines for configuration management, using either the Azure portal, Azure Resource Manager templates, or PowerShell. Under the hood, and without an administrator having to remote into the VM, the Azure VM Desired State Configuration extension registers the VM with Azure Automation DSC. Since the Azure VM Desired State Configuration extension runs asynchronously, steps to track its progress or troubleshoot it are provided in the following [Troubleshooting Azure virtual machine onboarding](#) section.

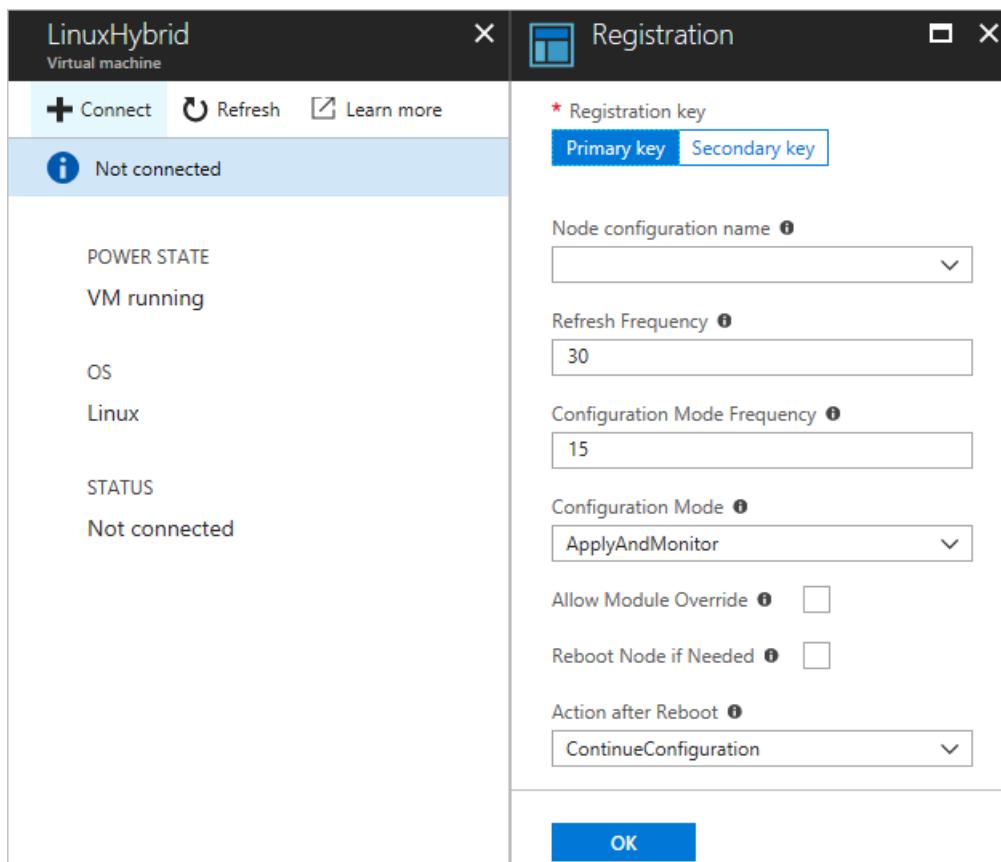
Azure portal

In the [Azure portal](#), navigate to the Azure Automation account where you want to onboard virtual machines. On the Automation account dashboard, click **DSC Nodes** -> **+ Add Azure VM**.

Select an Azure virtual machine to onboard.

If the machine does not have the PowerShell desired state extension installed and the power state is running, click **Connect**.

Under **Registration**, enter the [PowerShell DSC Local Configuration Manager values](#) required for your use case, and optionally a node configuration to assign to the VM.



Azure Resource Manager templates

Azure virtual machines can be deployed and onboarded to Azure Automation DSC via Azure Resource Manager templates. See [Configure a VM via DSC extension](#) and [Azure Automation DSC](#) for an example template that onboards an existing VM to Azure Automation DSC. To find the registration key and registration URL taken as input in this template, see the following [Secure registration](#) section.

PowerShell

The [Register-AzureRmAutomationDscNode](#) cmdlet can be used to onboard virtual machines in the Azure portal via PowerShell.

Amazon Web Services (AWS) virtual machines

You can easily onboard Amazon Web Services virtual machines for configuration management by Azure Automation DSC using the AWS DSC Toolkit. You can learn more about the toolkit [here](#).

Physical/virtual Windows machines on-premises, or in a cloud other than Azure/AWS

On-premises Windows machines and Windows machines in non-Azure clouds (such as Amazon Web Services) can also be onboarded to Azure Automation DSC, as long as they have outbound access to the internet, via a few simple steps:

1. Make sure the latest version of [WMF 5](#) is installed on the machines you want to onboard to Azure Automation DSC.
2. Follow the directions in following section [Generating DSC metaconfigurations](#) to generate a folder containing the needed DSC metaconfigurations.
3. Remotely apply the PowerShell DSC metaconfiguration to the machines you want to onboard. **The machine this command is run from must have the latest version of WMF 5 installed:**

```
Set-DscLocalConfigurationManager -Path C:\Users\joe\Desktop\DscMetaConfigs -ComputerName MyServer1, MyServer2
```

4. If you cannot apply the PowerShell DSC metaconfigurations remotely, copy the metaconfigurations folder from step 2 onto each machine to onboard. Then call **Set-DscLocalConfigurationManager** locally on each machine to onboard.
5. Using the Azure portal or cmdlets, check that the machines to onboard now show up as DSC nodes registered in your Azure Automation account.

Physical/virtual Linux machines on-premises, in Azure, or in a cloud other than Azure

On-premises Linux machines, Linux machines in Azure, and Linux machines in non-Azure clouds can also be onboarded to Azure Automation DSC, as long as they have outbound access to the internet, via a few simple steps:

1. Make sure the latest version of [PowerShell Desired State Configuration for Linux](#) is installed on the machines you want to onboard to Azure Automation DSC.
2. If the [PowerShell DSC Local Configuration Manager defaults](#) match your use case, and you want to onboard machines such that they **both** pull from and report to Azure Automation DSC:
 - On each Linux machine to onboard to Azure Automation DSC, use Register.py to onboard using the PowerShell DSC Local Configuration Manager defaults:

```
/opt/microsoft/dsc/Scripts/Register.py <Automation account registration key> <Automation account registration URL>
```

- To find the registration key and registration URL for your Automation account, see the following [Secure registration](#) section.

If the PowerShell DSC Local Configuration Manager defaults **do not** match your use case, or you

want to onboard machines such that they only report to Azure Automation DSC, but do not pull configuration or PowerShell modules from it, follow steps 3 - 6. Otherwise, proceed directly to step 6.

3. Follow the directions in the following **Generating DSC metaconfigurations** section to generate a folder containing the needed DSC metaconfigurations.
4. Remotely apply the PowerShell DSC metaconfiguration to the machines you want to onboard:

```
$SecurePass = ConvertTo-SecureString -String "<root password>" -AsPlainText -Force  
$Cred = New-Object System.Management.Automation.PSCredential "root", $SecurePass  
$Opt = New-CimSessionOption -UseSsl -SkipCACheck -SkipCNCheck -SkipRevocationCheck  
  
# need a CimSession for each Linux machine to onboard  
  
$Session = New-CimSession -Credential $Cred -ComputerName <your Linux machine> -Port 5986 -  
Authentication basic -SessionOption $Opt  
  
Set-DscLocalConfigurationManager -CimSession $Session -Path C:\Users\joe\Desktop\DscMetaConfigs
```

The machine this command is run from must have the latest version of [WMF 5](#) installed.

1. If you cannot apply the PowerShell DSC metaconfigurations remotely, for each Linux machine to onboard, copy the metaconfiguration corresponding to that machine from the folder in step 5 onto the Linux machine. Then call `SetDscLocalConfigurationManager.py` locally on each Linux machine you want to onboard to Azure Automation DSC:

```
/opt/microsoft/dsc/Scripts/SetDscLocalConfigurationManager.py -configurationmof <path to  
metaconfiguration file>
```

2. Using the Azure portal or cmdlets, check that the machines to onboard now show up as DSC nodes registered in your Azure Automation account.

Generating DSC metaconfigurations

To generically onboard any machine to Azure Automation DSC, a [DSC metaconfiguration](#) can be generated that, when applied, tells the DSC agent on the machine to pull from and/or report to Azure Automation DSC. DSC metaconfigurations for Azure Automation DSC can be generated using either a PowerShell DSC configuration, or the Azure Automation PowerShell cmdlets.

NOTE

DSC metaconfigurations contain the secrets needed to onboard a machine to an Automation account for management. Make sure to properly protect any DSC metaconfigurations you create, or delete them after use.

Using a DSC Configuration

1. Open the PowerShell ISE as an administrator in a machine in your local environment. The machine must have the latest version of [WMF 5](#) installed.
2. Copy the following script locally. This script contains a PowerShell DSC configuration for creating metaconfigurations, and a command to kick off the metaconfiguration creation.

NOTE

Dsc Node Configuration names are case sensitive in the portal. If the case is mismatched the node will not show up under DSC Nodes.

```

```powershell
The DSC configuration that will generate metaconfigurations
[DscLocalConfigurationManager()]
Configuration DscMetaConfigs
{

 param
 (
 [Parameter(Mandatory=$True)]
 [String]$RegistrationUrl,

 [Parameter(Mandatory=$True)]
 [String]$RegistrationKey,

 [Parameter(Mandatory=$True)]
 [String[]]$ComputerName,

 [Int]$RefreshFrequencyMins = 30,

 [Int]$ConfigurationModeFrequencyMins = 15,

 [String]$ConfigurationMode = "ApplyAndMonitor",

 [String]$NodeConfigurationName,

 [Boolean]$RebootNodeIfNeeded= $False,

 [String]$ActionAfterReboot = "ContinueConfiguration",

 [Boolean]$AllowModuleOverwrite = $False,

 [Boolean]$ReportOnly
)

 if (!$NodeConfigurationName -or $NodeConfigurationName -eq "")
 {
 $ConfigurationNames = $null
 }
 else
 {
 $ConfigurationNames = @($NodeConfigurationName)
 }

 if ($ReportOnly)
 {
 $RefreshMode = "PUSH"
 }
 else
 {
 $RefreshMode = "PULL"
 }

 Node $ComputerName
 {

 Settings
 {
 RefreshFrequencyMins = $RefreshFrequencyMins
 RefreshMode = $RefreshMode
 ConfigurationMode = $ConfigurationMode
 AllowModuleOverwrite = $AllowModuleOverwrite
 RebootNodeIfNeeded = $RebootNodeIfNeeded
 ActionAfterReboot = $ActionAfterReboot
 ConfigurationModeFrequencyMins = $ConfigurationModeFrequencyMins
 }

 if (!$ReportOnly)
 {

```

```

ConfigurationRepositoryWeb AzureAutomationDSC
{
 ServerUrl = $RegistrationUrl
 RegistrationKey = $RegistrationKey
 ConfigurationNames = $ConfigurationNames
}

ResourceRepositoryWeb AzureAutomationDSC
{
 ServerUrl = $RegistrationUrl
 RegistrationKey = $RegistrationKey
}
}

ReportServerWeb AzureAutomationDSC
{
 ServerUrl = $RegistrationUrl
 RegistrationKey = $RegistrationKey
}
}

Create the metaconfigurations
NOTE: DSC Node Configuration names are case sensitive in the portal.
TODO: edit the below as needed for your use case
$Params = @{
 RegistrationUrl = '<fill me in>';
 RegistrationKey = '<fill me in>';
 ComputerName = @('<some VM to onboard>', '<some other VM to onboard>');
 NodeConfigurationName = 'SimpleConfig.webserver';
 RefreshFrequencyMins = 30;
 ConfigurationModeFrequencyMins = 15;
 RebootNodeIfNeeded = $False;
 AllowModuleOverwrite = $False;
 ConfigurationMode = 'ApplyAndMonitor';
 ActionAfterReboot = 'ContinueConfiguration';
 ReportOnly = $False; # Set to $True to have machines only report to AA DSC but not pull from it
}

Use PowerShell splatting to pass parameters to the DSC configuration being invoked
For more info about splatting, run: Get-Help -Name about_Splatting
DscMetaConfigs @Params
```

```

1. Fill in the registration key and URL for your Automation account, as well as the names of the machines to onboard. All other parameters are optional. To find the registration key and registration URL for your Automation account, see the following [Secure registration](#) section.
2. If you want the machines to report DSC status information to Azure Automation DSC, but not pull configuration or PowerShell modules, set the **ReportOnly** parameter to true.
3. Run the script. You should now have a folder called **DscMetaConfigs** in your working directory, containing the PowerShell DSC metaconfigurations for the machines to onboard (as an administrator):

```
Set-DscLocalConfigurationManager -Path ./DscMetaConfigs
```

Using the Azure Automation cmdlets

If the PowerShell DSC Local Configuration Manager defaults match your use case, and you want to onboard machines such that they both pull from and report to Azure Automation DSC, the Azure Automation cmdlets provide a simplified method of generating the DSC metaconfigurations needed:

1. Open the PowerShell console or PowerShell ISE as an administrator in a machine in your local environment.
2. Connect to Azure Resource Manager using **Connect-AzureRmAccount**

3. Download the PowerShell DSC metaconfigurations for the machines you want to onboard from the Automation account to which you want to onboard nodes:

```
# Define the parameters for Get-AzureRmAutomationDscOnboardingMetaconfig using PowerShell Splatting
$Params = @{

    ResourceGroupName = 'ContosoResources'; # The name of the ARM Resource Group that contains your
    Azure Automation Account
    AutomationAccountName = 'ContosoAutomation'; # The name of the Azure Automation Account where you
    want a node on-boarded to
    ComputerName = @('web01', 'web02', 'sql01'); # The names of the computers that the meta
    configuration will be generated for
    OutputFolder = "$env:UserProfile\Desktop\";

}

# Use PowerShell splatting to pass parameters to the Azure Automation cmdlet being invoked
# For more info about splatting, run: Get-Help -Name about_Splatting
Get-AzureRmAutomationDscOnboardingMetaconfig @Params
```

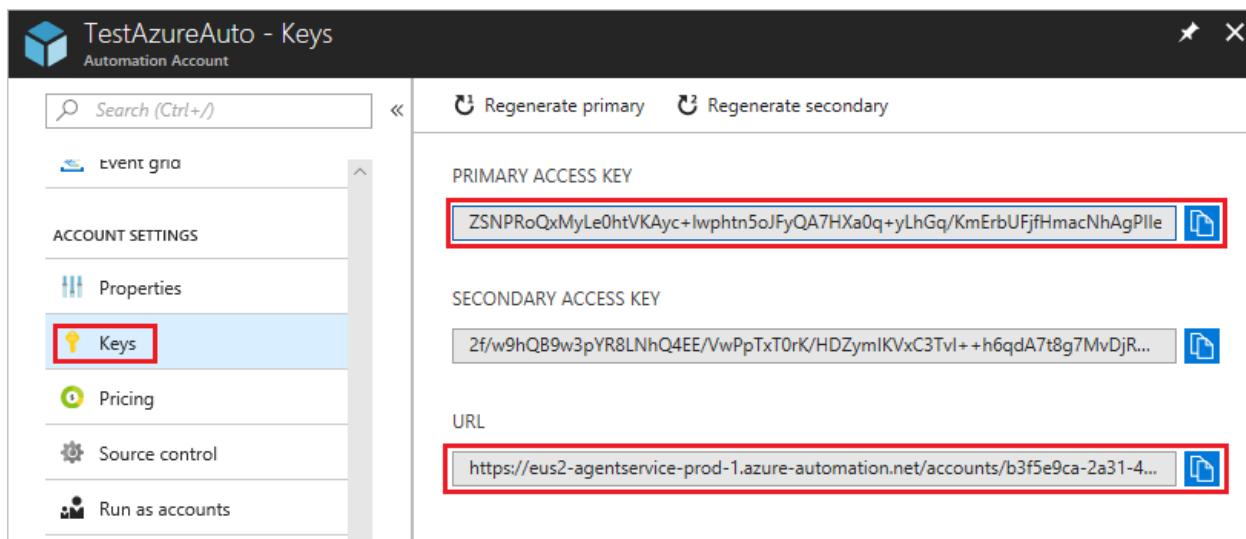
4. You should now have a folder called **DscMetaConfigs**, containing the PowerShell DSC metaconfigurations for the machines to onboard (as an administrator):

```
Set-DscLocalConfigurationManager -Path $env:UserProfile\Desktop\DscMetaConfigs
```

Secure registration

Machines can securely onboard to an Azure Automation account through the WMF 5 DSC registration protocol, which allows a DSC node to authenticate to a PowerShell DSC Pull or Reporting server (including Azure Automation DSC). The node registers to the server at a **Registration URL**, authenticating using a **Registration key**. During registration, the DSC node and DSC Pull/Reporting server negotiate a unique certificate for this node to use for authentication to the server post-registration. This process prevents onboarded nodes from impersonating one another, such as if a node is compromised and behaving maliciously. After registration, the Registration key is not used for authentication again, and is deleted from the node.

You can get the information required for the DSC registration protocol from **Keys** under **Account Settings** in the Azure portal. Open this blade by clicking the key icon on the **Essentials** panel for the Automation account.



- Registration URL is the URL field in the Manage Keys blade.
- Registration key is the Primary Access Key or Secondary Access Key in the Manage Keys blade. Either key can be used.

For added security, the primary and secondary access keys of an Automation account can be regenerated at any

time (on the **Manage Keys** blade) to prevent future node registrations using previous keys.

Troubleshooting Azure virtual machine onboarding

Azure Automation DSC lets you easily onboard Azure Windows VMs for configuration management. Under the hood, the Azure VM Desired State Configuration extension is used to register the VM with Azure Automation DSC. Since the Azure VM Desired State Configuration extension runs asynchronously, tracking its progress and troubleshooting its execution may be important.

NOTE

Any method of onboarding an Azure Windows VM to Azure Automation DSC that uses the Azure VM Desired State Configuration extension could take up to an hour for the node to show up as registered in Azure Automation. This is due to the installation of Windows Management Framework 5.0 on the VM by the Azure VM DSC extension, which is required to onboard the VM to Azure Automation DSC.

To troubleshoot or view the status of the Azure VM Desired State Configuration extension, in the Azure portal navigate to the VM being onboarded, then click **Extensions** under **Settings**. Then click **DSC** or **DSCForLinux** depending on your operating system. For more details, you can click **View detailed status**.

Certificate expiration and reregistration

After registering a machine as a DSC node in Azure Automation DSC, there are a number of reasons why you may need to reregister that node in the future:

- After registering, each node automatically negotiates a unique certificate for authentication that expires after one year. Currently, the PowerShell DSC registration protocol cannot automatically renew certificates when they are nearing expiration, so you need to reregister the nodes after a year's time. Before reregistering, ensure that each node is running Windows Management Framework 5.0 RTM. If a node's authentication certificate expires, and the node is not reregistered, the node is unable to communicate with Azure Automation and is marked 'Unresponsive.' Reregistration performed 90 days or less from the certificate expiration time, or at any point after the certificate expiration time, will result in a new certificate being generated and used.
- To change any [PowerShell DSC Local Configuration Manager values](#) that were set during initial registration of the node, such as ConfigurationMode. Currently, these DSC agent values can only be changed through reregistration. The one exception is the Node Configuration assigned to the node -- this can be changed in Azure Automation DSC directly.

Reregistration can be performed in the same way you registered the node initially, using any of the onboarding methods described in this document. You do not need to unregister a node from Azure Automation DSC before reregistering it.

Related Articles

- [Azure Automation DSC overview](#)
- [Azure Automation DSC cmdlets](#)
- [Azure Automation DSC pricing](#)

Compiling configurations in Azure Automation DSC

7/6/2018 • 7 minutes to read • [Edit Online](#)

You can compile Desired State Configuration (DSC) configurations in two ways with Azure Automation: in the Azure portal, and with Windows PowerShell. The following table helps you determine when to use which method based on the characteristics of each:

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 Windows 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

Once you have decided on a compilation method, use the following procedures to start compiling.

Compiling a DSC Configuration with the Azure portal

1. From your Automation account, click **DSC Configurations**.
2. Click a configuration to open its blade.
3. Click **Compile**.
4. If the configuration has no parameters, you are prompted to confirm whether you want to compile it. If the configuration has parameters, the **Compile Configuration** blade opens so you can provide parameter values. See the following **Basic Parameters** section for further details on parameters.
5. The **Compilation Job** blade is opened so that you can track the compilation job's status, and the node configurations (MOF configuration documents) it caused to be placed on the Azure Automation DSC Pull Server.

Compiling a DSC Configuration with Windows PowerShell

You can use `Start-AzureRmAutomationDscCompilationJob` to start compiling with Windows PowerShell. The following sample code starts compilation of a DSC configuration called **SampleConfig**.

```
Start-AzureRmAutomationDscCompilationJob -ResourceGroupName "MyResourceGroup" -AutomationAccountName "MyAutomationAccount" -ConfigurationName "SampleConfig"
```

`Start-AzureRmAutomationDscCompilationJob` returns a compilation job object that you can use to track its status. You can then use this compilation job object with `Get-AzureRmAutomationDscCompilationJob` to determine the status of the compilation job, and `Get-AzureRmAutomationDscCompilationJobOutput` to view its streams (output). The following sample code starts compilation of the **SampleConfig** configuration, waits until it has completed, and then

displays its streams.

```
$CompilationJob = Start-AzureRmAutomationDscCompilationJob -ResourceGroupName "MyResourceGroup" -  
AutomationAccountName "MyAutomationAccount" -ConfigurationName "SampleConfig"  
  
while($CompilationJob.EndTime -eq $null -and $CompilationJob.Exception -eq $null)  
{  
    $CompilationJob = $CompilationJob | Get-AzureRmAutomationDscCompilationJob  
    Start-Sleep -Seconds 3  
}  
  
$CompilationJob | Get-AzureRmAutomationDscCompilationJobOutput -Stream Any
```

Basic Parameters

Parameter declaration in DSC configurations, including parameter types and properties, works the same as in Azure Automation runbooks. See [Starting a runbook in Azure Automation](#) to learn more about runbook parameters.

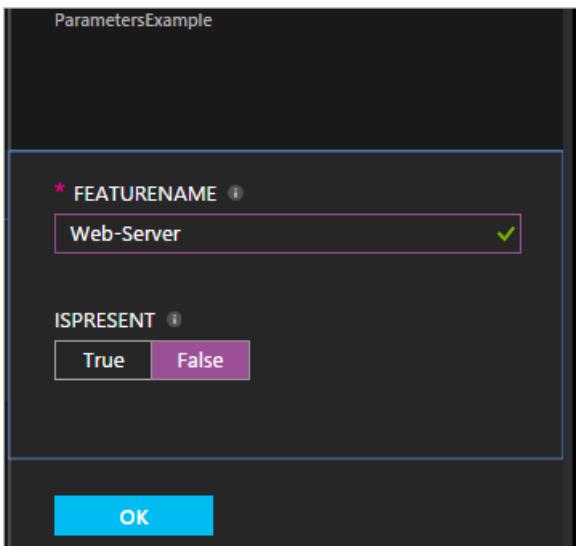
The following example uses two parameters called **FeatureName** and **IsPresent**, to determine the values of properties in the **ParametersExample.sample** node configuration, generated during compilation.

```
Configuration ParametersExample  
{  
    param(  
        [Parameter(Mandatory=$true)]  
  
        [string] $FeatureName,  
  
        [Parameter(Mandatory=$true)]  
        [boolean] $IsPresent  
    )  
  
    $EnsureString = "Present"  
    if($IsPresent -eq $false)  
    {  
        $EnsureString = "Absent"  
    }  
  
    Node "sample"  
    {  
        WindowsFeature ($FeatureName + "Feature")  
        {  
            Ensure = $EnsureString  
            Name = $FeatureName  
        }  
    }  
}
```

You can compile DSC Configurations that use basic parameters in the Azure Automation DSC portal, or with Azure PowerShell:

Portal

In the portal, you can enter parameter values after clicking **Compile**.



PowerShell

PowerShell requires parameters in a **hashtable** where the key matches the parameter name, and the value equals the parameter value.

```
$Parameters = @{
    "FeatureName" = "Web-Server"
    "IsPresent" = $False
}

Start-AzureRmAutomationDscCompilationJob -ResourceGroupName "MyResourceGroup" -AutomationAccountName
"MyAutomationAccount" -ConfigurationName "ParametersExample" -Parameters $Parameters
```

For information about passing PSCredentials as parameters, see [Credential Assets](#) below.

Composite Resources

Composite Resources allow you to use DSC configurations as nested resources inside of a configuration. This enables you to apply multiple configurations to a single resource. See [Composite resources: Using a DSC configuration as a resource](#) to learn more about **Composite Resources**

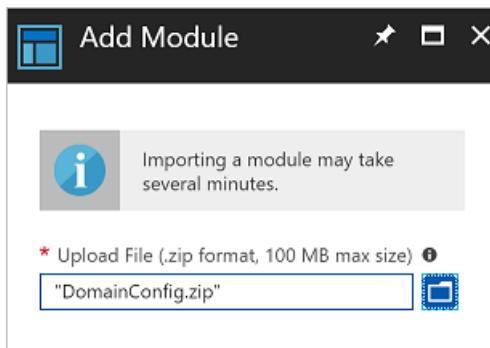
NOTE

In order for **Composite Resources** to compile correctly, you must first ensure that any DSC Resources that the composite relies on are first installed in the Azure Automation Account Modules repository or it doesn't import properly.

To add a DSC **Composite Resource**, you must add the resource module to an archive (*.zip). Go to the Modules repository on your Azure Automation Account. Then click on the 'Add a Module' button.

| Add a module | Update Azure Modules | Browse gallery | Refresh |
|---------------|----------------------|----------------|---------------------|
| NAME | | | LAST MODIFIED |
| Azure | | | 7/21/2017, 10:52 AM |
| Azure.Storage | | | 7/21/2017, 10:49 AM |

Navigate to the directory where your archive is located. Select the archive file, and click OK.



You are taken back to the modules directory, where you can monitor the status of your **Composite Resource** while it unpacks and registers with Azure Automation.

| | | | |
|--------------|--------------------|-------------------------|---------|
| DomainConfig | 12/1/2017, 2:22 PM | Importing newer version | 1.0.0.0 |
|--------------|--------------------|-------------------------|---------|

Once the module is registered, you can then click on it to validate that the **Composite Resources** are now available to be used in a configuration.

| NAME | DESCRIPTION |
|------------------|-------------|
| DomainController | |
| JoinDomain | |

Then you can call the **Composite Resource** into your configuration like so:

```

Node ($AllNodes.Where{$_.Role -eq "WebServer"}).NodeName
{
    JoinDomain DomainJoin
    {
        DomainName = $DomainName
        Admincreds = $Admincreds
    }

    PSWAWebServer InstallPSWAWebServer
    {
        DependsOn = "[JoinDomain]DomainJoin"
    }
}

```

ConfigurationData

ConfigurationData allows you to separate structural configuration from any environment-specific configuration while using PowerShell DSC. See [Separating "What" from "Where" in PowerShell DSC](#) to learn more about **ConfigurationData**.

NOTE

You can use **ConfigurationData** when compiling in Azure Automation DSC using Azure PowerShell, but not in the Azure portal.

The following example DSC configuration uses **ConfigurationData** via the **\$ConfigurationData** and **\$AllNodes** keywords. You also need the [xWebAdministration module](#) for this example:

```

Configuration ConfigurationDataSample
{
    Import-DscResource -ModuleName xWebAdministration -Name MSFT_xWebsite

    Write-Verbose $ConfigurationData.NonNodeData.SomeMessage

    Node $AllNodes.Where{$_.Role -eq "WebServer"}.NodeName
    {
        xWebsite Site
        {
            Name = $Node.SiteName
            PhysicalPath = $Node.SiteContents
            Ensure   = "Present"
        }
    }
}

```

You can compile the preceding DSC configuration with PowerShell. The following PowerShell adds two node configurations to the Azure Automation DSC Pull Server: **ConfigurationDataSample.MyVM1** and **ConfigurationDataSample.MyVM3**:

```

$ConfigData = @{
    AllNodes = @(
        @{
            NodeName = "MyVM1"
            Role = "WebServer"
        },
        @{
            NodeName = "MyVM2"
            Role = "SQLServer"
        },
        @{
            NodeName = "MyVM3"
            Role = "WebServer"
        }
    )
}

NonNodeData = @{
    SomeMessage = "I love Azure Automation DSC!"
}

Start-AzureRmAutomationDscCompilationJob -ResourceGroupName "MyResourceGroup" -AutomationAccountName
"MyAutomationAccount" -ConfigurationName "ConfigurationDataSample" -ConfigurationData $ConfigData

```

Assets

Asset references are the same in Azure Automation DSC configurations and runbooks. See the following for more information:

- [Certificates](#)
- [Connections](#)
- [Credentials](#)
- [Variables](#)

Credential Assets

While DSC configurations in Azure Automation can reference credential assets using **Get-AutomationPSCredential**, credential assets can also be passed in via parameters, if desired. If a configuration takes a parameter of **PSCredential** type, then you need to pass the string name of an Azure Automation credential asset as that parameter's value, rather than a PSCredential object. Behind the scenes, the Azure Automation credential asset with that name is retrieved and passed to the configuration.

Keeping credentials secure in node configurations (MOF configuration documents) requires encrypting the credentials in the node configuration MOF file. However, currently you must tell PowerShell DSC it is okay for credentials to be outputted in plain text during node configuration MOF generation, because PowerShell DSC doesn't know that Azure Automation will be encrypting the entire MOF file after its generation via a compilation job.

You can tell PowerShell DSC that it is okay for credentials to be outputted in plain text in the generated node configuration MOFs using **ConfigurationData**. You should pass `PSDscAllowPlainTextPassword = $true` via **ConfigurationData** for each node block's name that appears in the DSC configuration and uses credentials.

The following example shows a DSC configuration that uses an Automation credential asset.

```

Configuration CredentialSample
{
    Import-DscResource -ModuleName PSDesiredStateConfiguration
    $Cred = Get-AutomationPSCredential "SomeCredentialAsset"

    Node $AllNodes.NodeName
    {
        File ExampleFile
        {
            SourcePath = "\\\$Server\share\path\file.ext"
            DestinationPath = "C:\destinationPath"
            Credential = $Cred
        }
    }
}

```

You can compile the preceding DSC configuration with PowerShell. The following PowerShell adds two node configurations to the Azure Automation DSC Pull Server: **CredentialSample.MyVM1** and **CredentialSample.MyVM2**.

```

$ConfigData = @{
    AllNodes = @(
        @{
            NodeName = "*"
            PSDscAllowPlainTextPassword = $True
        },
        @{
            NodeName = "MyVM1"
        },
        @{
            NodeName = "MyVM2"
        }
    )
}

Start-AzureRmAutomationDscCompilationJob -ResourceGroupName "MyResourceGroup" -AutomationAccountName
"MyAutomationAccount" -ConfigurationName "CredentialSample" -ConfigurationData $ConfigData

```

NOTE

When compilation is complete you may receive an error stating: **The 'Microsoft.PowerShell.Management' module was not imported because the 'Microsoft.PowerShell.Management' snap-in was already imported.** This warning can safely be ignored.

Importing node configurations

You can also import node configurations (MOFs) that you have compiled outside of Azure. One advantage of this is that node configurations can be signed. A signed node configuration is verified locally on a managed node by the DSC agent, ensuring that the configuration being applied to the node comes from an authorized source.

NOTE

You can use import signed configurations into your Azure Automation account, but Azure Automation does not currently support compiling signed configurations.

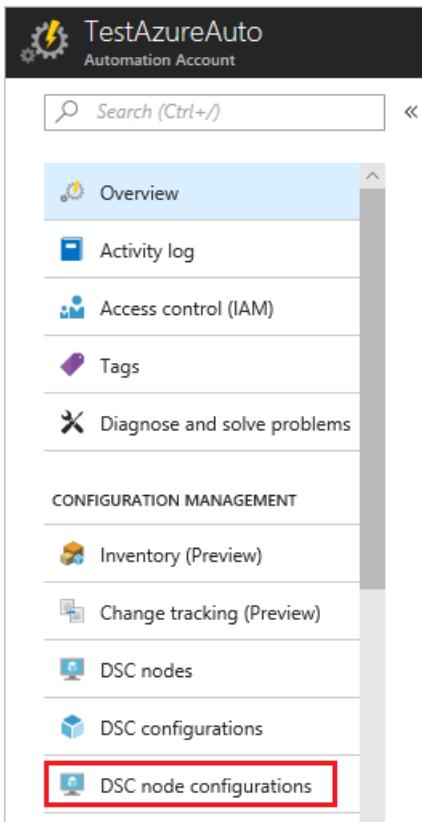
NOTE

A node configuration file must be no larger than 1 MB to allow it to be imported into Azure Automation.

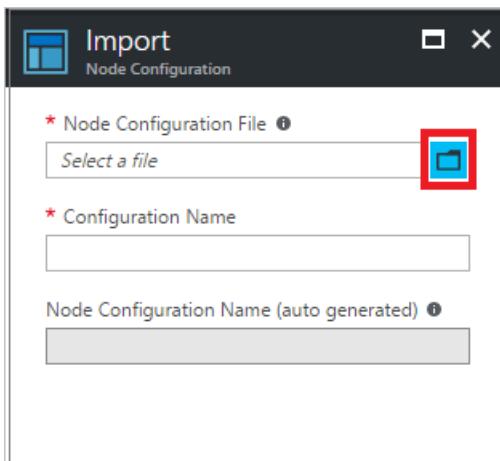
You can learn how to sign node configurations at <https://msdn.microsoft.com/powershell/wmf/5.1/dsc-improvements#how-to-sign-configuration-and-module>.

Importing a node configuration in the Azure portal

1. From your Automation account, click **DSC node configurations** under **Configuration Management**.



2. In the **DSC node configurations** blade, click **Add a NodeConfiguration**.
3. In the **Import** blade, click the folder icon next to the **Node Configuration File** textbox to browse for a node configuration file (MOF) on your local computer.



4. Enter a name in the **Configuration Name** textbox. This name must match the name of the configuration from which the node configuration was compiled.
5. Click **OK**.

Importing a node configuration with PowerShell

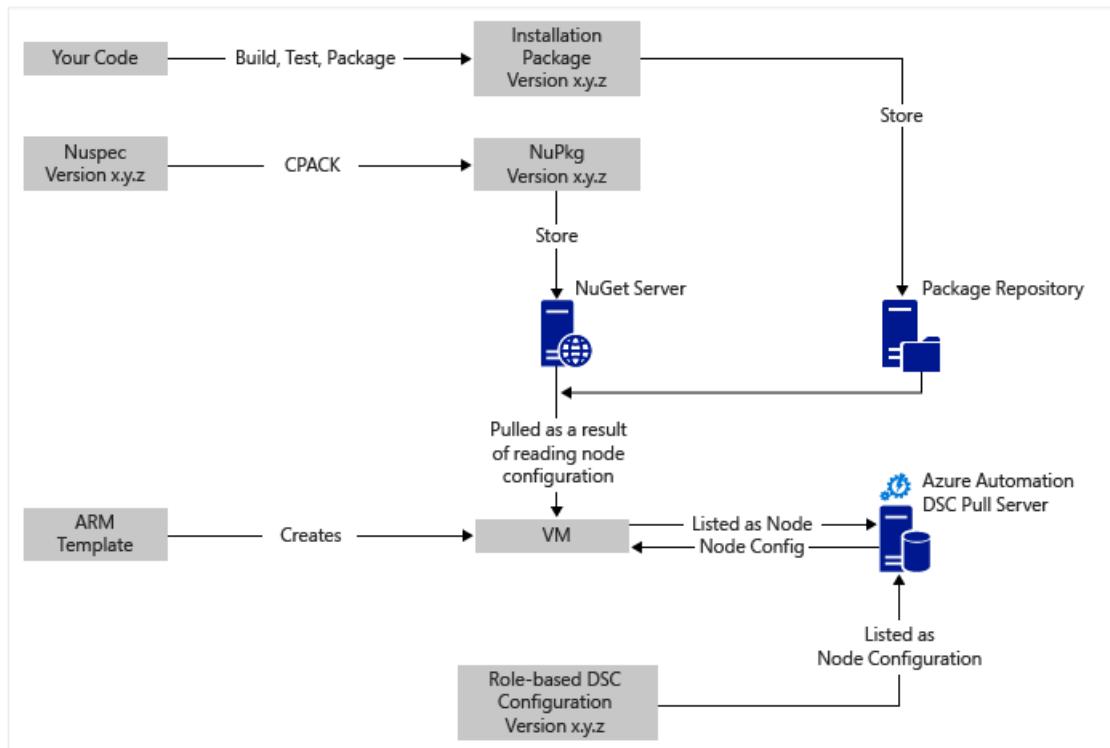
You can use the [Import-AzureRmAutomationDscNodeConfiguration](#) cmdlet to import a node configuration into your automation account.

```
Import-AzureRmAutomationDscNodeConfiguration -AutomationAccountName "MyAutomationAccount" -ResourceGroupName "MyResourceGroup" -ConfigurationName "MyNodeConfiguration" -Path "C:\MyConfigurations\TestVM1.mof"
```

Usage Example: Continuous deployment to Virtual Machines using Automation DSC and Chocolatey

5/21/2018 • 11 minutes to read • [Edit Online](#)

In a DevOps world there are many tools to assist with various points in the Continuous Integration pipeline. Azure Automation Desired State Configuration (DSC) is a welcome new addition to the options that DevOps teams can employ. This article demonstrates setting up Continuous Deployment (CD) for a Windows computer. You can easily extend the technique to include as many Windows computers as necessary in the role (a web site, for example), and from there to additional roles as well.



At a high level

There is quite a bit going on here, but fortunately it can be broken into two main processes:

- Writing code and testing it, then creating and publishing installation packages for major and minor versions of the system.
- Creating and managing VMs that will install and execute the code in the packages.

Once both of these core processes are in place, it's a short step to automatically update the package running on any particular VM as new versions are created and deployed.

Component overview

Package managers such as [apt-get](#) are pretty well known in the Linux world, but not so much in the Windows world. [Chocolatey](#) is such a thing, and Scott Hanselman's [blog](#) on the topic is a great intro. In a nutshell, Chocolatey allows you to install packages from a central repository of packages into a Windows system using the command line. You can create and manage your own repository, and Chocolatey can install packages from any number of repositories that you designate.

Desired State Configuration (DSC) ([overview](#)) is a PowerShell tool that allows you to declare the configuration that you want for a machine. For example, you can say, "I want Chocolatey installed, I want IIS installed, I want port 80 opened, I want version 1.0.0 of my website installed." The DSC Local Configuration Manager (LCM) implements that configuration. A DSC Pull Server holds a repository of configurations for your machines. The LCM on each machine checks in periodically to see if its configuration matches the stored configuration. It can either report status or attempt to bring the machine back into alignment with the stored configuration. You can edit the stored configuration on the pull server to cause a machine or set of machines to come into alignment with the changed configuration.

Azure Automation is a managed service in Microsoft Azure that allows you to automate various tasks using runbooks, nodes, credentials, resources and assets such as schedules and global variables. Azure Automation DSC extends this automation capability to include PowerShell DSC tools. Here's a great [overview](#).

A DSC Resource is a module of code that has specific capabilities, such as managing networking, Active Directory, or SQL Server. The Chocolatey DSC Resource knows how to access a NuGet Server (among others), download packages, install packages, and so on. There are many other DSC Resources in the [PowerShell Gallery](#). These modules are installed into your Azure Automation DSC Pull Server (by you) so they can be used by your configurations.

Resource Manager templates provide a declarative way of generating your infrastructure - things like networks, subnets, network security and routing, load balancers, NICs, VMs, and so on. Here's an [article](#) that compares the Resource Manager deployment model (declarative) with the Azure Service Management (ASM, or classic) deployment model (imperative), and discusses the core resource providers, compute, storage and network.

One key feature of an Resource Manager template is its ability to install a VM extension into the VM as it's provisioned. A VM extension has specific capabilities such as running a custom script, installing anti-virus software, or running a DSC configuration script. There are many other types of VM extensions.

Quick trip around the diagram

Starting at the top, you write your code, build and test, then create an installation package. Chocolatey can handle various types of installation packages, such as MSI, MSU, ZIP. And you have the full power of PowerShell to do the actual installation if Chocolatey's native capabilities aren't quite up to it. Put the package into someplace reachable – a package repository. This usage example uses a public folder in an Azure blob storage account, but it can be anywhere. Chocolatey works natively with NuGet servers and a few others for management of package metadata. [This article](#) describes the options. This usage example uses NuGet. A Nuspec is metadata about your packages. The Nuspec's are "compiled" into NuPkg's and stored in a NuGet server. When your configuration requests a package by name, and references a NuGet server, the Chocolatey DSC Resource (now on the VM) grabs the package and installs it for you. You can also request a specific version of a package.

In the bottom left portion of the picture, there is an Azure Resource Manager (ARM) template. In this usage example, the VM extension registers the VM with the Azure Automation DSC Pull Server (that is, a pull server) as a Node. The configuration is stored in the pull server. Actually, it's stored twice: once as plain text and once compiled as an MOF file (for those that know about such things.) In the portal, the MOF is a "node configuration" (as opposed to simply "configuration"). It's the artifact that's associated with a Node so the node will know its configuration. Details below show how to assign the node configuration to the node.

Presumably you're already doing the bit at the top, or most of it. Creating the nuspec, compiling and storing it in a NuGet server is a small thing. And you're already managing VMs. Taking the next step to continuous deployment requires setting up the pull server (once), registering your nodes with it (once), and creating and storing the configuration there (initially). Then as packages are upgraded and deployed to the repository, refresh the Configuration and Node Configuration in the pull server (repeat as needed).

If you're not starting with an ARM template, that's also OK. There are PowerShell cmdlets designed to help you register your VMs with the pull server and all of the rest. For more details, see this article: [Onboarding machines](#)

for management by Azure Automation DSC

Step 1: Setting up the pull server and automation account

At an authenticated (Connect-AzureRmAccount) PowerShell command line: (can take a few minutes while the pull server is set up)

```
New-AzureRmResourceGroup -Name MY-AUTOMATION-RG -Location MY-RG-LOCATION-IN-QUOTES  
New-AzureRmAutomationAccount -ResourceGroupName MY-AUTOMATION-RG -Location MY-RG-LOCATION-IN-QUOTES -Name MY-AUTOMATION-ACCOUNT
```

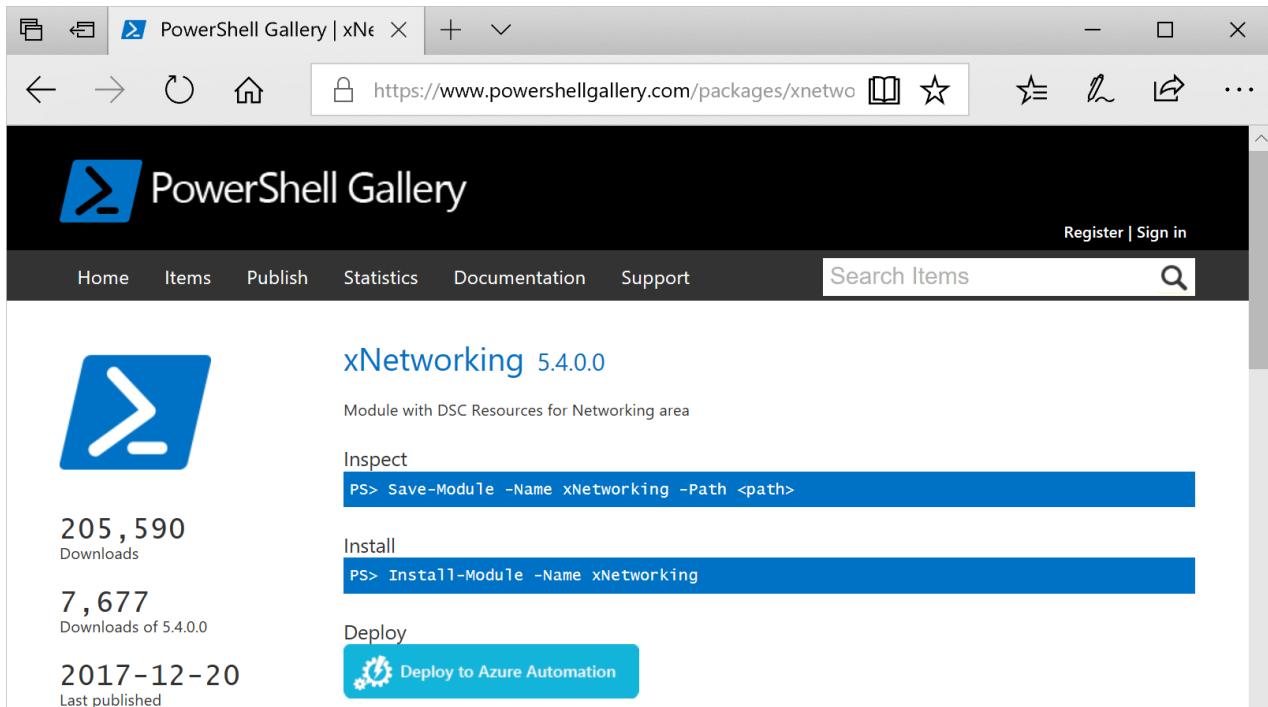
You can put your automation account into any of the following regions (aka location): East US 2, South Central US, US Gov Virginia, West Europe, Southeast Asia, Japan East, Central India and Australia Southeast, Canada Central, North Europe.

Step 2: VM extension tweaks to the ARM template

Details for VM registration (using the PowerShell DSC VM extension) provided in this [Azure Quickstart Template](#). This step registers your new VM with the pull server in the list of DSC Nodes. Part of this registration is specifying the node configuration to be applied to the node. This node configuration doesn't have to exist yet in the pull server, so it's OK that Step 4 is where this is done for the first time. But here in Step 2 you do need to have decided the name of the node and the name of the configuration. In this usage example, the node is 'isvbox' and the configuration is 'ISVBoxConfig'. So the node configuration name (to be specified in DeploymentTemplate.json) is 'ISVBoxConfig.isvbox'.

Step 3: Adding required DSC resources to the pull server

The PowerShell Gallery is instrumented to install DSC resources into your Azure Automation account. Navigate to the resource you want and click the "Deploy to Azure Automation" button.



Another technique recently added to the Azure Portal allows you to pull in new modules or update existing modules. Click through the Automation Account resource, the Assets tile, and finally the Modules tile. The Browse Gallery icon allows you to see the list of modules in the gallery, drill down into details and ultimately import into your Automation Account. This is a great way to keep your modules up to date from time to time. And, the import

feature checks dependencies with other modules to ensure nothing gets out of sync.

Or, there's the manual approach. The folder structure of a PowerShell Integration Module for a Windows computer is a little different from the folder structure expected by the Azure Automation. This requires a little tweaking on your part. But it's not hard, and it's done only once per resource (unless you want to upgrade it in future.) For more information on authoring PowerShell Integration Modules, see this article: [Authoring Integration Modules for Azure Automation](#)

- Install the module that you need on your workstation, as follows:
 - Install [Windows Management Framework, v5](#) (not needed for Windows 10)
 - `Install-Module -Name MODULE-NAME` <—grabs the module from the PowerShell Gallery
- Copy the module folder from `c:\Program Files\WindowsPowerShell\Modules\MODULE-NAME` to a temp folder
- Delete samples and documentation from the main folder
- Zip the main folder, naming the ZIP file exactly the same as the folder
- Put the ZIP file into a reachable HTTP location, such as blob storage in an Azure Storage Account.
- Run this PowerShell:

```
New-AzureRmAutomationModule ` 
    -ResourceGroupName MY-AUTOMATION-RG -AutomationAccountName MY-AUTOMATION-ACCOUNT ` 
    -Name MODULE-NAME -ContentLink "https://STORAGE-URI/CONTAINERNAME/MODULE-NAME.zip"
```

The included example performs these steps for cChoco and xNetworking. See the [Notes](#) for special handling for cChoco.

Step 4: Adding the node configuration to the pull server

There's nothing special about the first time you import your configuration into the pull server and compile. All subsequent import/compiles of the same configuration look exactly the same. Each time you update your package and need to push it out to production you do this step after ensuring the configuration file is correct – including the new version of your package. Here's the configuration file and PowerShell:

ISVBoxConfig.ps1:

```

Configuration ISVBoxConfig
{
    Import-DscResource -ModuleName cChoco
    Import-DscResource -ModuleName xNetworking

    Node "isvbox" {

        cChocoInstaller installChoco
        {
            InstallDir = "C:\choco"
        }

        WindowsFeature installIIS
        {
            Ensure="Present"
            Name="Web-Server"
        }

        xFirewall WebFirewallRule
        {
            Direction = "Inbound"
            Name = "Web-Server-TCP-In"
            DisplayName = "Web Server (TCP-In)"
            Description = "IIS allow incoming web site traffic."
            DisplayGroup = "IIS Incoming Traffic"
            State = "Enabled"
            Access = "Allow"
            Protocol = "TCP"
            LocalPort = "80"
            Ensure = "Present"
        }

        cChocoPackageInstaller trivialWeb
        {
            Name = "trivialweb"
            Version = "1.0.0"
            Source = "MY-NUGET-V2-SERVER-ADDRESS"
            DependsOn = "[cChocoInstaller]installChoco",
                        "[WindowsFeature]installIIS"
        }
    }
}

```

New-ConfigurationScript.ps1:

```

Import-AzureRmAutomationDscConfiguration ` 
-ResourceGroupName MY-AUTOMATION-RG -AutomationAccountName MY-AUTOMATION-ACCOUNT ` 
-SourcePath C:\temp\AzureAutomationDsc\ISVBoxConfig.ps1 ` 
-Published -Force

$jobData = Start-AzureRmAutomationDscCompilationJob ` 
-ResourceGroupName MY-AUTOMATION-RG -AutomationAccountName MY-AUTOMATION-ACCOUNT ` 
-ConfigurationName ISVBoxConfig

$compilationJobId = $jobData.Id

Get-AzureRmAutomationDscCompilationJob ` 
-ResourceGroupName MY-AUTOMATION-RG -AutomationAccountName MY-AUTOMATION-ACCOUNT ` 
-Id $compilationJobId

```

These steps result in a new node configuration named "ISVBoxConfig.isvbox" being placed on the pull server. The node configuration name is built as "configurationName.nodeName".

Step 5: Creating and maintaining package metadata

For each package that you put into the package repository, you need a nuspec that describes it. That nuspec must be compiled and stored in your NuGet server. This process is described [here](#). You can use MyGet.org as a NuGet server. They sell this service, but have a starter SKU that's free. At NuGet.org you'll find instructions on installing your own NuGet server for your private packages.

Step 6: Tying it all together

Each time a version passes QA and is approved for deployment, the package is created, nuspec and nupkg updated and deployed to the NuGet server. In addition, the configuration (Step 4 above) must be updated to agree with the new version number. It must be sent to the pull server and compiled. From that point on, it's up to the VMs that depend on that configuration to pull the update and install it. Each of these updates are simple - just a line or two of PowerShell. In the case of Visual Studio Team Services, some of them are encapsulated in build tasks that can be chained together in a build. This [article](#) provides more details. This [GitHub repo](#) details the various available build tasks.

Notes

This usage example starts with a VM from a generic Windows Server 2012 R2 image from the Azure gallery. You can start from any stored image and then tweak from there with the DSC configuration. However, changing configuration that is baked into an image is much harder than dynamically updating the configuration using DSC.

You don't have to use an ARM template and the VM extension to use this technique with your VMs. And your VMs don't have to be on Azure to be under CD management. All that's necessary is that Chocolatey be installed and the LCM configured on the VM so it knows where the pull server is.

Of course, when you update a package on a VM that's in production, you need to take that VM out of rotation while the update is installed. How you do this varies widely. For example, with a VM behind an Azure Load Balancer, you can add a Custom Probe. While updating the VM, have the probe endpoint return a 400. The tweak necessary to cause this change can be inside your configuration, as can the tweak to switch it back to returning a 200 once the update is complete.

Full source for this usage example is in [this Visual Studio project](#) on GitHub.

Related Articles

- [Azure Automation DSC Overview](#)
- [Azure Automation DSC cmdlets](#)
- [Onboarding machines for management by Azure Automation DSC](#)

Forward Azure Automation DSC reporting data to Log Analytics

6/20/2018 • 7 minutes to read • [Edit Online](#)

Automation can send DSC node status data to your Log Analytics workspace.

Compliance status is visible in the Azure portal, or with PowerShell, for nodes and for individual DSC resources in node configurations. With Log Analytics you can:

- Get compliance information for managed nodes and individual resources
- Trigger an email or alert based on compliance status
- Write advanced queries across your managed nodes
- Correlate compliance status across Automation accounts
- Visualize your node compliance history over time

Prerequisites

To start sending your Automation DSC reports to Log Analytics, you need:

- The November 2016 or later release of [Azure PowerShell](#) (v2.3.0).
- An Azure Automation account. For more information, see [Getting Started with Azure Automation](#)
- A Log Analytics workspace with an **Automation & Control** service offering. For more information, see [Get started with Log Analytics](#).
- At least one Azure Automation DSC node. For more information, see [Onboarding machines for management by Azure Automation DSC](#)

Set up integration with Log Analytics

To begin importing data from Azure Automation DSC into Log Analytics, complete the following steps:

1. Log in to your Azure account in PowerShell. See [Log in with Azure PowerShell](#)
2. Get the *ResourceId* of your automation account by running the following PowerShell command: (if you have more than one automation account, choose the *ResourceID* for the account you want to configure).

```
# Find the ResourceId for the Automation Account  
Find-AzureRmResource -ResourceType "Microsoft.Automation/automationAccounts"
```

3. Get the *ResourceId* of your Log Analytics workspace by running the following PowerShell command: (if you have more than one workspace, choose the *ResourceID* for the workspace you want to configure).

```
# Find the ResourceId for the Log Analytics workspace  
Find-AzureRmResource -ResourceType "Microsoft.OperationalInsights/workspaces"
```

4. Run the following PowerShell command, replacing `<AutomationResourceId>` and `<WorkspaceResourceId>` with the *ResourceId* values from each of the previous steps:

```
Set-AzureRmDiagnosticSetting -ResourceId <AutomationResourceId> -WorkspaceId <WorkspaceResourceId> -  
Enabled $true -Categories "DscNodeStatus"
```

If you want to stop importing data from Azure Automation DSC into Log Analytics, run the following PowerShell command.

```
Set-AzureRmDiagnosticSetting -ResourceId <AutomationResourceId> -WorkspaceId <WorkspaceResourceId> -Enabled $false -Categories "DscNodeStatus"
```

View the DSC logs

After you set up integration with Log Analytics for your Automation DSC data, a **Log search** button will appear on the **DSC Nodes** blade of your automation account. Click the **Log Search** button to view the logs for DSC node data.

The screenshot shows the 'TestAzureAuto - DSC nodes' blade in the Azure portal. On the left, there's a sidebar with options like 'Inventory (Preview)', 'Change tracking (Preview)', 'DSC nodes' (which is selected), 'DSC configurations', and 'DSC node configurations'. At the top, there's a search bar, a 'Log search' button (which is highlighted with a red box), and other navigation buttons like 'Add Azure VM', 'Add Non-Azure', 'Learn more', and 'Refresh'. Below the search area, there's a message about VM DSC extension version costs. The main content area displays a table of DSC nodes:

| NAME | STATUS | NODE CONFIGURATION | LAST SEEN | VM DSC EXTENSION VERS... |
|------------|-----------|--------------------|-------------------|--------------------------|
| ContosoVM1 | Compliant | SetupIIS.WebServer | 1/23/2018 9:33 AM | Yes |

The **Log Search** blade opens, and you see a **DscNodeStatusData** operation for each DSC node, and a **DscResourceStatusData** operation for each **DSC resource** called in the Node configuration applied to that node.

The **DscResourceStatusData** operation contains error information for any DSC resources that failed.

Click each operation in the list to see the data for that operation.

You can also view the logs by [searching in Log Analytics. See [Find data using log searches](#). Type the following query to find your DSC logs:

```
Type=AzureDiagnostics ResourceProvider="MICROSOFT.AUTOMATION" Category = "DscNodeStatus"
```

You can also narrow the query by the operation name. For example: `Type=AzureDiagnostics ResourceProvider="MICROSOFT.AUTOMATION" Category = "DscNodeStatus" OperationName = "DscNodeStatusData"

Send an email when a DSC compliance check fails

One of our top customer requests is for the ability to send an email or a text when something goes wrong with a DSC configuration.

To create an alert rule, you start by creating a log search for the DSC report records that should invoke the alert. Click the **+ New Alert Rule** button to create and configure the alert rule.

- From the Log Analytics Overview page, click **Log Search**.
- Create a log search query for your alert by typing the following search into the query field:

```
Type=AzureDiagnostics Category=DscNodeStatus NodeName_s=DSCTEST1 OperationName=DscNodeStatusData ResultType=Failed
```

If you have set up logs from more than one Automation account or subscription to your workspace, you can group your alerts by subscription and Automation account.

Automation account name can be derived from the Resource field in the search of DscNodeStatusData.

- To open the **Create rule** screen, click **+ New Alert Rule** at the top of the page. For more information on the options to configure the alert, see [Create an alert rule](#).

Find failed DSC resources across all nodes

One advantage of using Log Analytics is that you can search for failed checks across nodes. To find all instances of DSC resources that failed.

1. From the Log Analytics Overview page, click **Log Search**.
2. Create a log search query for your alert by typing the following search into the query field:

```
Type=AzureDiagnostics Category=DscNodeStatus OperationName=DscResourceStatusData ResultType=Failed
```

View historical DSC node status

Finally, you may want to visualize your DSC node status history over time.

You can use this query to search for the status of your DSC node status over time.

```
Type=AzureDiagnostics ResourceProvider="MICROSOFT.AUTOMATION" Category=DscNodeStatus NOT(ResultType="started")  
| measure Count() by ResultType interval 1hour
```

This will display a chart of the node status over time.

Log Analytics records

Diagnostics from Azure Automation creates two categories of records in Log Analytics.

DscNodeStatusData

| PROPERTY | DESCRIPTION |
|------------------------|--|
| TimeGenerated | Date and time when the compliance check ran. |
| OperationName | DscNodeStatusData |
| ResultType | Whether the node is compliant. |
| NodeName_s | The name of the managed node. |
| NodeComplianceStatus_s | Whether the node is compliant. |
| DscReportStatus | Whether the compliance check ran successfully. |

| PROPERTY | DESCRIPTION |
|---------------------|--|
| ConfigurationMode | <p>How the configuration is applied to the node. Possible values are "ApplyOnly", "ApplyandMonitor", and "ApplyandAutoCorrect".</p> <ul style="list-style-type: none"> • ApplyOnly: DSC applies the configuration and does nothing further unless a new configuration is pushed to the target node or when a new configuration is pulled from a server. After initial application of a new configuration, DSC does not check for drift from a previously configured state. DSC attempts to apply the configuration until it is successful before ApplyOnly takes effect. • ApplyAndMonitor: This is the default value. The LCM applies any new configurations. After initial application of a new configuration, if the target node drifts from the desired state, DSC reports the discrepancy in logs. DSC attempts to apply the configuration until it is successful before ApplyAndMonitor takes effect. • ApplyAndAutoCorrect: DSC applies any new configurations. After initial application of a new configuration, if the target node drifts from the desired state, DSC reports the discrepancy in logs, and then reapplies the current configuration. |
| HostName_s | The name of the managed node. |
| IPAddress | The IPv4 address of the managed node. |
| Category | DscNodeStatus |
| Resource | The name of the Azure Automation account. |
| Tenant_g | GUID that identifies the tenant for the Caller. |
| NodeId_g | GUID that identifies the managed node. |
| DscReportId_g | GUID that identifies the report. |
| LastSeenTime_t | Date and time when the report was last viewed. |
| ReportStartTime_t | Date and time when the report was started. |
| ReportEndTime_t | Date and time when the report completed. |
| NumberOfResources_d | The number of DSC resources called in the configuration applied to the node. |
| SourceSystem | How Log Analytics collected the data. Always <i>Azure</i> for Azure diagnostics. |
| ResourceId | Specifies the Azure Automation account. |
| ResultDescription | The description for this operation. |
| SubscriptionId | The Azure subscription Id (GUID) for the Automation account. |

| PROPERTY | DESCRIPTION |
|------------------|---|
| ResourceGroup | Name of the resource group for the Automation account. |
| ResourceProvider | MICROSOFT.AUTOMATION |
| ResourceType | AUTOMATIONACCOUNTS |
| CorrelationId | GUID that is the Correlation Id of the compliance report. |

DscResourceStatusData

| PROPERTY | DESCRIPTION |
|------------------------|--|
| TimeGenerated | Date and time when the compliance check ran. |
| OperationName | DscResourceStatusData |
| ResultType | Whether the resource is compliant. |
| NodeName_s | The name of the managed node. |
| Category | DscNodeStatus |
| Resource | The name of the Azure Automation account. |
| Tenant_g | GUID that identifies the tenant for the Caller. |
| NodeId_g | GUID that identifies the managed node. |
| DscReportId_g | GUID that identifies the report. |
| DscResourceId_s | The name of the DSC resource instance. |
| DscResourceName_s | The name of the DSC resource. |
| DscResourceStatus_s | Whether the DSC resource is in compliance. |
| DscModuleName_s | The name of the PowerShell module that contains the DSC resource. |
| DscModuleVersion_s | The version of the PowerShell module that contains the DSC resource. |
| DscConfigurationName_s | The name of the configuration applied to the node. |
| ErrorCode_s | The error code if the resource failed. |
| ErrorMessage_s | The error message if the resource failed. |
| DscResourceDuration_d | The time, in seconds, that the DSC resource ran. |

| PROPERTY | DESCRIPTION |
|-------------------|---|
| SourceSystem | How Log Analytics collected the data. Always Azure for Azure diagnostics. |
| ResourceId | Specifies the Azure Automation account. |
| ResultDescription | The description for this operation. |
| SubscriptionId | The Azure subscription Id (GUID) for the Automation account. |
| ResourceGroup | Name of the resource group for the Automation account. |
| ResourceProvider | MICROSOFT.AUTOMATION |
| ResourceType | AUTOMATIONACCOUNTS |
| CorrelationId | GUID that is the Correlation Id of the compliance report. |

Summary

By sending your Automation DSC data to Log Analytics, you can get better insight into the status of your Automation DSC nodes by:

- Setting up alerts to notify you when there is an issue
- Using custom views and search queries to visualize your runbook results, runbook job status, and other related key indicators or metrics.

Log Analytics provides greater operational visibility to your Automation DSC data and can help address incidents more quickly.

Next steps

- To learn more about how to construct different search queries and review the Automation DSC logs with Log Analytics, see [Log searches in Log Analytics](#)
- To learn more about using Azure Automation DSC, see [Getting started with Azure Automation DSC](#)
- To learn more about Log Analytics and data collection sources, see [Collecting Azure storage data in Log Analytics overview](#)

Certificate assets in Azure Automation

5/21/2018 • 3 minutes to read • [Edit Online](#)

Certificates can be stored securely in Azure Automation so they can be accessed by runbooks or DSC configurations using the **Get-AzureRmAutomationCertificate** activity for Azure Resource Manager resources. This capability allows you to create runbooks and DSC configurations that use certificates for authentication or adds them to Azure or third-party resources.

NOTE

Secure assets in Azure Automation include credentials, certificates, connections, and encrypted variables. These assets are encrypted and stored in Azure Automation using a unique key that is generated for each automation account. This key is stored in Key Vault. Before storing a secure asset, the key is loaded from Key Vault and then used to encrypt the asset.

AzureRM PowerShell cmdlets

For AzureRM, the cmdlets in the following table are used to create and manage automation credential assets with Windows PowerShell. They ship as part of the [AzureRM.Automation module](#) which is available for use in Automation runbooks and DSC configurations.

| CMDLETS | DESCRIPTION |
|---|---|
| Get-AzureRmAutomationCertificate | Retrieves information about a certificate to use in a runbook or DSC configuration. You can only retrieve the certificate itself from Get-AutomationCertificate activity. |
| New-AzureRmAutomationCertificate | Creates a new certificate into Azure Automation. |
| Remove-AzureRmAutomationCertificate | Removes a certificate from Azure Automation. |
| Set-AzureRmAutomationCertificate | Sets the properties for an existing certificate including uploading the certificate file and setting the password for a .pfx. |
| Add-AzureCertificate | Uploads a service certificate for the specified cloud service. |

Activities

The activities in the following table are used to access certificates in a runbook and DSC configurations.

| ACTIVITIES | DESCRIPTION |
|---|---|
| Get-AutomationCertificate | Gets a certificate to use in a runbook or DSC configuration. Returns a System.Security.Cryptography.X509Certificates.X509Certificate2 object. |

NOTE

You should avoid using variables in the `-Name` parameter of **Get-AutomationCertificate** in a runbook or DSC configuration as it complicates discovering dependencies between runbooks or DSC configuration, and Automation variables at design time.

Python2 functions

The function in the following table is used to access certificates in a Python2 runbook.

| FUNCTION | DESCRIPTION |
|--|--|
| <code>automationassets.get_automation_certificate</code> | Retrieves information about a certificate asset. |

NOTE

You must import the **automationassets** module in the beginning of your Python runbook in order to access the asset functions.

Creating a new certificate

When you create a new certificate, you upload a .cer or .pfx file to Azure Automation. If you mark the certificate as exportable, then you can transfer it out of the Azure Automation certificate store. If it is not exportable, then it can only be used for signing within the runbook or DSC configuration. Azure Automation requires the certificate to have the provider: **Microsoft Enhanced RSA and AES Cryptographic Provider**.

To create a new certificate with the Azure portal

1. From your Automation account, click the **Assets** tile to open the **Assets** blade.
2. Click the **Certificates** tile to open the **Certificates** blade.
3. Click **Add a certificate** at the top of the blade.
4. Type a name for the certificate in the **Name** box.
5. To browse for a .cer or .pfx file, click **Select a file** under **Upload a certificate file**. If you select a .pfx file, specify a password and whether it is allowed to be exported.
6. Click **Create** to save the new certificate asset.

To create a new certificate with Windows PowerShell

The following example demonstrates how to create a new Automation certificate and mark it exportable. This imports an existing .pfx file.

```
$certName = 'MyCertificate'  
$certPath = '.\MyCert.pfx'  
$certPwd = ConvertTo-SecureString -String 'P@$$w0rd' -AsPlainText -Force  
$ResourceGroup = "ResourceGroup01"  
  
New-AzureRmAutomationCertificate -AutomationAccountName "MyAutomationAccount" -Name $certName -Path $certPath  
-Password $certPwd -Exportable -ResourceGroupName $ResourceGroup
```

Using a certificate

To use a certificate, use the **Get-AutomationCertificate** activity. You cannot use the **Get-AzureRmAutomationCertificate** cmdlet since it returns information about the certificate asset but not the certificate itself.

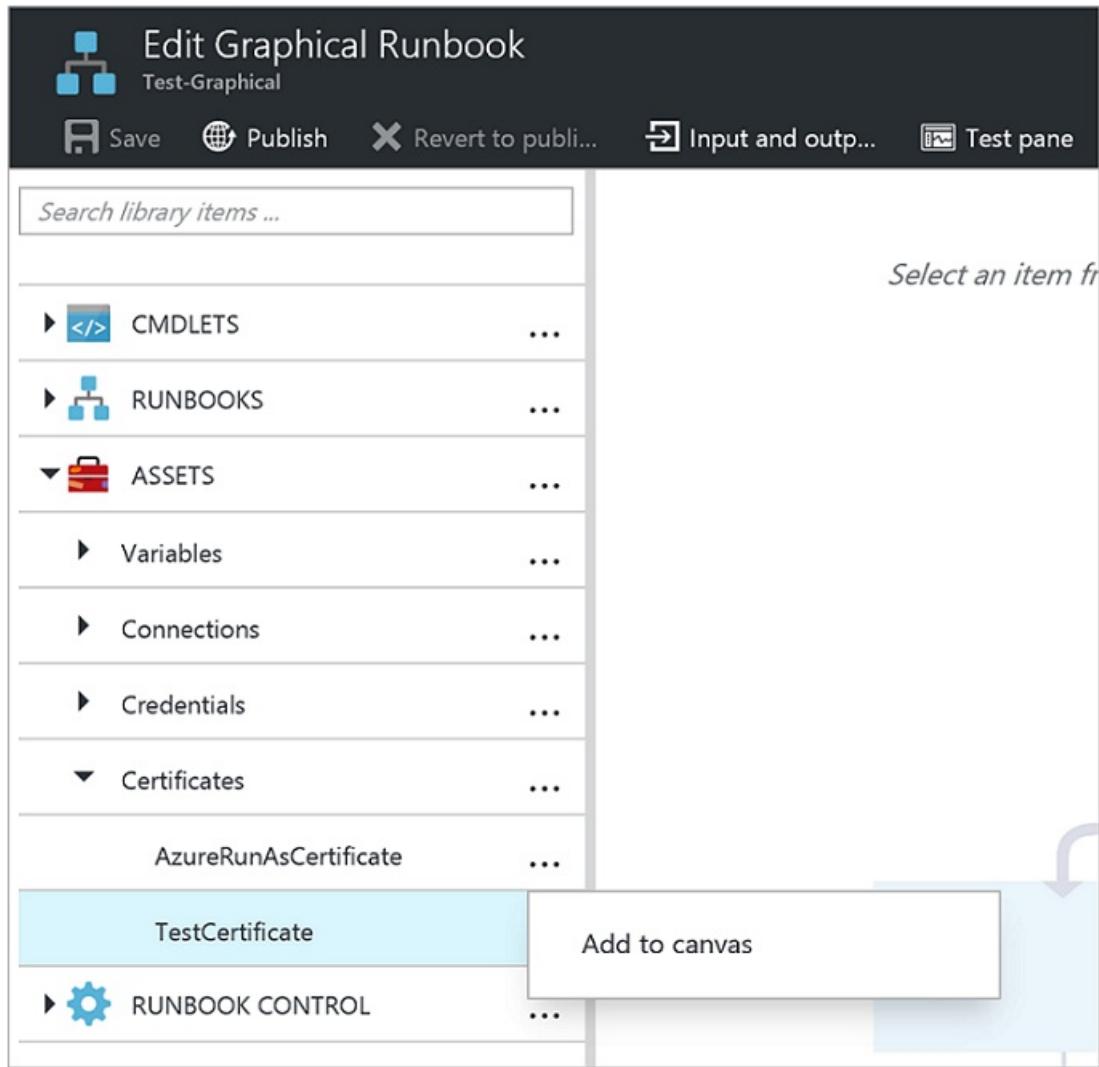
Textual runbook sample

The following sample code shows how to add a certificate to a cloud service in a runbook. In this sample, the password is retrieved from an encrypted automation variable.

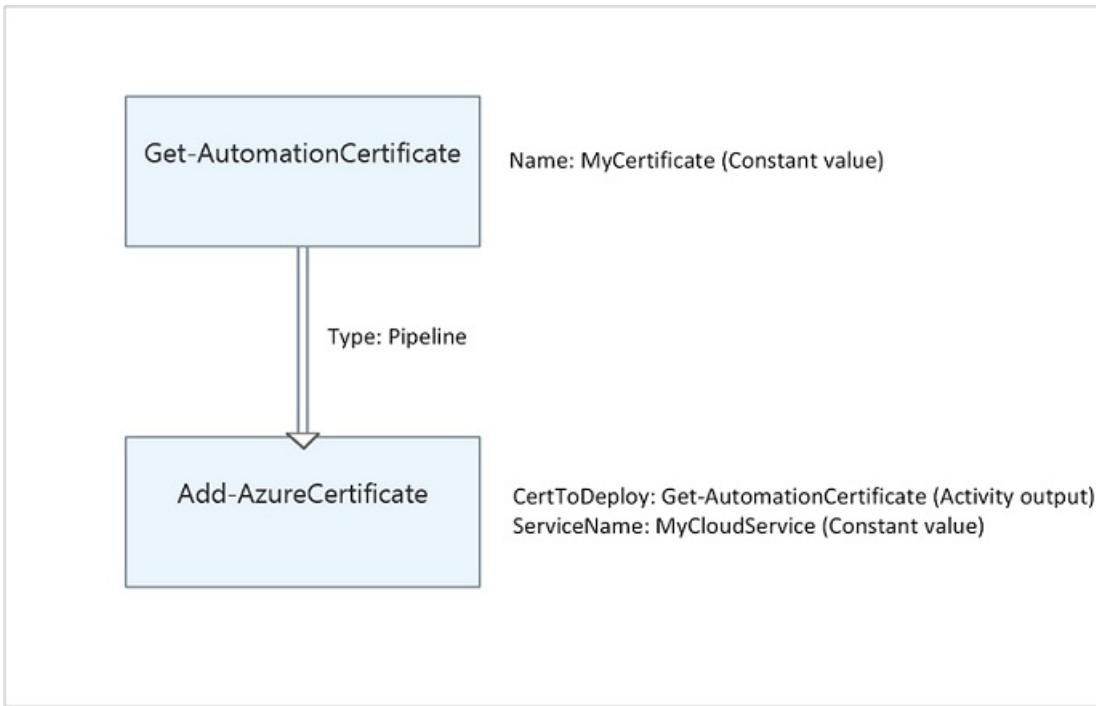
```
$serviceName = 'MyCloudService'  
$cert = Get-AutomationCertificate -Name 'MyCertificate'  
$certPwd = Get-AzureRmAutomationVariable -ResourceGroupName "ResouceGroup01" `  
-AutomationAccountName "MyAutomationAccount" -Name 'MyCertPassword'  
Add-AzureCertificate -ServiceName $serviceName -CertToDeploy $cert
```

Graphical runbook sample

You add a **Get-AutomationCertificate** to a graphical runbook by right-clicking on the certificate in the Library pane of the graphical editor and selecting **Add to canvas**.



The following image shows an example of using a certificate in a graphical runbook. This is the same as the preceding example for adding a certificate to a cloud service from a textual runbook.



Python2 sample

The following sample shows how to access certificates in Python2 runbooks.

```
# get a reference to the Azure Automation certificate
cert = automationassets.get_automation_certificate("AzureRunAsCertificate")

# returns the binary cert content
print cert
```

Next steps

- To learn more about working with links to control the logical flow of activities your runbook is designed to perform, see [Links in graphical authoring](#).

Connection assets in Azure Automation

7/3/2018 • 6 minutes to read • [Edit Online](#)

An Automation connection asset contains the information required to connect to an external service or application from a runbook or DSC configuration. This may include information required for authentication such as a username and password in addition to connection information such as a URL or a port. The value of a connection is keeping all of the properties for connecting to a particular application in one asset as opposed to creating multiple variables. The user can edit the values for a connection in one place, and you can pass the name of a connection to a runbook or DSC configuration in a single parameter. The properties for a connection can be accessed in the runbook or DSC configuration with the **Get-AutomationConnection** activity.

When you create a connection, you must specify a *connection type*. The connection type is a template that defines a set of properties. The connection defines values for each property defined in its connection type. Connection types are added to Azure Automation in integration modules or created with the [Azure Automation API](#) if the integration module includes a connection type and is imported into your Automation account. Otherwise, you will need to create a metadata file to specify an Automation connection type. For further information regarding this, see [Integration Modules](#).

NOTE

Secure assets in Azure Automation include credentials, certificates, connections, and encrypted variables. These assets are encrypted and stored in Azure Automation using a unique key that is generated for each automation account. This key is stored in Key Vault. Before storing a secure asset, the key is loaded from Key Vault and then used to encrypt the asset.

Windows PowerShell Cmdlets

The cmdlets in the following table are used to create and manage Automation connections with Windows PowerShell. They ship as part of the [Azure PowerShell module](#) which is available for use in Automation runbooks and DSC configurations.

| CMDLET | DESCRIPTION |
|---|---|
| Get-AzureRmAutomationConnection | Retrieves a connection. Includes a hash table with the values of the connection's fields. |
| New-AzureRmAutomationConnection | Creates a new connection. |
| Remove-AzureRmAutomationConnection | Remove an existing connection. |
| Set-AzureRmAutomationConnectionFieldValue | Sets the value of a particular field for an existing connection. |

Activities

The activities in the following table are used to access connections in a runbook or DSC configuration.

| ACTIVITIES | DESCRIPTION |
|--|---|
| Get-AutomationConnection | Gets a connection to use. Returns a hash table with the properties of the connection. |

NOTE

You should avoid using variables with the `-Name` parameter of **Get-AutomationConnection** since this can complicate discovering dependencies between runbooks or DSC configurations, and connection assets at design time.

Python2 functions

The function in the following table is used to access connections in a Python2 runbook.

| FUNCTION | DESCRIPTION |
|--|---|
| automationassets.get_automation_connection | Retrieves a connection. Returns a dictionary with the properties of the connection. |

NOTE

You must import the "automationassets" module at the top of your Python runbook in order to access the asset functions.

Creating a New Connection

To create a new connection with the Azure portal

1. From your automation account, click the **Assets** part to open the **Assets** blade.
2. Click the **Connections** part to open the **Connections** blade.
3. Click **Add a connection** at the top of the blade.
4. In the **Type** dropdown, select the type of connection you want to create. The form will present the properties for that particular type.
5. Complete the form and click **Create** to save the new connection.

To create a new connection with Windows PowerShell

Create a new connection with Windows PowerShell using the [New-AzureRmAutomationConnection](#) cmdlet. This cmdlet has a parameter named **ConnectionFieldValues** that expects a [hash table](#) defining values for each of the properties defined by the connection type.

If you are familiar with the Automation [Run As account](#) to authenticate runbooks using the service principal, the PowerShell script, provided as an alternative to creating the Run As account from the portal, creates a new connection asset using the following sample commands.

```
$ConnectionAssetName = "AzureRunAsConnection"
$ConnectionFieldValues = @{
    "ApplicationId" = $Application.ApplicationId; "TenantId" = $TenantID.TenantId;
    "CertificateThumbprint" = $Cert.Thumbprint; "SubscriptionId" = $SubscriptionId}
New-AzureRmAutomationConnection -ResourceGroupName $ResourceGroup -AutomationAccountName
$AutomationAccountName -Name $ConnectionAssetName -ConnectionTypeName AzureServicePrincipal -
ConnectionFieldValues $ConnectionFieldValues
```

You are able to use the script to create the connection asset because when you create your Automation account, it automatically includes several global modules by default along with the connection type **AzureServicePrincipal** to create the **AzureRunAsConnection** connection asset. This is important to keep in mind, because if you attempt to create a new connection asset to connect to a service or application with a different authentication method, it will fail because the connection type is not already defined in your Automation account. For further information on how to create your own connection type for your custom or module from the [PowerShell Gallery](#), see [Integration Modules](#).

Using a connection in a runbook or DSC configuration

You retrieve a connection in a runbook or DSC configuration with the **Get-AutomationConnection** cmdlet. You cannot use the **Get-AzureRmAutomationConnection** activity. This activity retrieves the values of the different fields in the connection and returns them as a [hash table](#) which can then be used with the appropriate commands in the runbook or DSC configuration.

Textual runbook sample

The following sample commands show how to use the Run As account mentioned earlier, to authenticate with Azure Resource Manager resources in your runbook. It uses the connection asset representing the Run As account, which references the certificate-based service principal, not credentials.

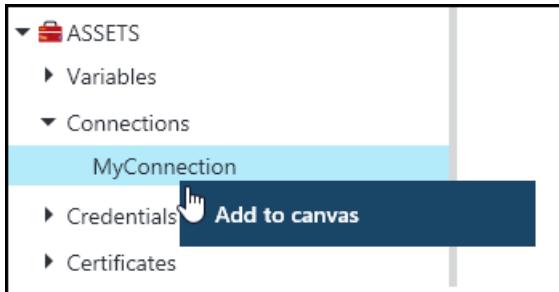
```
$Conn = Get-AutomationConnection -Name AzureRunAsConnection  
Connect-AzureRmAccount -ServicePrincipal -Tenant $Conn.TenantID -ApplicationId $Conn.ApplicationID -  
CertificateThumbprint $Conn.CertificateThumbprint
```

IMPORTANT

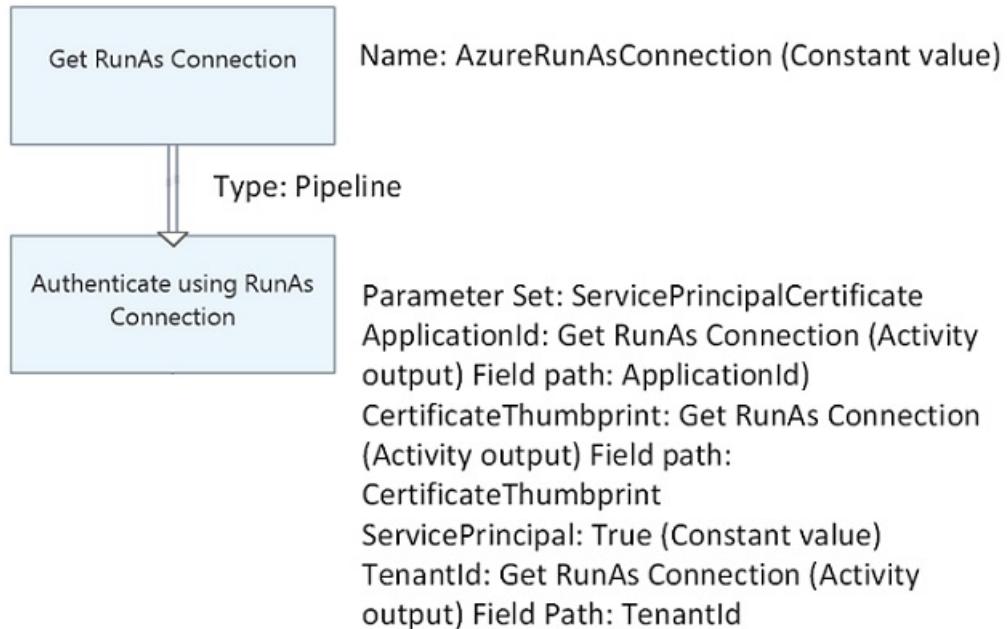
Add-AzureRmAccount is now an alias for **Connect-AzureRMAccount**. When searching your library items, if you do not see **Connect-AzureRMAccount**, you can use **Add-AzureRmAccount**, or you can update your modules in your Automation Account.

Graphical runbook samples

You add a **Get-AutomationConnection** activity to a graphical runbook by right-clicking on the connection in the Library pane of the graphical editor and selecting **Add to canvas**.



The following image shows an example of using a connection in a graphical runbook. This is the same example shown above for authenticating using the Run As account with a textual runbook. This example uses the **Constant value** data set for the **Get RunAs Connection** activity that uses a connection object for authentication. A [pipeline link](#) is used here since the ServicePrincipalCertificate parameter set is expecting a single object.



Python2 runbook sample

The following sample shows how to authenticate using the Run As connection in a Python2 runbook.

```

""" Tutorial to show how to authenticate against Azure resource manager resources """
import azure.mgmt.resource
import automationassets

def get_automation_runas_credential(runas_connection):
    """ Returns credentials to authenticate against Azure resoruce manager """
    from OpenSSL import crypto
    from msrestazure import azure_active_directory
    import adal

    # Get the Azure Automation Run As service principal certificate
    cert = automationassets.get_automation_certificate("AzureRunAsCertificate")
    pks12_cert = crypto.load_pkcs12(cert)
    pem_pkey = crypto.dump_privatekey(crypto.FILETYPE_PEM, pks12_cert.get_privatekey())

    # Get Run As connection information for the Azure Automation service principal
    application_id = runas_connection["ApplicationId"]
    thumbprint = runas_connection["CertificateThumbprint"]
    tenant_id = runas_connection["TenantId"]

    # Authenticate with service principal certificate
    resource = "https://management.core.windows.net/"
    authority_url = ("https://login.microsoftonline.com/" + tenant_id)
    context = adal.AuthenticationContext(authority_url)
    return azure_active_directory.AdalAuthentication(
        lambda: context.acquire_token_with_client_certificate(
            resource,
            application_id,
            pem_pkey,
            thumbprint)
    )

# Authenticate to Azure using the Azure Automation Run As service principal
runas_connection = automationassets.get_automation_connection("AzureRunAsConnection")
azure_credential = get_automation_runas_credential(runas_connection)

```

Next steps

- Review [Links in graphical authoring](#) to understand how to direct and control the flow of logic in your runbooks.
- To learn more about Azure Automation's use of PowerShell modules and best practices for creating your own PowerShell modules to work as Integration Modules within Azure Automation, see [Integration Modules](#).

Credential assets in Azure Automation

5/21/2018 • 4 minutes to read • [Edit Online](#)

An Automation credential asset holds an object which contains security credentials such as a username and password. Runbooks and DSC configurations may use cmdlets that accept a PSCredential object for authentication, or they may extract the username and password of the PSCredential object to provide to some application or service requiring authentication. The properties for a credential are stored securely in Azure Automation and can be accessed in the runbook or DSC configuration with the [Get-AutomationPSCredential](#) activity.

NOTE

If you're interested in viewing or deleting personal data, please see the [Azure Data Subject Requests for the GDPR](#) article. If you're looking for general info about GDPR, see the [GDPR section of the Service Trust portal](#).

NOTE

Secure assets in Azure Automation include credentials, certificates, connections, and encrypted variables. These assets are encrypted and stored in Azure Automation using a unique key that is generated for each automation account. This key is stored in Key Vault. Before storing a secure asset, the key is loaded from Key Vault and then used to encrypt the asset.

Azure Classic PowerShell cmdlets

The cmdlets in the following table are used to create and manage automation credential assets with Windows PowerShell. They ship as part of the [Azure PowerShell module](#) which is available for use in Automation runbooks and DSC configurations.

| CMDLETS | DESCRIPTION |
|---|---|
| Get-AzureAutomationCredential | Retrieves information about a credential asset. You can only retrieve the credential itself from Get-AutomationPSCredential activity. |
| New-AzureAutomationCredential | Creates a new Automation credential. |
| Remove- AzureAutomationCredential | Removes an Automation credential. |
| Set- AzureAutomationCredential | Sets the properties for an existing Automation credential. |

AzureRM PowerShell cmdlets

For AzureRM, the cmdlets in the following table are used to create and manage automation credential assets with Windows PowerShell. They ship as part of the [AzureRM.Automation module](#) which is available for use in Automation runbooks and DSC configurations.

| CMDLETS | DESCRIPTION |
|---|---|
| Get-AzureRmAutomationCredential | Retrieves information about a credential asset. |

| CMDLETS | DESCRIPTION |
|------------------------------------|--|
| New-AzureRmAutomationCredential | Creates a new Automation credential. |
| Remove-AzureRmAutomationCredential | Removes an Automation credential. |
| Set-AzureRmAutomationCredential | Sets the properties for an existing Automation credential. |

Activities

The activities in the following table are used to access credentials in a runbook and DSC configurations.

| ACTIVITIES | DESCRIPTION |
|----------------------------|---|
| Get-AutomationPSCredential | Gets a credential to use in a runbook or DSC configuration. Returns a System.Management.Automation.PSCredential object. |

NOTE

You should avoid using variables in the –Name parameter of Get-AutomationPSCredential since this can complicate discovering dependencies between runbooks or DSC configurations, and credential assets at design time.

Python2 functions

The function in the following table is used to access credentials in a Python2 runbook.

| FUNCTION | DESCRIPTION |
|--|---|
| automationassets.get_automation_credential | Retrieves information about a credential asset. |

NOTE

You must import the "automationassets" module at the top of your Python runbook in order to access the asset functions.

Creating a new credential asset

To create a new credential asset with the Azure portal

1. From your automation account, click the **Assets** part to open the **Assets** blade.
2. Click the **Credentials** part to open the **Credentials** blade.
3. Click **Add a credential** at the top of the blade.
4. Complete the form and click **Create** to save the new credential.

To create a new credential asset with Windows PowerShell

The following sample commands show how to create a new automation credential. A PSCredential object is first created with the name and password and then used to create the credential asset. Alternatively, you could use the **Get-Credential** cmdlet to be prompted to type in a name and password.

```
$user = "MyDomain\MyUser"
$pw = ConvertTo-SecureString "PassWord!" -AsPlainText -Force
$cred = New-Object -TypeName System.Management.Automation.PSCredential -ArgumentList $user, $pw
New-AzureAutomationCredential -AutomationAccountName "MyAutomationAccount" -Name "MyCredential" -Value $cred
```

Using a PowerShell credential

You retrieve a credential asset in a runbook or DSC configuration with the **Get-AutomationPSCredential** activity. This returns a **PSCredential object** that you can use with an activity or cmdlet that requires a **PSCredential** parameter. You can also retrieve the properties of the credential object to use individually. The object has a property for the username and the secure password, or you can use the **GetNetworkCredential** method to return a **NetworkCredential** object that will provide an unsecured version of the password.

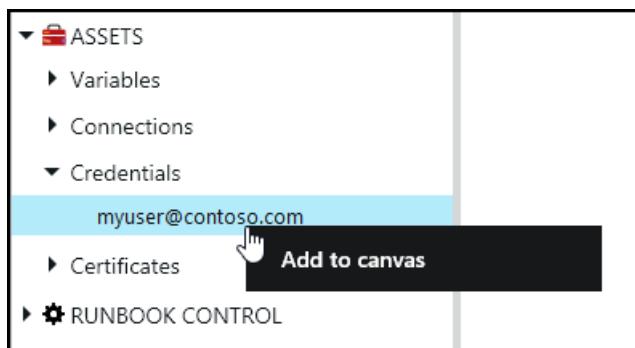
Textual runbook sample

The following sample commands show how to use a PowerShell credential in a runbook. In this example, the credential is retrieved and its username and password assigned to variables.

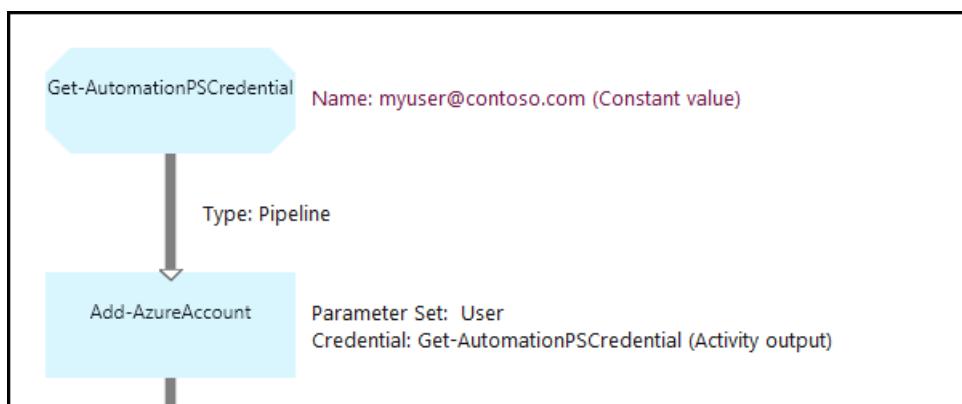
```
$myCredential = Get-AutomationPSCredential -Name 'MyCredential'
$userName = $myCredential.UserName
$securePassword = $myCredential.Password
$password = $myCredential.GetNetworkCredential().Password
```

Graphical runbook sample

You add a **Get-AutomationPSCredential** activity to a graphical runbook by right-clicking on the credential in the Library pane of the graphical editor and selecting **Add to canvas**.



The following image shows an example of using a credential in a graphical runbook. In this case, it is being used to provide authentication for a runbook to Azure resources as described in [Authenticate Runbooks with Azure AD User account](#). The first activity retrieves the credential that has access to the Azure subscription. The **Add-AzureAccount** activity then uses this credential to provide authentication for any activities that come after it. A [pipeline link](#) is here since **Get-AutomationPSCredential** is expecting a single object.



Using a PowerShell credential in DSC

While DSC configurations in Azure Automation can reference credential assets using **Get-AutomationPSCredential**, credential assets can also be passed in via parameters, if desired. For more information, see [Compiling configurations in Azure Automation DSC](#).

Using credentials in Python2

The following sample shows an example of accessing credentials in Python2 runbooks.

```
import automationassets
from automationassets import AutomationAssetNotFound

# get a credential
cred = automationassets.get_automation_credential("credtest")
print cred["username"]
print cred["password"]
```

Next Steps

- To learn more about links in graphical authoring, see [Links in graphical authoring](#)
- To understand the different authentication methods with Automation, see [Azure Automation Security](#)
- To get started with Graphical runbooks, see [My first graphical runbook](#)
- To get started with PowerShell workflow runbooks, see [My first PowerShell workflow runbook](#)
- To get started with Python2 runbooks, see [My first Python2 runbook](#)

Azure Automation Integration Modules

5/21/2018 • 10 minutes to read • [Edit Online](#)

PowerShell is the fundamental technology behind Azure Automation. Since Azure Automation is built on PowerShell, PowerShell modules are key to the extensibility of Azure Automation. In this article, we guide you through the specifics of Azure Automation's use of PowerShell modules, referred to as "Integration Modules", and best practices for creating your own PowerShell modules to make sure they work as Integration Modules within Azure Automation.

What is a PowerShell Module?

A PowerShell module is a group of PowerShell cmdlets like **Get-Date** or **Copy-Item**, that can be used from the PowerShell console, scripts, workflows, runbooks, and PowerShell DSC resources like WindowsFeature or File, that can be used from PowerShell DSC configurations. All of the functionality of PowerShell is exposed through cmdlets and DSC resources, and every cmdlet/DSC resource is backed by a PowerShell module, many of which ship with PowerShell itself. For example, the **Get-Date** cmdlet is part of the Microsoft.PowerShell.Utility PowerShell module, and **Copy-Item** cmdlet is part of the Microsoft.PowerShell.Management PowerShell module and the Package DSC resource is part of the PSDesiredStateConfiguration PowerShell module. Both of these modules ship with PowerShell. But many PowerShell modules do not ship as part of PowerShell, and are instead distributed with first or third-party products like System Center 2012 Configuration Manager or by the vast PowerShell community on places like PowerShell Gallery. The modules are useful because they make complex tasks simpler through encapsulated functionality. You can learn more about [PowerShell modules on MSDN](#).

What is an Azure Automation Integration Module?

An Integration Module isn't different from a PowerShell module. It's simply a PowerShell module that optionally contains one additional file - a metadata file specifying an Azure Automation connection type to be used with the module's cmdlets in runbooks. Optional file or not, these PowerShell modules can be imported into Azure Automation to make their cmdlets available for use within runbooks and their DSC resources available for use within DSC configurations. Behind the scenes, Azure Automation stores these modules, and at runbook job and DSC compilation job execution time, loads them into the Azure Automation sandboxes where runbooks are executed and DSC configurations are compiled. Any DSC resources in modules are also automatically placed on the Automation DSC pull server, so that they can be pulled by machines attempting to apply DSC configurations.

We ship a number of Azure PowerShell modules out of the box in Azure Automation for you to use so you can get started automating Azure management right away, but you can import PowerShell modules for whatever system, service, or tool you want to integrate with.

NOTE

Certain modules are shipped as "global modules" in the Automation service. These global modules are available to you when you create an automation account, and we update them sometimes, which automatically pushes them out to your automation account. If you don't want them to be auto-updated, you can always import the same module yourself, and that takes precedence over the global module version of that module that we ship in the service.

The format in which you import an Integration Module package is a compressed file with the same name as the module and a .zip extension. It contains the Windows PowerShell module and any supporting files, including a manifest file (.psd1) if the module has one.

If the module should contain an Azure Automation connection type, it must also contain a file with the name

<ModuleName>-Automation.json that specifies the connection type properties. This is a json file placed within the module folder of your compressed .zip file, and contains the fields of a "connection" that is required to connect to the system or service the module represents. This ends up creating a connection type in Azure Automation. Using this file you can set the field names, types, and whether the fields should be encrypted and / or optional, for the connection type of the module. The following is a template in the json file format:

```
{  
    "ConnectionFields": [  
        {  
            "IsEncrypted": false,  
            "IsOptional": false,  
            "Name": "ComputerName",  
            "TypeName": "System.String"  
        },  
        {  
            "IsEncrypted": false,  
            "IsOptional": true,  
            "Name": "Username",  
            "TypeName": "System.String"  
        },  
        {  
            "IsEncrypted": true,  
            "IsOptional": false,  
            "Name": "Password",  
            "TypeName": "System.String"  
        }],  
    "ConnectionTypeName": "DataProtectionManager",  
    "IntegrationModuleName": "DataProtectionManager"  
}
```

If you have deployed Service Management Automation and created Integration Modules packages for your automation runbooks, this should look familiar to you.

Authoring Best Practices

Even though Integration Modules are essentially PowerShell modules, there's still a number of things we recommend you consider while authoring a PowerShell module, to make it most usable in Azure Automation. Some of these are Azure Automation specific, and some of them are useful just to make your modules work well in PowerShell Workflow, regardless of whether or not you're using Automation.

1. Include a synopsis, description, and help URI for every cmdlet in the module. In PowerShell, you can define certain help information for cmdlets to allow the user to receive help on using them with the **Get-Help** cmdlet. For example, here's how you can define a synopsis and help URI for a PowerShell module written in a .psm1 file.

```

<#
.SYNOPSIS
Gets all outgoing phone numbers for this Twilio account
#>
function Get-TwilioPhoneNumbers {
[CmdletBinding(DefaultParameterSetName='SpecifyConnectionFields',
HelpUri='http://www.twilio.com/docs/api/rest/outgoing-caller-ids')]
param(
[Parameter(ParameterSetName='SpecifyConnectionFields', Mandatory=$true)]
[ValidateNotNullOrEmpty()]
[string]
$AccountSid,
[Parameter(ParameterSetName='SpecifyConnectionFields', Mandatory=$true)]
[ValidateNotNullOrEmpty()]
[string]
$AuthToken,
[Parameter(ParameterSetName='UseConnectionObject', Mandatory=$true)]
[ValidateNotNullOrEmpty()]
[Hashtable]
$Connection
)
$cred = CreateTwilioCredential -Connection $Connection -AccountSid $AccountSid -AuthToken $AuthToken
$uri = "$TWILIO_BASE_URL/Accounts/" + $cred.UserName + "/IncomingPhoneNumbers"
$response = Invoke-RestMethod -Method Get -Uri $uri -Credential $cred
$response.TwilioResponse.IncomingPhoneNumbers.IncomingPhoneNumber
}

```

Providing this info will not only show this help using the **Get-Help** cmdlet in the PowerShell console, it will also expose this help functionality within Azure Automation. For example, when inserting activities during runbook authoring. Clicking “View detailed help” will open the help URI in another tab of the web browser you’re using to access Azure Automation.

| Activities | |
|----------------------|---|
| 13 |  |
| NAME | DESCRIPTION |
| Connect-WSMan | Connect-WSMan [[-ComputerName] <string>] [-Appli...] |
| Disable-WSManCredSSP | Disable-WSManCredSSP [-Role] <string> [<Common...] |
| Disconnect-WSMan | Disconnect-WSMan [[-ComputerName] <string>] [<C...] |
| Enable-WSManCredSSP | Enable-WSManCredSSP [-Role] <string> [[-DelegateC...] |
| Get-WSManCredSSP | Get-WSManCredSSP [<CommonParameters>] |
| Get-WSManInstance | Get-WSManInstance [-ResourceURI] <uri> [-Aplicati... |
| Invoke-WSManAction | Invoke-WSManAction [-ResourceURI] <uri> [-Action]... |

2. If the module runs against a remote system,

- a. It should contain an Integration Module metadata file that defines the information needed to connect to that remote system, meaning the connection type.
- b. Each cmdlet in the module should be able to take in a connection object (an instance of that connection type) as a parameter.

Cmdlets in the module become easier to use in Azure Automation if you allow passing an object with the

fields of the connection type as a parameter to the cmdlet. This way users don't have to map parameters of the connection asset to the cmdlet's corresponding parameters each time they call a cmdlet. Based on the runbook example above, it uses a Twilio connection asset called CorpTwilio to access Twilio and return all the phone numbers in the account. Notice how it is mapping the fields of the connection to the parameters of the cmdlet?

```
workflow Get-CorpTwilioPhones
{
    $CorpTwilio = Get-AutomationConnection -Name 'CorpTwilio'

    Get-TwilioPhoneNumbers
        -AccountSid $CorpTwilio.AccountSid
        -AuthToken $CorpTwilio.AuthToken
}
```

An easier and better way to approach this is directly passing the connection object to the cmdlet -

```
workflow Get-CorpTwilioPhones
{
    $CorpTwilio = Get-AutomationConnection -Name 'CorpTwilio'

    Get-TwilioPhoneNumbers -Connection $CorpTwilio
}
```

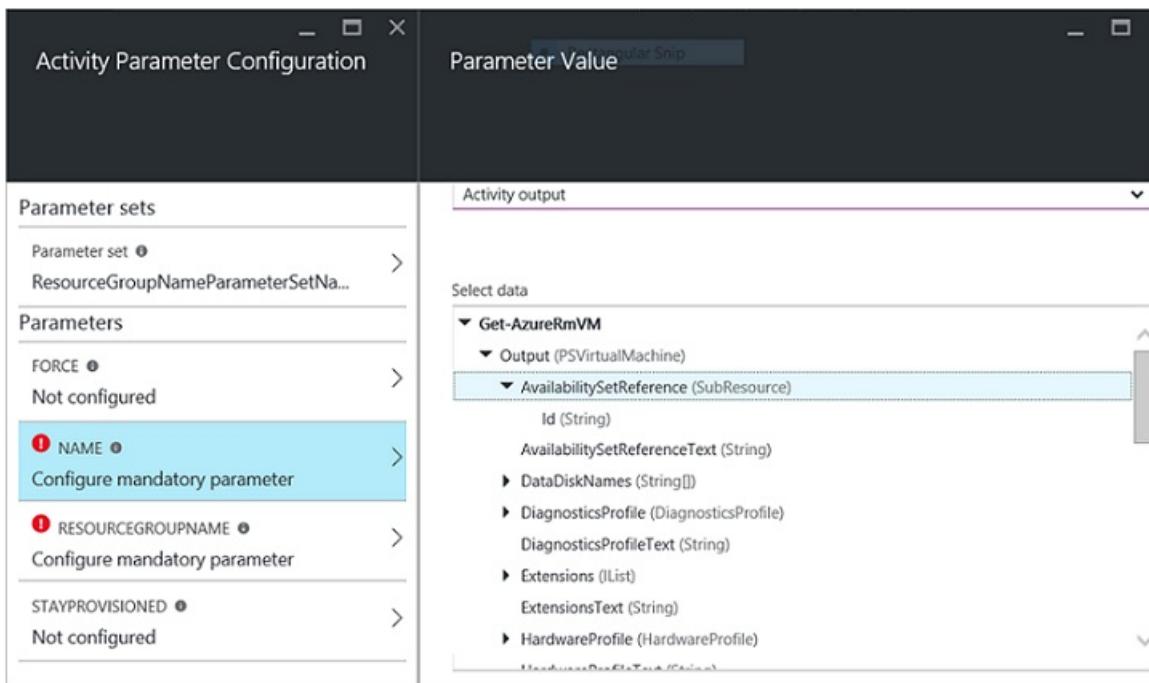
You can enable behavior like this for your cmdlets by allowing them to accept a connection object directly as a parameter, instead of just connection fields for parameters. Usually you want a parameter set for each, so that a user not using Azure Automation can call your cmdlets without constructing a hashtable to act as the connection object. Parameter set **SpecifyConnectionFields** below is used to pass the connection field properties one by one. **UseConnectionObject** lets you pass the connection straight through. As you can see, the Send-TwilioSMS cmdlet in the [Twilio PowerShell module](#) allows passing either way:

```
function Send-TwilioSMS {
    [CmdletBinding(DefaultParameterSetName='SpecifyConnectionFields', ` 
    HelpUri='http://www.twilio.com/docs/api/rest/sending-sms')]
    param(
        [Parameter(ParameterSetName='SpecifyConnectionFields', Mandatory=$true)]
        [ValidateNotNullOrEmpty()]
        [string]
        $AccountSid,

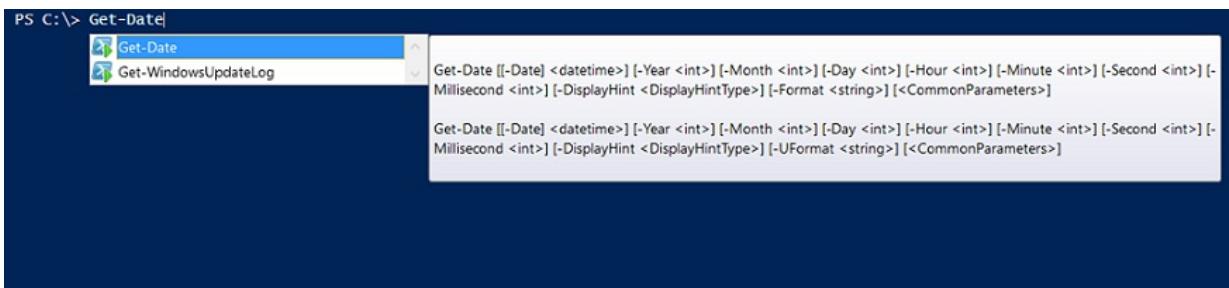
        [Parameter(ParameterSetName='SpecifyConnectionFields', Mandatory=$true)]
        [ValidateNotNullOrEmpty()]
        [string]
        $AuthToken,

        [Parameter(ParameterSetName='UseConnectionObject', Mandatory=$true)]
        [ValidateNotNullOrEmpty()]
        [Hashtable]
        $Connection
    )
}
```

3. Define output type for all cmdlets in the module. Defining an output type for a cmdlet allows design-time IntelliSense to help you determine the output properties of the cmdlet, for use during authoring. It is especially helpful during Automation runbook graphical authoring, where design time knowledge is key to an easy user experience with your module.



This is similar to the "type ahead" functionality of a cmdlet's output in PowerShell ISE without having to run it.



- Cmdlets in the module should not take complex object types for parameters. PowerShell Workflow is different from PowerShell in that it stores complex types in deserialized form. Primitive types stay as primitives, but complex types are converted to their deserialized versions, which are essentially property bags. For example, if you used the **Get-Process** cmdlet in a runbook (or a PowerShell Workflow for that matter), it would return an object of type [Deserialized.System.Diagnostic.Process], not the expected [System.Diagnostic.Process] type. This type has all the same properties as the non-deserialized type, but none of the methods. And if you try to pass this value as a parameter to a cmdlet, where the cmdlet expects a [System.Diagnostic.Process] value for this parameter, you receive the following error: *Cannot process argument transformation on parameter 'process'. Error: "Cannot convert the "System.Diagnostics.Process (CcmExec)" value of type "Deserialized.System.Diagnostics.Process" to type "System.Diagnostics.Process".* This is because there is a type mismatch between the expected [System.Diagnostic.Process] type and the given [Deserialized.System.Diagnostic.Process] type. The way around this issue is to ensure the cmdlets of your module do not take complex types for parameters. Here is the wrong way to do it.

```
function Get-ProcessDescription {
    param (
        [System.Diagnostic.Process] $process
    )
    $process.Description
}
```

And here is the right way, taking in a primitive that can be used internally by the cmdlet to grab the complex object and use it. Since cmdlets execute in the context of PowerShell, not PowerShell Workflow, inside the cmdlet \$process becomes the correct [System.Diagnostic.Process] type.

```

function Get-ProcessDescription {
    param (
        [String] $processName
    )
    $process = Get-Process -Name $processName

    $process.Description
}

```

Connection assets in runbooks are hashtables, which are a complex type, and yet these hashtables seem to be able to be passed into cmdlets for their `-Connection` parameter perfectly, with no cast exception.

Technically, some PowerShell types are able to cast properly from their serialized form to their deserialized form, and hence can be passed into cmdlets for parameters accepting the non-deserialized type. Hashtable is one of these. It's possible for a module author's defined types to be implemented in a way that they can correctly deserialize as well, but there are some trade-offs to consider. The type needs to have a default constructor, have all of its properties public, and have a `PSTypeConverter`. However, for already-defined types that the module author does not own, there is no way to "fix" them, hence the recommendation to avoid complex types for parameters all together. Runbook Authoring tip: If for some reason your cmdlets need to take a complex type parameter, or you are using someone else's module that requires a complex type parameter, the workaround in PowerShell Workflow runbooks and PowerShell Workflows in local PowerShell, is to wrap the cmdlet that generates the complex type and the cmdlet that consumes the complex type in the same `InlineScript` activity. Since `InlineScript` executes its contents as PowerShell rather than PowerShell Workflow, the cmdlet generating the complex type would produce that correct type, not the deserialized complex type.

5. Make all cmdlets in the module stateless. PowerShell Workflow runs every cmdlet called in the workflow in a different session. This means any cmdlets that depend on session state created / modified by other cmdlets in the same module will not work in PowerShell Workflow runbooks. Here is an example of what not to do.

```

$globalNum = 0
function Set-GlobalNum {
    param(
        [int] $num
    )

    $globalNum = $num
}
function Get-GlobalNumTimesTwo {
    $output = $globalNum * 2

    $output
}

```

6. The module should be fully contained in an Xcopy-able package. Because Azure Automation modules are distributed to the Automation sandboxes when runbooks need to execute, they need to work independently of the host they are running on. What this means is that you should be able to Zip up the module package, move it to any other host with the same or newer PowerShell version, and have it function as normal when imported into that host's PowerShell environment. In order for that to happen, the module should not depend on any files outside the module folder (the folder that gets zipped up when importing into Azure Automation), or on any unique registry settings on a host, such as those set by the install of a product. If this best practice is not followed, the module will not be usable in Azure Automation.

Next steps

- To get started with PowerShell workflow runbooks, see [My first PowerShell workflow runbook](#)
- To learn more about creating PowerShell Modules, see [Writing a Windows PowerShell Module](#)

Scheduling a runbook in Azure Automation

5/21/2018 • 5 minutes to read • [Edit Online](#)

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.

NOTE

Schedules do not currently support Azure Automation DSC configurations.

Windows PowerShell Cmdlets

The cmdlets in the following table are used to create and manage schedules with Windows PowerShell in Azure Automation. They ship as part of the [Azure PowerShell module](#).

| CMDLETS | DESCRIPTION |
|--|---|
| Get-AzureRmAutomationSchedule | Retrieves a schedule. |
| New-AzureRmAutomationSchedule | Creates a new schedule. |
| Remove-AzureRmAutomationSchedule | Removes a schedule. |
| Set-AzureRmAutomationSchedule | Sets the properties for an existing schedule. |
| Get-AzureRmAutomationScheduledRunbook | Retrieves scheduled runbooks. |
| Register-AzureRmAutomationScheduledRunbook | Associates a runbook with a schedule. |
| Unregister-AzureRmAutomationScheduledRunbook | Dissociates a runbook from a schedule. |

Creating a schedule

You can create a new schedule for runbooks in the Azure portal or with Windows PowerShell.

NOTE

Azure Automation uses the latest modules in your Automation account when a new scheduled job is run. To avoid impacting your runbooks and the processes they automate, you should first test any runbooks that have linked schedules with an Automation account dedicated for testing. This validates your scheduled runbooks continue to work correctly and if not, you can further troubleshoot and apply any changes required before migrating the updated runbook version to production. Your Automation account does not automatically get any new versions of modules unless you have updated them manually by selecting the [Update Azure Modules](#) option from the **Modules**.

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 **Recurrence**. If you select **Once** specify a **Start time**, and then click **Create**. If you select **Recurrence**, specify a **Start time** and the frequency for how often you want the runbook to repeat - by **hour**, **day**, **week**, or by **month**. If you select **week** or **month** from the drop-down list, the **Recurrence option** appears in the pane and upon selection, the **Recurrence option** pane is presented and you can select the day of week if you selected **week**. If you selected **month**, you can choose by **weekdays** or specific days of the month on the calendar and finally, do you want to run it on the last day of the month or not and then click **OK**.

To create a new schedule with Windows PowerShell

You use the [New-AzureRmAutomationSchedule](#) cmdlet to create schedules. You specify the start time for the schedule and the frequency it should run.

The following sample commands show how to create a schedule for the 15th and 30th of every month using an Azure Resource Manager cmdlet.

```
$automationAccountName = "MyAutomationAccount"
$scheduleName = "Sample-MonthlyDaysOfMonthSchedule"
New-AzureRMAutomationSchedule -AutomationAccountName $automationAccountName -Name ` 
$scheduleName -StartTime "7/01/2016 15:30:00" -MonthInterval 1 ` 
-DaysOfMonth Fifteenth,Thirtieth -ResourceGroupName "ResourceGroup01"
```

Linking a schedule to a runbook

A runbook can be linked to multiple schedules, and a schedule can have multiple runbooks linked to it. If a runbook has parameters, then you can provide values for them. You must provide values for any mandatory parameters and may provide values for any optional parameters. These values are used each time the runbook is started by this schedule. You can attach the same runbook to another schedule and specify different parameter values.

To link a schedule to a runbook with the Azure portal

1. In the Azure portal, from your automation account, select **Runbooks** under the section **Process Automation** on the left.
2. Click on the name of the runbook to schedule.
3. If the runbook is not currently linked to a schedule, then you are offered the option to create a new schedule or link to an existing schedule.
4. 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 accordingly.

To link a schedule to a runbook with Windows PowerShell

You can use the [Register-AzureRmAutomationScheduledRunbook](#) cmdlet to link a schedule. You can specify values for the runbook's parameters with the **Parameters** parameter. For more information on specifying parameter values, see [Starting a Runbook in Azure Automation](#). The following sample commands show how to link a schedule to a runbook using an Azure Resource Manager cmdlet with parameters.

```

$automationAccountName = "MyAutomationAccount"
$runbookName = "Test-Runbook"
$scheduleName = "Sample-DailySchedule"
$params = @{"FirstName"="Joe";"LastName"="Smith";"RepeatCount"=2;"Show"=$true}
Register-AzureRmAutomationScheduledRunbook -AutomationAccountName $automationAccountName ` 
-Name $runbookName -ScheduleName $scheduleName -Parameters $params ` 
-ResourceGroupName "ResourceGroup01"

```

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:

- 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.
- 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.

Disabling a schedule

When you disable a schedule, any runbook linked to it no longer runs on that schedule. You can manually disable a schedule or set an expiration time for schedules with a frequency when you create them. When the expiration time is reached, the schedule is disabled.

To disable a schedule from the Azure portal

1. In the Azure portal, from your Automation account, select **Schedules** under the section **Shared Resources** on the left.
2. Click the name of a schedule to open the details pane.
3. Change **Enabled** to **No**.

To disable a schedule with Windows PowerShell

You can use the [Set-AzureRmAutomationSchedule](#) cmdlet to change the properties of an existing schedule. To disable the schedule, specify **false** for the **IsEnabled** parameter.

The following sample commands show how to disable a schedule for a runbook using an Azure Resource Manager cmdlet.

```

$automationAccountName = "MyAutomationAccount"
$scheduleName = "Sample-MonthlyDaysOfMonthSchedule"
Set-AzureRmAutomationSchedule -AutomationAccountName $automationAccountName ` 
-Name $scheduleName -IsEnabled $false -ResourceGroupName "ResourceGroup01"

```

Next steps

- To get started with runbooks in Azure Automation, see [Starting a Runbook in Azure Automation](#)

Variable assets in Azure Automation

5/21/2018 • 7 minutes to read • [Edit Online](#)

Variable assets are values that are available to all runbooks and DSC configurations in your automation account. They can be created, modified, and retrieved from the Azure portal, Windows PowerShell, and from within a runbook or DSC configuration. 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 Windows 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, etc.

Automation variables are persisted so that they continue to be available even if the runbook or DSC configuration fails. This also allows a value to be set by one runbook that is then used by another, or is used by the same runbook or DSC configuration the next time that it is run.

When a variable is created, you can specify that it is stored encrypted. When a variable is encrypted, it is stored securely in Azure Automation, and its value cannot be retrieved from the [Get-AzureRmAutomationVariable](#) cmdlet that ships as part of the Azure PowerShell module. The only way that an encrypted value can be retrieved is from the **Get-AutomationVariable** activity in a runbook or DSC configuration.

NOTE

Secure assets in Azure Automation include credentials, certificates, connections, and encrypted variables. These assets are encrypted and stored in Azure Automation using a unique key that is generated for each automation account. This key is stored in Key Vault. Before storing a secure asset, the key is loaded from Key Vault and then used to encrypt the asset.

Variable types

When you create a variable with the Azure portal, you must specify a data type from the drop-down list so the portal can display the appropriate control for entering the variable value. The variable is not restricted to this data type, but you must set the variable using Windows PowerShell if you want to specify a value of a different type. If you specify **Not defined**, then the value of the variable is set to **\$null**, and you must set the value with the [Set-AzureRmAutomationVariable](#) cmdlet or **Set-AutomationVariable** activity. You cannot create or change the value for a complex variable type in the portal, but you can provide a value of any type using Windows PowerShell. Complex types are returned as a [PSCustomObject](#).

You can store multiple values to a single variable by creating an array or hashtable and saving it to the variable.

The following are a list of variable types available in Automation:

- String
- Integer
- DateTime
- Boolean
- Null

AzureRM PowerShell cmdlets

For AzureRM, the cmdlets in the following table are used to create and manage automation credential assets with Windows PowerShell. They ship as part of the [AzureRM.Automation module](#) which is available for use in Automation runbooks and DSC configurations.

| CMDLETS | DESCRIPTION |
|--|--|
| Get-AzureRmAutomationVariable | Retrieves the value of an existing variable. |
| New-AzureRmAutomationVariable | Creates a new variable and sets its value. |
| Remove-AzureRmAutomationVariable | Removes an existing variable. |
| Set-AzureRmAutomationVariable | Sets the value for an existing variable. |

Activities

The activities in the following table are used to access credentials in a runbook and DSC configurations.

| ACTIVITIES | DESCRIPTION |
|--|--|
| Get-AutomationVariable | Retrieves the value of an existing variable. |
| Set-AutomationVariable | Sets the value for an existing variable. |

NOTE

You should avoid using variables in the `-Name` parameter of **Get-AutomationVariable** in a runbook or DSC configuration since this can complicate discovering dependencies between runbooks or DSC configuration, and Automation variables at design time.

The functions in the following table are used to access and retrieve variables in a Python2 runbook.

| PYTHON2 FUNCTIONS | DESCRIPTION |
|---|--|
| <code>automationassets.get_automation_variable</code> | Retrieves the value of an existing variable. |
| <code>automationassets.set_automation_variable</code> | Sets the value for an existing variable. |

NOTE

You must import the "automationassets" module at the top of your Python runbook in order to access the asset functions.

Creating a new Automation variable

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 variable with Windows PowerShell

The [New-AzureRmAutomationVariable](#) cmdlet creates a new variable and sets its initial value. You can retrieve

the value using [Get-AzureRmAutomationVariable](#). If the value is a simple type, then that same type is returned. If it's a complex type, then a **PSCustomObject** is returned.

The following sample commands show how to create a variable of type string and then return its value.

```
New-AzureRmAutomationVariable -ResourceGroupName "ResourceGroup01" -AutomationAccountName "MyAutomationAccount" -Name 'MyStringVariable' -Encrypted $false -Value 'My String'  
$string = (Get-AzureRmAutomationVariable -ResourceGroupName "ResourceGroup01" -AutomationAccountName "MyAutomationAccount" -Name 'MyStringVariable').Value
```

The following sample commands show how to create a variable with a complex type and then return its properties. In this case, a virtual machine object from **Get-AzureRmVm** is used.

```
$vm = Get-AzureRmVm -ResourceGroupName "ResourceGroup01" -Name "VM01"  
New-AzureRmAutomationVariable -AutomationAccountName "MyAutomationAccount" -Name "MyComplexVariable" -Encrypted $false -Value $vm  
  
$vmValue = (Get-AzureRmAutomationVariable -ResourceGroupName "ResourceGroup01" -AutomationAccountName "MyAutomationAccount" -Name "MyComplexVariable").Value  
$vmName = $vmValue.Name  
$vmIpAddress = $vmValue.IpAddress
```

Using a variable in a runbook or DSC configuration

Use the **Set-AutomationVariable** activity to set the value of an Automation variable in a PowerShell runbook or DSC configuration, and the **Get-AutomationVariable** to retrieve it. You shouldn't use the **Set-AzureRMAutomationVariable** or **Get-AzureRMAutomationVariable** cmdlets in a runbook or DSC configuration since they are less efficient than the workflow activities. You also cannot retrieve the value of secure variables with **Get-AzureRMAutomationVariable**. The only way to create a new variable from within a runbook or DSC configuration is to use the [New-AzureRMAutomationVariable](#) cmdlet.

Textual runbook samples

Setting and retrieving a simple value from a variable

The following sample commands show how to set and retrieve a variable in a textual runbook. In this sample, it is assumed that variables of type integer named *NumberOfIterations* and *NumberOfRunnings* and a variable of type string named *SampleMessage* have already been created.

```
$NumberOfIterations = Get-AzureRmAutomationVariable -ResourceGroupName "ResourceGroup01" -AutomationAccountName "MyAutomationAccount" -Name 'NumberOfIterations'  
$NumberOfRunnings = Get-AzureRmAutomationVariable -ResourceGroupName "ResourceGroup01" -AutomationAccountName "MyAutomationAccount" -Name 'NumberOfRunnings'  
$SampleMessage = Get-AutomationVariable -Name 'SampleMessage'  
  
Write-Output "Runbook has been run $NumberOfRunnings times."  
  
for ($i = 1; $i -le $NumberOfIterations; $i++) {  
    Write-Output "$i`:` $SampleMessage"  
}  
Set-AzureRmAutomationVariable -ResourceGroupName "ResourceGroup01" -AutomationAccountName "MyAutomationAccount" -Name NumberOfRunnings -Value ($NumberOfRunnings += 1)
```

Setting and retrieving a complex object in a variable

The following sample code shows how to update a variable with a complex value in a textual runbook. In this sample, an Azure virtual machine is retrieved with **Get-AzureVM** and saved to an existing Automation variable. As explained in [Variable types](#), this is stored as a PSCustomObject.

```
$vm = Get-AzureVM -ServiceName "MyVM" -Name "MyVM"
Set-AutomationVariable -Name "MyComplexVariable" -Value $vm
```

In the following code, the value is retrieved from the variable and used to start the virtual machine.

```
$vmObject = Get-AutomationVariable -Name "MyComplexVariable"
if ($vmObject.PowerState -eq 'Stopped') {
    Start-AzureVM -ServiceName $vmObject.ServiceName -Name $vmObject.Name
}
```

Setting and retrieving a collection in a variable

The following sample code shows how to use a variable with a collection of complex values in a textual runbook. In this sample, multiple Azure virtual machines are retrieved with **Get-AzureVM** and saved to an existing Automation variable. As explained in [Variable types](#), this is stored as a collection of `PSCustomObjects`.

```
$vms = Get-AzureVM | Where -FilterScript {$_.Name -match "my"}
Set-AutomationVariable -Name 'MyComplexVariable' -Value $vms
```

In the following code, the collection is retrieved from the variable and used to start each virtual machine.

```
$vmValues = Get-AutomationVariable -Name "MyComplexVariable"
ForEach ($vmValue in $vmValues)
{
    if ($vmValue.PowerState -eq 'Stopped') {
        Start-AzureVM -ServiceName $vmValue.ServiceName -Name $vmValue.Name
    }
}
```

Setting and retrieving a variable in Python2

The following sample code shows how to use a variable, set a variable, and handle an exception for a non-existent variable in a Python2 runbook.

```
import automationassets
from automationassets import AutomationAssetNotFound

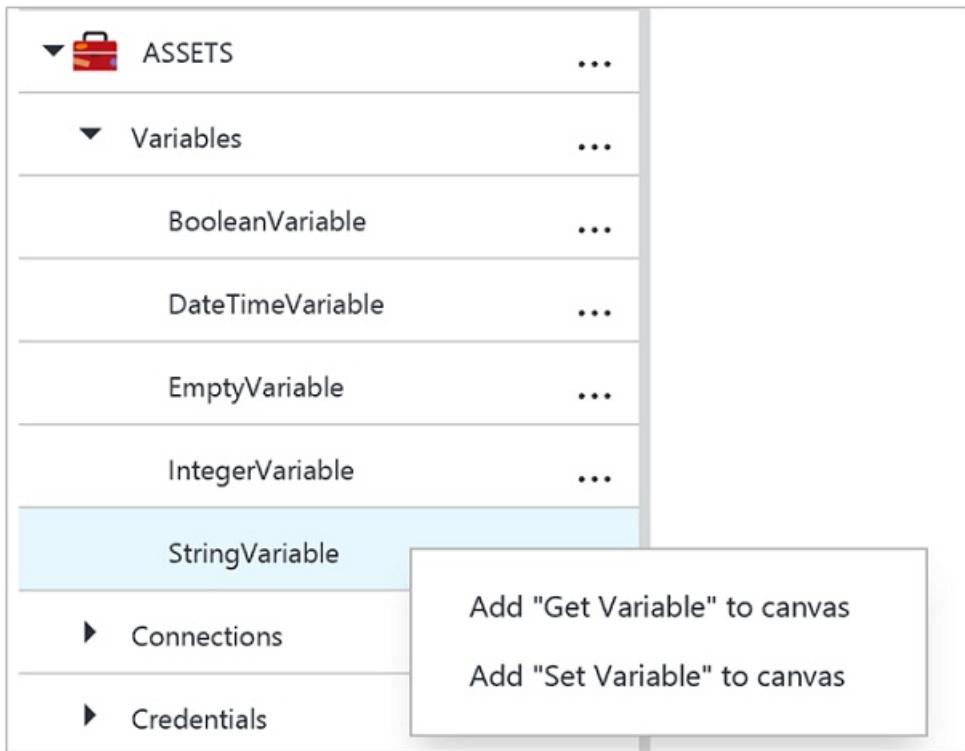
# get a variable
value = automationassets.get_automation_variable("test-variable")
print value

# set a variable (value can be int/bool/string)
automationassets.set_automation_variable("test-variable", True)
automationassets.set_automation_variable("test-variable", 4)
automationassets.set_automation_variable("test-variable", "test-string")

# handle a non-existent variable exception
try:
    value = automationassets.get_automation_variable("non-existing variable")
except AutomationAssetNotFound:
    print "variable not found"
```

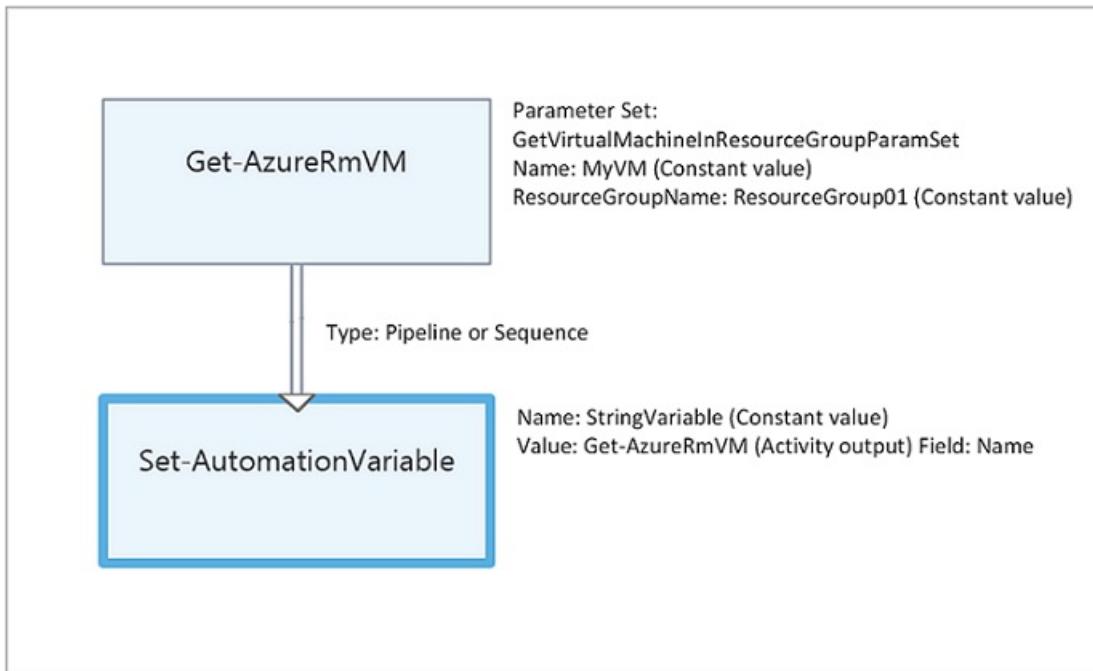
Graphical runbook samples

In a graphical runbook, you add the **Get-AutomationVariable** or **Set-AutomationVariable** by right-clicking on the variable in the Library pane of the graphical editor and selecting the activity you want.



Setting values in a variable

The following image shows sample activities to update a variable with a simple value in a graphical runbook. In this sample, a single Azure virtual machine is retrieved with **Get-AzureRmVM** and the computer name is saved to an existing Automation variable with a type of String. It doesn't matter whether the [link is a pipeline or sequence](#) since you only expect a single object in the output.



Next Steps

- To learn more about connecting activities together in graphical authoring, see [Links in graphical authoring](#)
- To get started with Graphical runbooks, see [My first graphical runbook](#)

How to update Azure PowerShell modules in Azure Automation

7/9/2018 • 2 minutes to read • [Edit Online](#)

The most common Azure PowerShell modules are provided by default in each Automation account. The Azure team updates the Azure modules regularly, so in the Automation account you are provided a way to update the modules in the account when new versions are available from the portal.

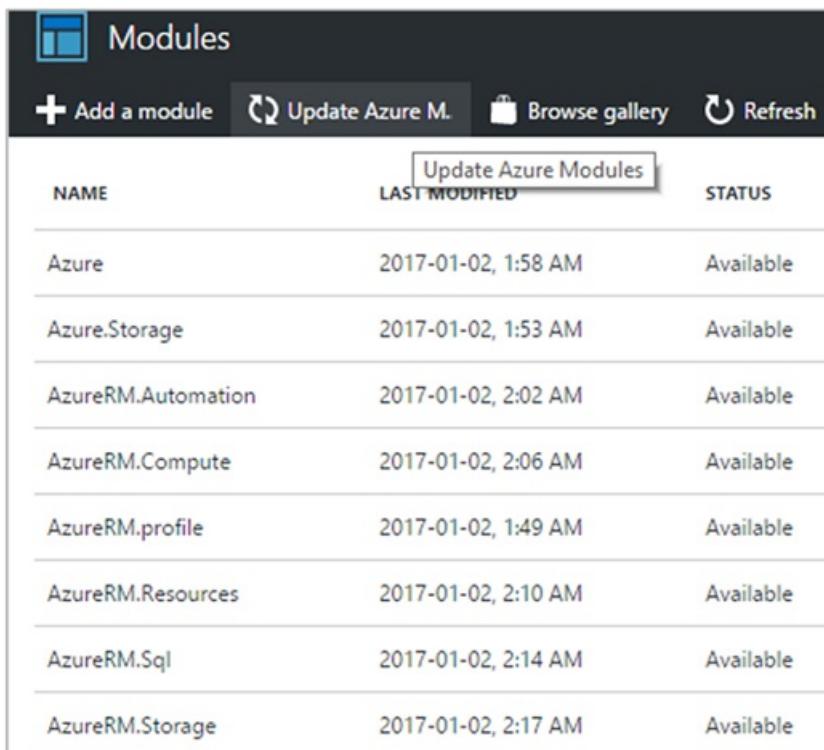
Because modules are updated regularly by the product group, changes can occur with the included cmdlets, which may negatively impact your runbooks depending on the type of change, such as renaming a parameter or deprecating a cmdlet entirely. To avoid impacting your runbooks and the processes they automate, it is recommended that you test and validate before proceeding. If you do not have a dedicated Automation account intended for this purpose, consider creating one so that you can test many different scenarios and permutations during the development of your runbooks, in addition to iterative changes such as updating the PowerShell modules. After the results are validated and you have applied any changes required, proceed with coordinating the migration of any runbooks that required modification and perform the following update as described in production.

NOTE

A new Automation account might not contain the latest modules.

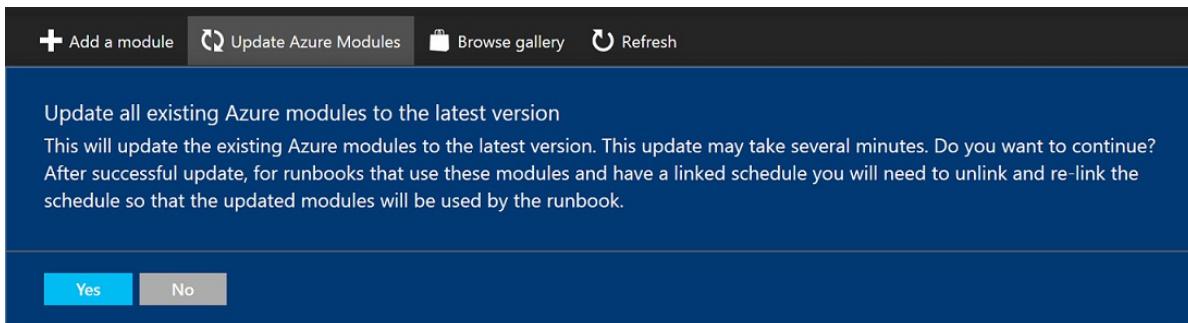
Updating Azure Modules

1. In the Modules page of your Automation account, there is an option called **Update Azure Modules**. It is always enabled.



| NAME | LAST MODIFIED | STATUS |
|--------------------|---------------------|-----------|
| Azure | 2017-01-02, 1:58 AM | Available |
| Azure.Storage | 2017-01-02, 1:53 AM | Available |
| AzureRM.Automation | 2017-01-02, 2:02 AM | Available |
| AzureRM.Compute | 2017-01-02, 2:06 AM | Available |
| AzureRM.profile | 2017-01-02, 1:49 AM | Available |
| AzureRM.Resources | 2017-01-02, 2:10 AM | Available |
| AzureRM.Sql | 2017-01-02, 2:14 AM | Available |
| AzureRM.Storage | 2017-01-02, 2:17 AM | Available |

2. Click **Update Azure Modules**, a confirmation notification is shown that asks if you want to continue.



3. Click **Yes** and the module update process begins. The update process takes about 15-20 minutes to update the following modules:

- Azure
- Azure.Storage
- AzureRm.Automation
- AzureRm.Compute
- AzureRm.Profile
- AzureRm.Resources
- AzureRm.Sql
- AzureRm.Storage

If the modules are already up-to-date, then the process completes in a few seconds. When the update process completes, you are notified.

The screenshot shows the 'Modules' page in the Azure portal. The top navigation bar includes 'Add a module', 'Update Azure M.', 'Browse gallery', and 'Refresh'. A green header bar displays the message 'Azure modules have been updated'. Below this is a table listing the updated modules:

| NAME | LAST MODIFIED | STATUS |
|--------------------|---------------------|-----------|
| Azure | 2017-02-11, 3:07 PM | Available |
| Azure.Storage | 2017-02-11, 3:00 PM | Available |
| AzureRM.Automation | 2017-02-11, 3:00 PM | Available |
| AzureRM.Compute | 2017-02-11, 3:00 PM | Available |
| AzureRM.profile | 2017-02-11, 3:01 PM | Available |
| AzureRM.Resources | 2017-02-11, 3:02 PM | Available |
| AzureRM.Sql | 2017-02-11, 3:00 PM | Available |
| AzureRM.Storage | 2017-02-11, 3:06 PM | Available |

NOTE

Azure Automation uses the latest modules in your Automation account when a new scheduled job is run.

If you use cmdlets from these Azure PowerShell modules in your runbooks, you want to run this update process

every month or so to make sure that you have the latest modules. Azure Automation uses the AzureRunAsConnection connection to authenticate when updating the modules, if the service principal is expired or no longer exists on the subscription level, the module update will fail.

Next steps

- To learn more about Integration Modules and how to create custom modules to further integrate Automation with other systems, services, or solutions, see [Integration Modules](#).
- Consider source control integration using [GitHub Enterprise](#) or [Visual Studio Team Services](#) to centrally manage and control releases of your Automation runbook and configuration portfolio.

Runbook and module galleries for Azure Automation

5/21/2018 • 5 minutes to read • [Edit Online](#)

Rather than creating your own runbooks and modules in Azure Automation, you can access a variety of scenarios that have already been built by Microsoft and the community. You can either use these scenarios without modification or you can use them as a starting point and edit them for your specific requirements.

You can get runbooks from the [Runbook Gallery](#) and modules from the [PowerShell Gallery](#). You can also contribute to the community by sharing scenarios that you develop.

Runbooks in Runbook Gallery

The [Runbook Gallery](#) provides a variety of runbooks from Microsoft and the community that you can import into Azure Automation. You can either download a runbook from the gallery, which is hosted in the [TechNet Script Center](#), or you can directly import runbooks from the gallery in the Azure portal.

You can only import directly from the Runbook Gallery using the Azure portal. You cannot perform this function using Windows PowerShell.

NOTE

You should validate the contents of any runbooks that you get from the Runbook Gallery and use extreme caution in installing and running them in a production environment.

To import a runbook from the Runbook Gallery with the Azure portal

1. In the Azure portal, open your Automation account.
2. Under **Process Automation**, click on **Runbooks gallery**
3. Locate the gallery item you want and select it to view its details. On the left you can enter additional search parameters for the publisher and type.

The screenshot shows the Azure Runbook Gallery interface. On the left, there is a search bar and a filter section with dropdowns for 'Popularity' and 'Type'. The 'Type' dropdown has three checked options: 'PowerShell script', 'Graphical runbook', and 'PowerShell workflow'. To the right of the search bar, there is a 'Gallery Source' dropdown set to 'Script Center'. Below these are sections for 'Publisher' (with 'Microsoft' checked) and 'Community'. A large list of runbooks is displayed on the right, each with a preview icon, name, description, and details like 'Created by', 'Ratings', 'Downloads', and 'Last updated'. At the bottom right of the interface is a blue 'OK' button.

| Name | Description | Created by | Ratings | Downloads | Last updated |
|--|--|----------------------------|-----------|-----------|--------------|
| Start Azure V2 VMs | This Graphical PowerShell runbook connects to Azure using an Automation Run As account and starts all V2 VMs in an Azure subscription or in a resource group or a single named V2 VM. You can attach a recurring schedule to this runbook to run it at a specific time. The asso | SC Automation Product Team | 4.27 of 5 | 41,631 | 10/23/2016 |
| Stop Azure V2 VMs | This Graphical PowerShell runbook connects to Azure using an Automation Run As account and stops all V2 VMs in an Azure subscription or in a resource group or a single named V2 VM. You can attach a recurring schedule to this runbook to run it at a specific time. | SC Automation Product Team | 4.25 of 5 | 46,511 | 10/23/2016 |
| Scheduled Virtual Machine Shutdown/Startup | Automates the scheduled startup and shutdown of Azure virtual machines. Schedules are implemented by tagging VMs or resource groups with individual simple schedules. Schedules can define multiple time periods for shutdown, including time ranges and days of week or dates. | Automys | 4.81 of 5 | 34,144 | 2/29/2016 |
| Stop-Start-AzureVM | This PowerShell Workflow runbook connects to Azure using an Automation Credential and Starts/Stops a VM/a list of VMs/All VMs in a Subscription in-parallel. | Pradeban Raja | 3.55 of 5 | 355 | 1/10/2018 |
| Shutdown/Start VMs by tag | | | | | |

4. Click on **View source project** to view the item in the [TechNet Script Center](#).
5. To import an item, click on it to view its details and then click the **Import** button.

Start Azure V2 VMs

[View Source](#)

Import

This Graphical PowerShell runbook connects to Azure using an Automation Run As account and starts all V2 VMs in an Azure subscription or in a resource group or a single named V2 VM. You can attach a recurring schedule to this runbook to run it at a specific time. The associated runbook has been created by SC Automation Product Team - Microsoft.

Tags: Azure Virtual Machines, Start VM, GraphicalPS

Ratings: 4.27 of 5
41,631 downloads
Last updated: 10/23/2016

[View Source Project](#)

```

graph TD
    A[Get Run As Connection] --> B[Connect to Azure]
    B --> C[Get single VM]
    B --> D[Get all VMs in RG]
    B --> E[Get all VMs in Sub]
    C --> B
    D --> B
    E --> B
    B --> F[READ ME]
  
```

ATTENTION
Each runbook is licensed to you under a license agreement by its owner, not Microsoft. Microsoft is not responsible for runbooks provided & licensed by the community members and does not screen for security, compatibility or performance. The runbooks are not supported under any Microsoft support program or service. The runbooks are provided AS IS without warranty of any kind.

6. Optionally, change the name of the runbook and then click **OK** to import the runbook.

7. The runbook appears on the **Runbooks** tab for the Automation Account.

Adding a runbook to the runbook gallery

Microsoft encourages you to add runbooks to the Runbook Gallery that you think would be useful to other customers. You can add a runbook by [uploading it to the Script Center](#) taking into account the following details.

- You must specify *Windows Azure* for the **Category** and *Automation* for the **Subcategory** for the runbook to be displayed in the wizard.
- The upload must be a single .ps1 or .graphrunbook file. If the runbook requires any modules, child runbooks, or assets, then you should list those in the description of the submission and in the comments section of the runbook. If you have a scenario requiring multiple runbooks, then upload each separately and list the names of the related runbooks in each of their descriptions. Make sure that you use the same tags so that they show up in the same category. A user will have to read the description to know that other runbooks are required for the scenario to work.
- Add the tag "GraphicalPS" if you are publishing a **Graphical runbook** (not a Graphical Workflow).
- Insert either a PowerShell or PowerShell Workflow code snippet into the description using **Insert code section** icon.
- The Summary for the upload is displayed in the Runbook Gallery results so you should provide detailed information that helps a user identify the functionality of the runbook.
- You should assign one to three of the following Tags to the upload. The runbook is listed in the wizard under the categories that match its tags. Any tags not on this list are ignored by the wizard. If you don't specify any matching tags, the runbook is listed under the Other category.
 - Backup
 - Capacity Management
 - Change Control
 - Compliance
 - Dev / Test Environments

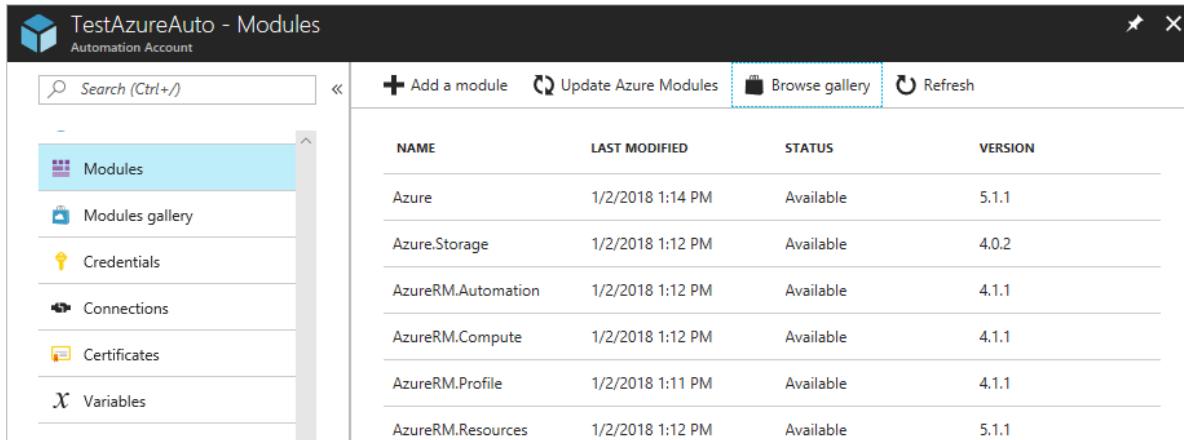
- Disaster Recovery
- Monitoring
- Patching
- Provisioning
- Remediation
- VM Lifecycle Management
- Automation updates the Gallery once an hour, so you won't see your contributions immediately.

Modules in PowerShell Gallery

PowerShell modules contain cmdlets that you can use in your runbooks, and existing modules that you can install in Azure Automation are available in the [PowerShell Gallery](#). You can launch this gallery from the Azure portal and install them directly into Azure Automation or you can download them and install them manually.

To import a module from the Automation Module Gallery with the Azure portal

1. In the Azure portal, open your Automation account.
2. Select **Modules** under **Shared Resources** to open the list of modules.
3. Click **Browse gallery** from the top of the page.



The screenshot shows the 'TestAzureAuto - Modules' blade in the Azure portal. On the left, there's a sidebar with links: 'Modules' (which is selected and highlighted in blue), 'Modules gallery', 'Credentials', 'Connections', 'Certificates', and 'Variables'. At the top right, there are buttons for 'Add a module', 'Update Azure Modules', 'Browse gallery' (which has a dashed blue border around it), and 'Refresh'. Below these buttons is a table with columns: NAME, LAST MODIFIED, STATUS, and VERSION. The table lists several modules:

| NAME | LAST MODIFIED | STATUS | VERSION |
|--------------------|------------------|-----------|---------|
| Azure | 1/2/2018 1:14 PM | Available | 5.1.1 |
| Azure.Storage | 1/2/2018 1:12 PM | Available | 4.0.2 |
| AzureRM.Automation | 1/2/2018 1:12 PM | Available | 4.1.1 |
| AzureRM.Compute | 1/2/2018 1:12 PM | Available | 4.1.1 |
| AzureRM.Profile | 1/2/2018 1:11 PM | Available | 4.1.1 |
| AzureRM.Resources | 1/2/2018 1:12 PM | Available | 5.1.1 |

4. On the **Browse gallery** page, you can search by the following fields:

- Module Name
- Tags
- Author
- Cmdlet/DSC resource name

5. Locate a module that you're interested in and select it to view its details.

When you drill into a specific module, you can view more information about the module, including a link back to the PowerShell Gallery, any required dependencies, and all of the cmdlets and/or DSC resources that the module contains.

The screenshot shows the 'Import' page for the 'Posh-SSH' module on the PowerShell Gallery. At the top, it says 'Provide SSH and SCP functionality for executing commands against remote hosts.' Below this, there's information about the module: 'Created by: cperez', 'Tags: PSModule', 'View Source Project', 'Version: 2.0.2', '320,993 downloads', and 'Last updated: 10/13/2017'. There are links to 'Learn more', 'View in PowerShell Gallery', 'Documentation', and 'Licensing Information (PowerShell Gallery Default)'. A section titled 'Content' contains a table with cmdlet names:

| TYPE | NAME |
|--------|-----------------|
| Cmdlet | Get-SCPFile |
| Cmdlet | Get-SCPFolder |
| Cmdlet | Get-SFTPFile |
| Cmdlet | Set-SFTPFile |
| Cmdlet | New-SFTPSession |
| Cmdlet | New-SSHSession |
| Cmdlet | Set-SCPFile |

6. To install the module directly into Azure Automation, click the **Import** button.
7. When you click the Import button, on the **Import** pane, you see the module name that you are about to import. If all the dependencies are installed, the **OK** button is activated. If you are missing dependencies, you need to import those before you can import this module.
8. On the **Import** page, click **OK** to import the module. While Azure Automation imports a module to your account, it extracts metadata about the module and the cmdlets. This may take a couple of minutes since each activity needs to be extracted.
9. You receive an initial notification that the module is being deployed and another notification when it has completed.
10. After the module is imported, you can see the available activities, and you can use its resources in your runbooks and Desired State Configuration.

NOTE

Modules that only support PowerShell core are not supported in Azure Automation and are unable to be imported in the Azure portal, or deployed directly from the PowerShell Gallery.

Requesting a runbook or module

You can send requests to [User Voice](#). If you need help writing a runbook or have a question about PowerShell, post a question to our [forum](#).

Next Steps

- To get started with runbooks, see [Creating or importing a runbook in Azure Automation](#)
- To understand the differences between PowerShell and PowerShell Workflow with runbooks, see [Learning PowerShell workflow](#)

Monitoring runbooks with metric alerts

5/25/2018 • 4 minutes to read • [Edit Online](#)

In this article, you learn how to create alerts based on the completion status of runbooks.

Log in to Azure

Log in to Azure at <https://portal.azure.com>

Create alert

Alerts allow you to define a condition to monitor for and an action to take when that condition is met.

In the Azure portal, navigate to **All services** and select **Monitor**. On the Monitor page, select **Alerts** and click **+ New Alert Rule**.

Define the alert condition

- Under **1. Define alert condition**, click **+ Select target**. Choose your subscription, and under **Filter by resource type**, select **Automation Accounts**. Choose your Automation Account and click **Done**.

The screenshot shows the 'Create rule' wizard in the Azure portal. The main left pane shows steps 1, 2, and 3: '1. Define alert condition', '2. Define alert details', and '3. Define action group'. Step 1 is expanded, showing 'Alert target' (selected) and 'Target Hierarchy'. Below it is a section for 'Alert criteria' with a note: 'No criteria defined, click on 'Add criteria' to select a signal and define its logic'. A 'Select target' button is visible. The right pane is a 'Select a resource' dialog. It has a search bar and dropdowns for 'Filter by subscription' (set to 'Microsoft Azure') and 'Filter by resource type' (set to 'Automation Accounts'). The 'RESOURCE' list shows 'Microsoft Azure' expanded, with 'DefaultResourceGroup-EUS' and 'ExampleAutoAccount' listed. 'ExampleAutoAccount' is selected. At the bottom of the dialog are 'Selection preview' and 'Available signal(s) : Metric, Activity Log' sections, and a 'Done' button.

Configure alert criteria

- Click **+ Add criteria**. Select **Metrics** for the **Signal type**, and choose **Total Jobs** from the table.
- The **Configure signal logic** page is where you define the logic that triggers the alert. Under the historical graph you are presented with two dimensions, **Runbook Name** and **Status**. Dimensions are different properties for a metric that can be used to filter results. For **Runbook Name**, select the runbook you want to alert on or leave blank to alert on all runbooks. For **Status**, select a status from the drop-down you want to monitor for. The runbook name and status values that appear in the dropdown are only for jobs that have ran in the past week.

If you want to alert on a status or runbook that is not shown in the dropdown, click the **+** next to the dimension. This opens a dialog that allows you to enter in a custom value, which has not emitted for that

dimension recently. If you enter a value that doesn't exist for a property your alert will not be triggered.

3. Under **Alert logic**, define the condition and threshold for your alert. A preview of your condition defined is shown underneath.
4. Under **Evaluated based on** select the timespan for the query and how often you want that query ran. For example, if you choose **Over the last 5 minutes** for **Period** and **Every 1 Minute** for **Frequency**, the alert looks for the number of runbooks that met your criteria over the past 5 minutes. This query is ran every minute, and once the alert criteria you defined is no longer found in a 5-minute window, the alert resolves itself. When finished, click **Done**.

Configure signal logic

This metric supports dimensions. Selecting the dimension values will help you filter to the right time series. If you do not select any value for a dimension, that dimension will be ignored.

| DIMENSION NAME | DIMENSION VALUES |
|----------------|------------------|
| Runbook Name | Hello-World |
| Status | Failed |

Alert logic

Condition **Greater than** Time Aggregation **Total** Threshold **0** count

Condition preview
Whenever the total jobs is greater than 0 count

Evaluated based on

Period **Over the last 5 minutes** Frequency **Every 1 Minute**

Done

Define alert details

1. Under **2. Define alert details**, give the alert a friendly name and description. Set the **Severity** to match your alert condition. There are five severities ranging from 0 to 5. The alerts are treated the same independent of the severity, you can match the severity to match your business logic.
2. At the bottom of the section is a button that allows you to enable the rule upon completion. By default rules are enabled at creation. If you select No, you can create the alert and it is created in a **Disabled** state. From the **Rules** page in Azure Monitor, you can select it and click **Enable** to enable the alert when you are ready.

Define the action to take

1. Under **3. Define action group**, click **+ New action group**. An action group is a group of actions that you can use across multiple alerts. These can include but are not limited to, email notifications, runbooks, webhooks, and many more. To learn more about action groups, see [Create and manage action groups](#)
2. In the **Action group name** box, give it a friendly name and short name. The short name is used in place of a full action group name when notifications are sent using this group.
3. In the **Actions** section under **ACTION TYPE**, select **Email/SMS/Push/Voice**.
4. On the **Email/SMS/Push/Voice** page, give it a name. Check the **Email** checkbox and enter in a valid email address to be used.

The screenshot shows two adjacent windows. The left window is titled 'Add action group' and contains fields for 'Action group name' (Email Automation Administrators), 'Short name' (emailauto), 'Subscription' (Microsoft Azure), and 'Resource group' (Default-ActivityLogAlerts). Below these are sections for 'Actions' (with a table showing one entry: Action Name Email Notification, Action Type Email/SMS/Push/Voice, Status Pending, Details Edit details), 'Privacy Statement', and 'Pricing'. At the bottom are 'OK' and 'Cancel' buttons. The right window is titled 'Email/SMS/Push/Voice' and contains fields for 'Name' (Email Notification), 'Email' (administrator@contoso.com checked), 'SMS' (unchecked), 'Country code' (1) and 'Phone number' (1234567890), a note about carrier charges, 'Azure app Push Notifications' (unchecked), a link to learn about connecting to Azure resources, an email address (email@example.com), a note about logging into the Azure account, 'Voice' (unchecked), 'Country code' (1) and 'Phone number' (1234567890), and an 'OK' button.

- Click **OK** on the **Email/SMS/Push/Voice** page to close it and click **OK** to close the **Add action group** page. The Name specified in this page is saved as the **ACTION NAME**.
- When complete, click **Save**. This creates the rule that alerts you when a runbook completed with a certain status.

NOTE

When adding an email address to an Action Group, a notification email is sent stating the address has been added to an Action Group.

Notification

When the alert criteria is met, the action group runs the action defined. In this article's example, an email is sent. The following image is an example of an email you receive after the alert is triggered:

 Wed 5/16/2018 11:19 AM
Azure Email Service No Reply Account
Azure: Activated Severity: 2 Runbooks failed

To ● Administrator

⚠ Azure monitoring rule 'Runbooks failed'
was activated for
'automationAccounts/ExampleAutoAccount'
at 5/16/2018 6:18:50 PM (UTC)

You can view more details for this alert in the [Microsoft Azure Management Portal](#).

RULE ID: /subscriptions/
/resourceGroups/ExampleAutoAccount/providers/microsoft.insights/metricAlerts/Runbooks%20failed

RESOURCE ID: /subscriptions/
/resourceGroups/ExampleAutoAccount/providers/Microsoft.Automation/automationAccounts/ExampleAutoAccount

CRITERIA: All of the following

Criteria
All of the following:
TotalJob GreaterThan0

Thank you,
Microsoft Azure Team

Once the metric is no longer outside of the threshold defined, the alert is deactivated and the action group runs the action defined. If an email action type is selected, a resolution email is sent stating it has been resolved.

Next steps

Continue to the following article to learn about other ways that you can integrate alertings into your Automation Account.

[Use an alert to trigger an Azure Automation runbook](#)

Use an alert to trigger an Azure Automation runbook

5/21/2018 • 7 minutes to read • [Edit Online](#)

You can use [Azure Monitor](#) to monitor base-level metrics and logs for most services in Azure. You can call Azure Automation runbooks by using [action groups](#) or by using classic alerts to automate tasks based on alerts. This article shows you how to configure and run a runbook by using alerts.

Alert types

You can use automation runbooks with three alert types:

- Classic metric alerts
- Activity log alerts
- Near real-time metric alerts

When an alert calls a runbook, the actual call is an HTTP POST request to the webhook. The body of the POST request contains a JSON-formatted object that has useful properties that are related to the alert. The following table lists links to the payload schema for each alert type:

| ALERT | DESCRIPTION | PAYOUT SCHEMA |
|-----------------------------|--|--|
| Classic metric alert | Sends a notification when any platform-level metric meets a specific condition. For example, when the value for CPU % on a VM is greater than 90 for the past 5 minutes. | Class metric alert payload schema |
| Activity log alert | Sends a notification when any new event in the Azure activity log matches specific conditions. For example, when a Delete VM operation occurs in myProductionResourceGroup or when a new Azure Service Health event with an Active status appears. | Activity log alert payload schema |
| Near real-time metric alert | Sends a notification faster than metric alerts when one or more platform-level metrics meet specified conditions. For example, when the value for CPU % on a VM is greater than 90 , and the value for Network In is greater than 500 MB for the past 5 minutes. | Near real-time metric alert payload schema |

Because the data that's provided by each type of alert is different, each alert type is handled differently. In the next section, you learn how to create a runbook to handle different types of alerts.

Create a runbook to handle alerts

To use Automation with alerts, you need a runbook that has logic that manages the alert JSON payload that's passed to the runbook. The following example runbook must be called from an Azure alert.

As described in the preceding section, each type of alert has a different schema. The script takes in the webhook data in the `WebhookData` runbook input parameter from an alert. Then, the script evaluates the JSON payload to

determine which alert type was used.

This example uses an alert from a VM. It retrieves the VM data from the payload, and then uses that information to stop the VM. The connection must be set up in the Automation account where the runbook is run.

The runbook uses the **AzureRunAsConnection** Run As account to authenticate with Azure to perform the management action against the VM.

Use this example to create a runbook called **Stop-AzureVmInResponsetoVMAlert**. You can modify the PowerShell script, and use it with many different resources.

1. Go to your Azure Automation account.
2. Under **PROCESS AUTOMATION**, select **Runbooks**.
3. At the top of the list of runbooks, select **Add a runbook**.
4. On the **Add Runbook** page, select **Quick Create**.
5. For the runbook name, enter **Stop-AzureVmInResponsetoVMAlert**. For the runbook type, select **PowerShell**. Then, select **Create**.
6. Copy the following PowerShell example into the **Edit** pane.

```
<#
.SYNOPSIS
This runbook stops a resource management VM in response to an Azure alert trigger.

.DESCRIPTION
This runbook stops a resource management VM in response to an Azure alert trigger.
The input is alert data that has the information required to identify which VM to stop.

DEPENDENCIES
- The runbook must be called from an Azure alert via a webhook.

REQUIRED AUTOMATION ASSETS
- An Automation connection asset called "AzureRunAsConnection" that is of type AzureRunAsConnection.
- An Automation certificate asset called "AzureRunAsCertificate".

.PARAMETER WebhookData
Optional. (The user doesn't need to enter anything, but the service always passes an object.)
This is the data that's sent in the webhook that's triggered from the alert.

.NOTES
AUTHOR: Azure Automation Team
LASTEDIT: 2017-11-22
#>

[OutputType("PSAzureOperationResponse")]

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

$ErrorActionPreference = "stop"

if ($WebhookData)
{
    # Get the data object from WebhookData.
    $WebhookBody = (ConvertFrom-Json -InputObject $WebhookData.RequestBody)

    # Get the info needed to identify the VM (depends on the payload schema).
    $schemaId = $WebhookBody.schemaId
    Write-Verbose "schemaId: $schemaId" -Verbose
    if ($schemaId -eq "AzureMonitorMetricAlert") {
        # This is the near-real-time Metric Alert schema
        $AlertContext = [object] ($WebhookBody.data).context
    }
}
```

```

$ResourceName = $AlertContext.resourceName
$status = ($WebhookBody.data).status
}
elseif ($schemaId -eq "Microsoft.Insights/activityLogs") {
    # This is the Activity Log Alert schema
    $AlertContext = [object] (($WebhookBody.data).context).activityLog
    $ResourceName = (($AlertContext.resourceId).Split("/"))[-1]
    $status = ($WebhookBody.data).status
}
elseif ($schemaId -eq $null) {
    # This is the original Metric Alert schema
    $AlertContext = [object] $WebhookBody.context
    $ResourceName = $AlertContext.resourceName
    $status = $WebhookBody.status
}
else {
    # The schema isn't supported.
    Write-Error "The alert data schema - $schemaId - is not supported."
}

Write-Verbose "status: $status" -Verbose
if ($status -eq "Activated")
{
    $ResourceType = $AlertContext.resourceType
    $ResourceGroupName = $AlertContext.resourceGroupName
    $SubId = $AlertContext.subscriptionId
    Write-Verbose "resourceType: $ResourceType" -Verbose
    Write-Verbose "resourceName: $ResourceName" -Verbose
    Write-Verbose "resourceGroupName: $ResourceGroupName" -Verbose
    Write-Verbose "subscriptionId: $SubId" -Verbose

    # Use this only if this is a resource management VM.
    if ($ResourceType -eq "Microsoft.Compute/virtualMachines")
    {
        # This is the VM.
        Write-Verbose "This is a resource management VM." -Verbose

        # Authenticate to Azure by using the service principal and certificate. Then, set the
        subscription.
        Write-Verbose "Authenticating to Azure with service principal and certificate" -Verbose
        $ConnectionAssetName = "AzureRunAsConnection"
        Write-Verbose "Get connection asset: $ConnectionAssetName" -Verbose
        $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-Verbose "Authenticating to Azure with service principal." -Verbose
        Connect-AzureRmAccount -ServicePrincipal -Tenant $Conn.TenantID -ApplicationId
$Conn.ApplicationID -CertificateThumbprint $Conn.CertificateThumbprint | Write-Verbose
        Write-Verbose "Setting subscription to work against: $SubId" -Verbose
        Set-AzureRmContext -SubscriptionId $SubId -ErrorAction Stop | Write-Verbose

        # Stop the VM.
        Write-Verbose "Stopping the VM - $ResourceName - in resource group - $ResourceGroupName -"
-Verbose
        Stop-AzureRmVM -Name $ResourceName -ResourceGroupName $ResourceGroupName -Force
        # [OutputType(PSAzureOperationResponse)]
    }
    else {
        # ResourceType isn't supported.
        Write-Error "$ResourceType is not a supported resource type for this runbook."
    }
}
else {
    # The alert status was not 'Activated', so no action taken.
    Write-Verbose ("No action taken. Alert status: " + $status) -Verbose
}

```

```
    }
    else {
        # Error
        Write-Error "This runbook is meant to be started from an Azure alert webhook only."
    }
}
```

7. Select **Publish** to save and publish the runbook.

Create an action group

An action group is a collection of actions that are triggered by an alert. Runbooks are just one of the many actions that you can use with action groups.

1. In the Azure portal, select **Monitor > SETTINGS > Action groups**.
2. Select **Add action group**, and then enter the required information:
 - a. In the **Action group name** box, enter a name.
 - b. In the **Short name** box, enter a name. The short name is used in place of a full action group name when notifications are sent by using this action group.
 - c. The **Subscription** box is automatically filled with your current subscription. This is the subscription in which the action group is saved.
 - d. Select the resource group in which the action group is saved.

For this example, you create two actions: a runbook action and a notification action.

Runbook action

To create a runbook action in the action group:

1. Under **Actions**, for **ACTION NAME**, enter a name for the action. For **ACTION TYPE**, select **Automation Runbook**.
2. Under **DETAILS**, select **Edit Details**.
3. On the **Configure Runbook** page, under **Runbook source**, select **User**.
4. Select your **Subscription** and **Automation account**, and then select the **Stop-AzureVmInResponseToVMAAlert** runbook.
5. When you are finished, select **OK**.

Notification action

To create a notification action in the action group:

1. Under **Actions**, for **ACTION NAME**, enter a name for the action. For **ACTION TYPE**, select **Email**.
2. Under **DETAILS** select, **Edit Details**.
3. On the **Email** page, enter the email address to use for notification, and then select **OK**. Adding an email address in addition to the runbook as an action is helpful. You are notified when the runbook is started.

Your action group should look like the following image:

Home > Monitor - Action groups > Add action group

Add action group

* Action group name ✓

* Short name ✓

* Subscription ▾

* Resource group ▾

Actions

| ACTION NAME | ACTION TYPE | STATUS | DETAILS |
|---|-------------------------------|--------|------------------------------|
| Runbook | Automation Runbo... | | Edit details |
| Email Notification | Email | | Edit details |
| <input type="text" value="Unique name for the action"/> | <input type="text" value=""/> | | |

[Privacy Statement](#)

[Pricing](#)

OK

4. To create the action group, select **OK**.

You can use this action group in the [activity log alerts](#) and [near real-time alerts](#) that you create.

Classic alert

Classic alerts are based on metrics, and do not use action groups. However, you can set up a runbook action based on a classic alert.

To create a classic alert:

1. Select **Add metric alert**.
2. Name your metric alert **myVMCPAAlert**. Enter a brief description for the alert.
3. For the metric alert condition, select **Greater than**. For the **Threshold** value, select **10**. For the **Period** value, select **Over the last five minutes**.
4. Under **Take action**, select **Run a runbook from this alert**.
5. On the **Configure Runbook** page, for **Runbook source**, select **User**. Choose your automation account, and then select the **Stop-AzureVmInResponsetoVMAalert** runbook. Select **OK**.
6. To save the alert rule, select **OK**.

Next steps

- For more information about starting an Automation runbook by using a webhook, see [Start a runbook from a webhook](#).
- For details about different ways to start a runbook, see [Starting a runbook](#).
- To learn how to create an activity log alert, see [Create activity log alerts](#).
- To learn how to create a near real-time alert, see [Create an alert rule in the Azure portal](#).

Azure Automation scenario - provision an AWS virtual machine

5/21/2018 • 4 minutes to read • [Edit Online](#)

In this article, you learn how you can leverage Azure Automation to provision a virtual machine in your Amazon Web Service (AWS) subscription and give that VM a specific name – which AWS refers to as “tagging” the VM.

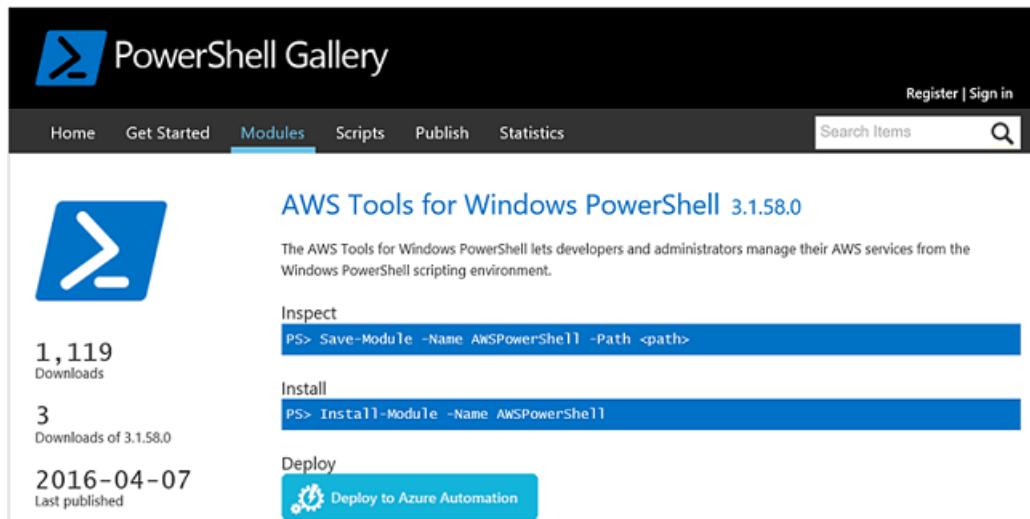
Prerequisites

For the purposes of this article, you need to have an Azure Automation account and an AWS subscription. For more information on setting up an Azure Automation account and configuring it with your AWS subscription credentials, review [Configure Authentication with Amazon Web Services](#). This account should be created or updated with your AWS subscription credentials before proceeding, as you reference this account in the steps below.

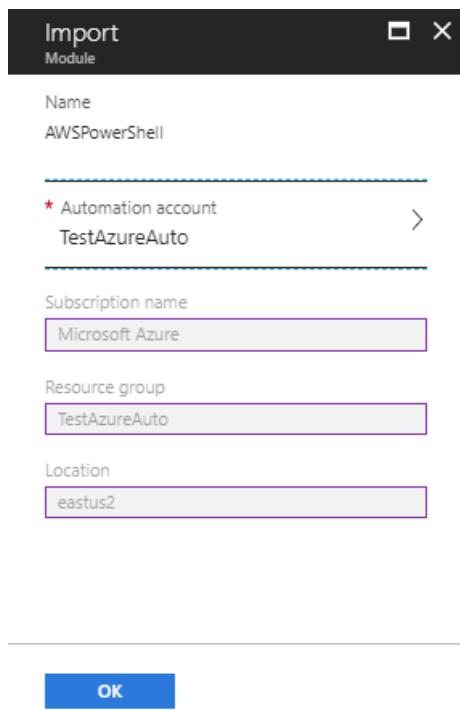
Deploy Amazon Web Services PowerShell Module

Your VM provisioning runbook leverages the AWS PowerShell module to do its work. Perform the following steps to add the module to your Automation account that is configured with your AWS subscription credentials.

1. Open your web browser and navigate to the [PowerShell Gallery](#) and click on the **Deploy to Azure Automation button**.



2. You are taken to the Azure login page and after authenticating, you will be routed to the Azure portal and presented with the following page:



3. Select the Automation Account to use and click **OK** to start deployment.

NOTE

While importing a PowerShell module into Azure Automation, it is also extracting the cmdlets and these activities do not appear until the module has completely finished importing and extracting the cmdlets. This process can take a few minutes.

4. In the Azure portal, open your Automation account referenced in step 3.
5. Click on the **Assets** tile and on the **Assets** pane, select the **Modules** tile.
6. On the **Modules** page, you see the **AWS PowerShell** module in the list.

Create AWS deploy VM runbook

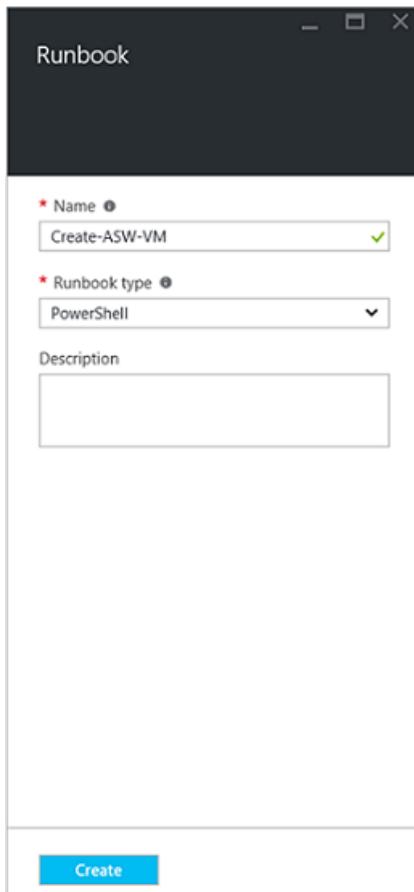
Once the AWS PowerShell Module has been deployed, you can now author a runbook to automate provisioning a virtual machine in AWS using a PowerShell script. The steps below demonstrate how to leverage native PowerShell script in Azure Automation.

NOTE

For further options and information regarding this script, please visit the [PowerShell Gallery](#).

1. Download the PowerShell script New-AwsVM from the PowerShell Gallery by opening a PowerShell session and typing the following:

```
Save-Script -Name New-AwsVM -Path <path>
```
2. From the Azure portal, open your Automation account and select **Runbooks** under the section **Process Automation** on the left.
3. From the **Runbooks** page, select **Add a runbook**.
4. On the **Add a runbook** pane, select **Quick Create** (Create a new runbook).
5. On the **Runbook** properties pane, type a name in the Name box for your runbook and from the **Runbook type** drop-down list select **PowerShell**, and then click **Create**.



- When the Edit PowerShell Runbook page appears, copy and paste the PowerShell script into the runbook authoring canvas.

```
34 #>
35 #ToDo:
36 #Change the default values for the following parameters if they do not match with yours:
37 $AWSRegion, $EC2ImageName, $MinCount, $MaxCount, $InstanceType
38 #Create an Azure Automation Asset called "AwsCred"
39 #Turn on Log verbose records and optionally log progress records under the runbook settings to see verbose
40
41 param (
42     [Parameter(Mandatory=$true)]
43     [string]$VMname,
44     [ValidateNotNullOrEmpty()])
45     [string]$AWSRegion = "us-west-2",
46     [ValidateNotNullOrEmpty()])
47     [string]$EC2ImageName = "WINDOWS_2012R2_BASE",
48     [ValidateNotNullOrEmpty()])
49     [string]$MinCount = 1,
50     [ValidateNotNullOrEmpty()])
51     [string]$MaxCount = 1,
52     [ValidateNotNullOrEmpty()])
53     [string]$InstanceType = "t2.micro"
54 )
55
56 # Get credentials to authenticate against AWS
57 $AwsCred = Get-AutomationPSCredential -Name "AwsCred"
58 $AwsAccessKeyId = $AwsCred.UserName
59 $AwsSecretKey = $AwsCred.GetNetworkCredential().Password
```

NOTE

Note the following when working with the example PowerShell script:

- The runbook contains a number of default parameter values. Evaluate all default values and update where necessary.
- If you have stored your AWS credentials as a credential asset named differently than **AWScred**, you need to update the script on line 57 to match accordingly.
- When working with the AWS CLI commands in PowerShell, especially with this example runbook, you must specify the AWS region. Otherwise, the cmdlets fail. View AWS topic [Specify AWS Region](#) in the AWS Tools for PowerShell document for further details.

7. To retrieve a list of image names from your AWS subscription, launch PowerShell ISE and import the AWS PowerShell Module. Authenticate against AWS by replacing **Get-AutomationPSCredential** in your ISE environment with **AWScred = Get-Credential**. This prompts you for your credentials and you can provide your **Access Key ID** for the username and **Secret Access Key** for the password. See the example below:

```
#Sample to get the AWS VM available images
#Please provide the path where you have downloaded the AWS PowerShell module
Import-Module AWSPowerShell
$AwsRegion = "us-west-2"
$AwsCred = Get-Credential
$AwsAccessKeyId = $AwsCred.UserName
$AwsSecretKey = $AwsCred.GetNetworkCredential().Password

# Set up the environment to access AWS
Set-AwsCredentials -AccessKey $AwsAccessKeyId -SecretKey $AwsSecretKey -StoreAs AWSProfile
Set-DefaultAWSRegion -Region $AwsRegion

Get-EC2ImageByName -ProfileName AWSProfile
```

The following output is returned:

```
PS C:\> Get-EC2ImageByName -ProfileName AWSProfile
WINDOWS_2012R2_BASE
WINDOWS_2012R2_SQL_SERVER_EXPRESS_2014
WINDOWS_2012R2_SQL_SERVER_STANDARD_2014
WINDOWS_2012R2_SQL_SERVER_WEB_2014
WINDOWS_2012_BASE
WINDOWS_2012_SQL_SERVER_EXPRESS_2014
WINDOWS_2012_SQL_SERVER_STANDARD_2014
WINDOWS_2012_SQL_SERVER_WEB_2014
WINDOWS_2012_SQL_SERVER_EXPRESS_2012
WINDOWS_2012_SQL_SERVER_STANDARD_2012
WINDOWS_2012_SQL_SERVER_WEB_2012
WINDOWS_2012_SQL_SERVER_EXPRESS_2008
WINDOWS_2012_SQL_SERVER_STANDARD_2008
WINDOWS_2012_SQL_SERVER_WEB_2008
WINDOWS_2008R2_BASE
WINDOWS_2008R2_SQL_SERVER_EXPRESS_2012
WINDOWS_2008R2_SQL_SERVER_STANDARD_2012
WINDOWS_2008R2_SQL_SERVER_WEB_2012
WINDOWS_2008R2_SQL_SERVER_EXPRESS_2008
WINDOWS_2008R2_SQL_SERVER_STANDARD_2008
WINDOWS_2008R2_SQL_SERVER_WEB_2008
WINDOWS_2008RTM_BASE
WINDOWS_2008RTM_SQL_SERVER_EXPRESS_2008
WINDOWS_2008RTM_SQL_SERVER_STANDARD_2008
WINDOWS_2008_BEANSTALK_IIS75
WINDOWS_2012_BEANSTALK_IIS8
VPC_NAT
```

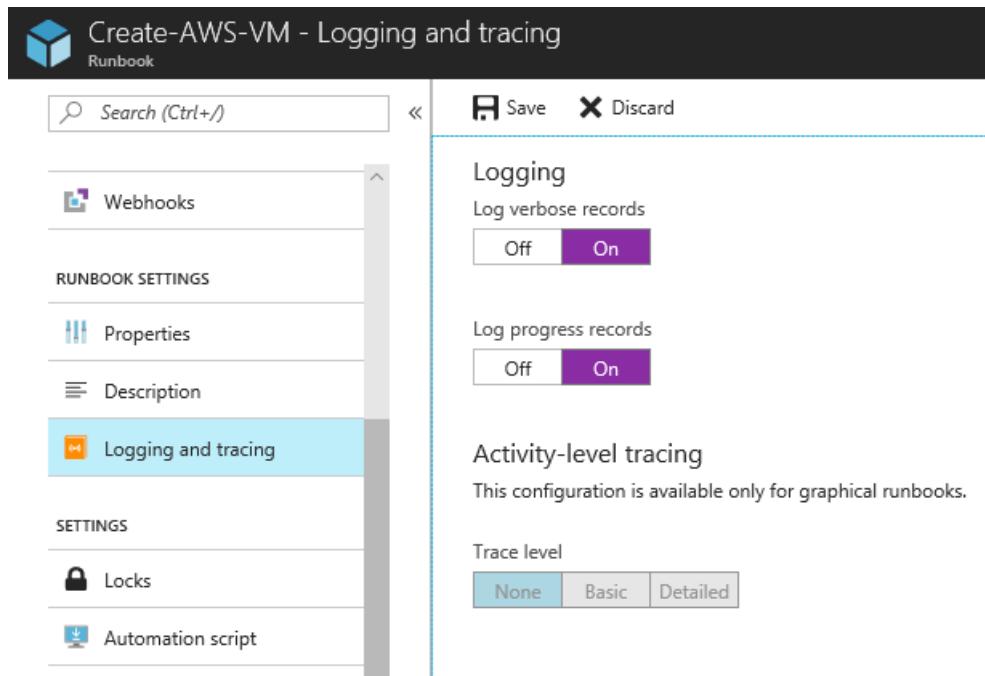
8. Copy and paste the one of the image names in an Automation variable as referenced in the runbook as **\$InstanceType**. Since in this example you are using the free AWS tiered subscription, you use **t2.micro** for your runbook example.

9. Save the runbook, then click **Publish** to publish the runbook and then **Yes** when prompted.

Testing the AWS VM runbook

Before you proceed with testing the runbook, you need to verify a few things. Specifically:

- An asset for authenticating against AWS has been created called **AWScred** or the script has been updated to reference the name of your credential asset.
- The AWS PowerShell module has been imported in Azure Automation
- A new runbook has been created and parameter values have been verified and updated where necessary
- **Log verbose records** and optionally **Log progress records** under the runbook setting **Logging and tracing** have been set to **On**.



1. You want to start the runbook, so click **Start** and then click **OK** when the Start Runbook pane opens.
2. On the Start Runbook pane, provide a **VMname**. Accept the default values for the other parameters that you preconfigured in the script earlier. Click **OK** to start the runbook job.

Start Runbook

Create-AWS-VM

Parameters

*** VMNAME** ⓘ
Enter a value
Mandatory, String

AWSREGION ⓘ
Default will be used
Optional, String, Default: "us-west-2"

EC2IMAGENAME ⓘ
Default will be used
Optional, String, Default: "WINDOWS_2012R2_BASE"

MINCOUNT ⓘ
Default will be used
Optional, String, Default: 1

MAXCOUNT ⓘ
Default will be used
Optional, String, Default: 1

INSTANCETYPE ⓘ
Default will be used
Optional, String, Default: "t2.micro"

Run Settings

Run on ⓘ
Azure Hybrid Worker

OK

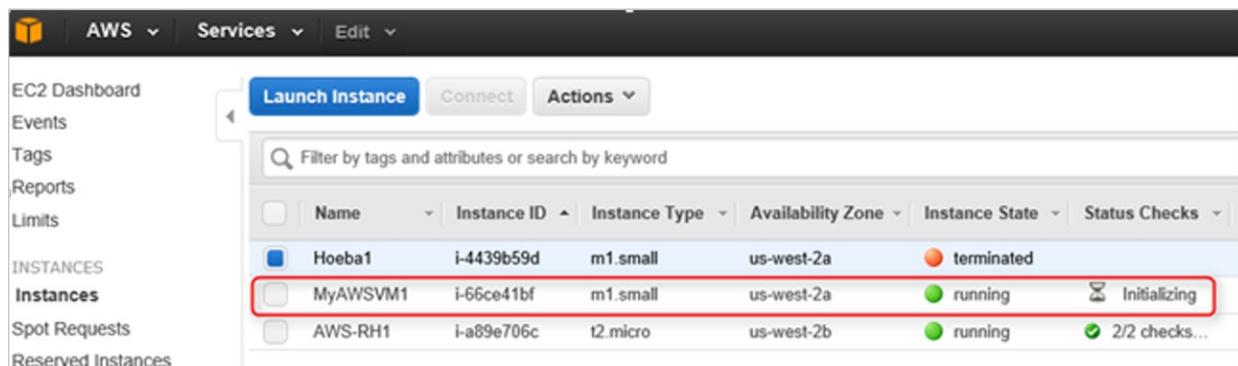
3. A job pane is opened for the runbook job that you created. Close this pane.

4. You can view progress of the job and view output **Streams** by selecting the **All Logs** tile from the runbook job page.

The screenshot shows two windows side-by-side. The left window is titled 'New-AwsVM 1/7/2016, 1:29 PM' and displays the 'Job' summary. It includes sections for 'Overview' (Job ID: caab11b8-4114-4277-80ac-c5657102354b, Created: 1/7/2016, 1:29 PM, Last update: 1/7/2016, 1:30 PM, Ran on Azure, Completed), 'Status' (Errors: 0, Warnings: 0), and 'Exception' (None). The right window is titled 'Streams' and shows the log history for the same job. The logs are listed in a table with columns for TIME, TYPE, and DETAILS. The log entries are:

| TIME | TYPE | DETAILS |
|-------------------|------------|---|
| 1/7/2016, 1:30 PM | → Progress | Preparing modules for first use. |
| 1/7/2016, 1:30 PM | Verbose | Authenticating against AWS... |
| 1/7/2016, 1:30 PM | → Progress | Preparing modules for first use. |
| 1/7/2016, 1:30 PM | Verbose | Credentials loaded from the supplied key parameters. |
| 1/7/2016, 1:30 PM | Verbose | Getting AWS Image... |
| 1/7/2016, 1:30 PM | Verbose | The following image has been found: Windows_Server-2012-R2-RTM... |
| 1/7/2016, 1:30 PM | Verbose | Creating new AWS Instance... |
| 1/7/2016, 1:30 PM | Output | |
| 1/7/2016, 1:30 PM | Verbose | Applying new VM Name... |
| 1/7/2016, 1:30 PM | Verbose | Successfully created AWS VM: MyAwsVM |

5. To confirm the VM is being provisioned, log into the AWS Management Console if you are not currently logged in.



The screenshot shows the AWS EC2 Instances page. On the left, there's a sidebar with links like EC2 Dashboard, Events, Tags, Reports, Limits, INSTANCES, Instances, Spot Requests, and Reserved Instances. The main area has tabs for Launch Instance, Connect, and Actions. A search bar says "Filter by tags and attributes or search by keyword". Below it is a table with columns: Name, Instance ID, Instance Type, Availability Zone, Instance State, and Status Checks. The table contains three rows:

| Name | Instance ID | Instance Type | Availability Zone | Instance State | Status Checks |
|----------|-------------|---------------|-------------------|----------------|---------------|
| Hoeba1 | i-4439b59d | m1.small | us-west-2a | terminated | |
| MyAWSVM1 | i-66ce41bf | m1.small | us-west-2a | running | Initializing |
| AWS-RH1 | i-a89e706c | t2.micro | us-west-2b | running | 2/2 checks... |

Next steps

- To get started with Graphical runbooks, see [My first graphical runbook](#)
- To get started with PowerShell workflow runbooks, see [My first PowerShell workflow runbook](#)
- To know more about runbook types, their advantages and limitations, see [Azure Automation runbook types](#)
- For more information on PowerShell script support feature, see [Native PowerShell script support in Azure Automation](#)

Start/Stop VMs during off-hours solution (preview) in Azure Automation

7/6/2018 • 21 minutes to read • [Edit Online](#)

The Start/Stop VMs during off-hours solution starts and stops your Azure virtual machines on user-defined schedules, provides insights through Azure Log Analytics, and sends optional emails by using [action groups](#). It supports both Azure Resource Manager and classic VMs for most scenarios.

This solution provides a decentralized automation option for users who want to reduce their costs by using serverless, low-cost resources. With this solution, you can:

- Schedule VMs to start and stop.
- Schedule VMs to start and stop in ascending order by using Azure Tags (not supported for classic VMs).
- Auto-stop VMs based on low CPU usage.

Prerequisites

- The runbooks work with an [Azure Run As account](#). The Run As account is the preferred authentication method, because it uses certificate authentication instead of a password that might expire or change frequently.
- This solution manages only VMs that are in the same subscription as your Azure Automation account.
- This solution is deployed only to the following Azure regions: Australia Southeast, Canada Central, Central India, East US, Japan East, Southeast Asia, UK South, and West Europe.

NOTE

The runbooks managing the VM schedule can target VMs in any region.

Deploy the solution

Perform the following steps to add the Start/Stop VMs during off-hours solution to your Automation account, and then configure the variables to customize the solution.

1. In the Azure portal, click **Create a resource**.
2. In the Marketplace page, type a keyword such as **Start** or **Start/Stop**. As you begin typing, the list filters based on your input. Alternatively, you can type in one or more keywords from the full name of the solution and then press Enter. Select **Start/Stop VMs during off-hours [Preview]** from the search results.
3. In the **Start/Stop VMs during off-hours [Preview]** page for the selected solution, review the summary information and then click **Create**.

Home > New > Marketplace > Everything > Start/Stop VMs during off-hours [Preview]

Start/Stop VMs during off-hours [Preview]

The Start/Stop VMs during off-hours solution starts and stops your Azure Virtual Machines on a schedule or by utilization. Save money by making sure VMs are off when not being used.

The Start/Stop VMs during off-hours solution relies on three Azure services:

- **Automation:** starts and stops your virtual machines on a schedule
- **Log Analytics:** visualizes the successful start and stop of your machines
- **Monitor:** alerts and email notification for VM state changes

You will be charged based on the pricing of the services above. Note that the logs for all runbooks jobs in the selected Automation account will be sent to Log Analytics as part of this solution.

If you use this solution, it will only use automation job minutes and log ingestion. That is, this solution does not add additional OMS nodes to your environment.

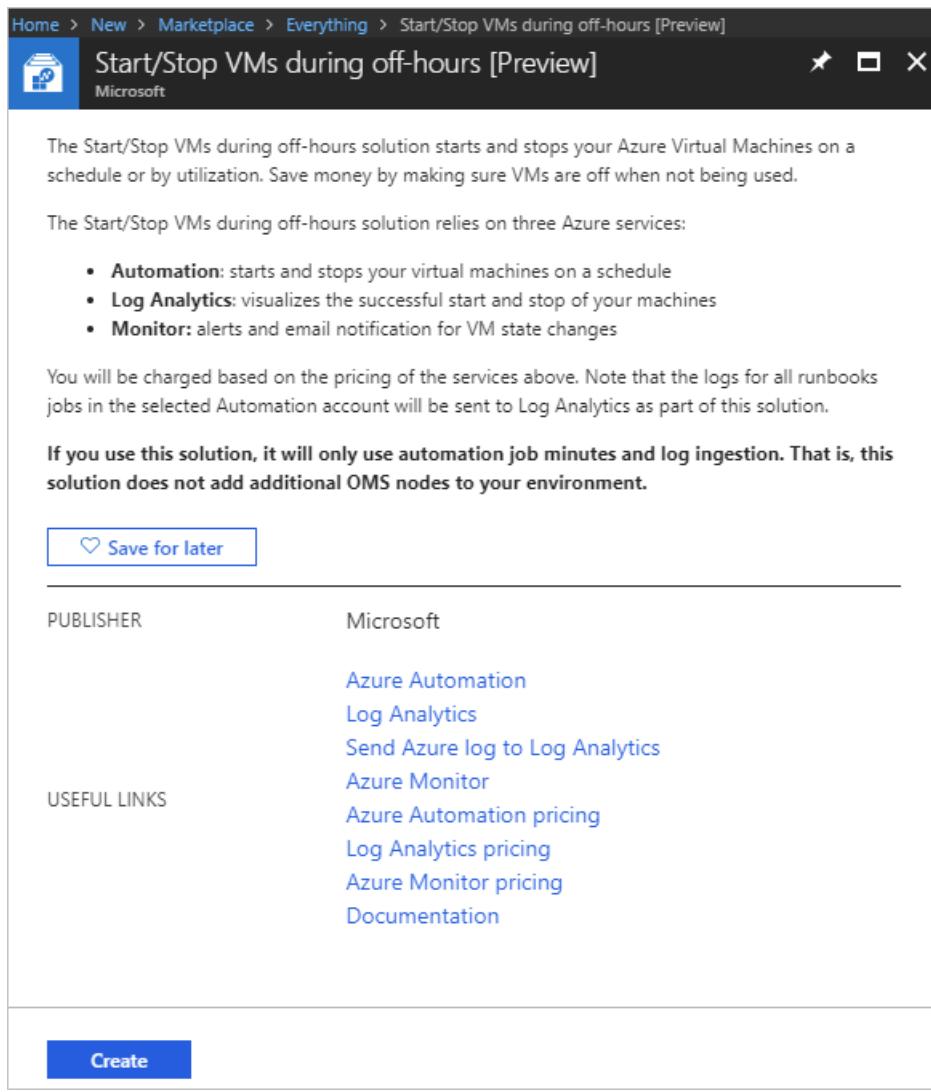
[Save for later](#)

PUBLISHER Microsoft

Azure Automation
Log Analytics
Send Azure log to Log Analytics
Azure Monitor
Azure Automation pricing
Log Analytics pricing
Azure Monitor pricing
Documentation

USEFUL LINKS

Create



4. The **Add Solution** page appears. You are prompted to configure the solution before you can import it into your Automation subscription.

Add Solution

★ Workspace
Create a new OMS workspace, or ...

★ Automation account
Create an automation account, or...

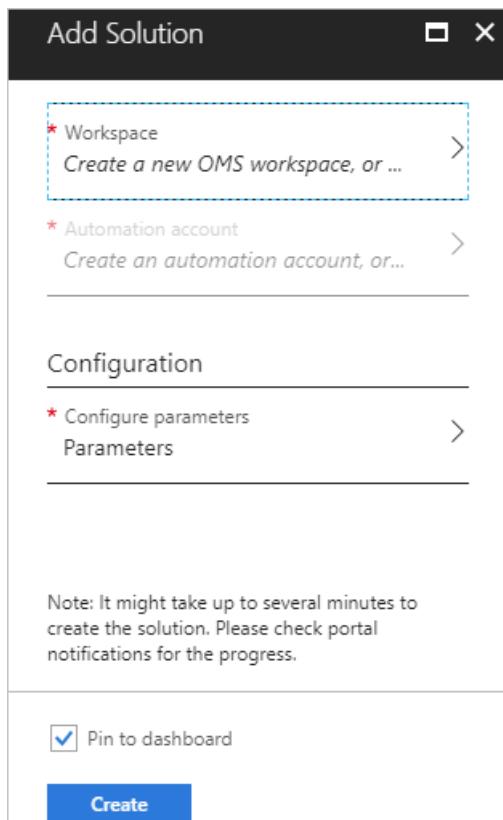
Configuration

★ Configure parameters
Parameters

Note: It might take up to several minutes to create the solution. Please check portal notifications for the progress.

Pin to dashboard

Create



5. On the **Add Solution** page, select **Workspace**. Select a Log Analytics workspace that's linked to the same Azure subscription that the Automation account is in. If you don't have a workspace, select **Create New Workspace**. On the **OMS Workspace** page, perform the following:

- Specify a name for the new **OMS Workspace**.
- Select a **Subscription** to link to by selecting from the drop-down list, if the default selected is not appropriate.
- For **Resource Group**, you can create a new resource group or select an existing one.
- Select a **Location**. Currently, the only locations available are **Australia Southeast, Canada Central, Central India, East US, Japan East, Southeast Asia, UK South, and West Europe**.
- Select a **Pricing tier**. Choose the **Per GB (Standalone)** option. Log Analytics has updated [pricing](#) and the Per GB tier is the only option.

6. After providing the required information on the **OMS workspace** page, click **Create**. You can track its progress under **Notifications** from the menu, which returns you to the **Add Solution** page when done.

7. On the **Add Solution** page, select **Automation account**. If you're creating a new Log Analytics workspace, you need to also create a new Automation account to be associated with it. Select **Create an Automation account**, and on the **Add Automation account** page, provide the following:

- In the **Name** field, enter the name of the Automation account.

All other options are automatically populated based on the Log Analytics workspace selected. These options cannot be modified. An Azure Run As account is the default authentication method for the runbooks included in this solution. After you click **OK**, the configuration options are validated and the Automation account is created. You can track its progress under **Notifications** from the menu.

8. Finally, on the **Add Solution** page, select **Configuration**. The **Parameters** page appears.

The screenshot shows the 'Parameters' configuration dialog box. It includes sections for 'Vm runbook' (with 'Target ResourceGroup Names' set to '*'), 'Schedule' (with 'Daily Start Time' set to '2018-03-22 1:00:00 PM' and 'Daily Stop Time' set to '2018-03-22 10:00:00 PM'), and 'Email functionality' (with 'Receive Email Notifications' set to 'Yes' and 'Email Addresses' set to 'admin@contoso.com'). A blue 'OK' button is at the bottom.

Here, you're prompted to:

- Specify the **Target ResourceGroup Names**. These are resource group names that contain VMs to be

managed by this solution. You can enter more than one name and separate each by using a comma (values are not case-sensitive). Using a wildcard is supported if you want to target VMs in all resource groups in the subscription. This value is stored in the **External_Start_ResourceGroupNames** and **External_Stop_ResourceGroupNames** variables.

- Specify the **VM Exclude List (string)**. This is the name of one or more virtual machines from the target resource group. You can enter more than one name and separate each by using a comma (values are not case-sensitive). Using a wildcard is supported. This value is stored in the **External_ExcludeVMNames** variable.
- Select a **Schedule**. This is a recurring date and time for starting and stopping the VMs in the target resource groups. By default, the schedule is configured for 30 minutes from now. Selecting a different region is not available. To configure the schedule to your specific time zone after configuring the solution, see [Modifying the startup and shutdown schedule](#).
- To receive **Email notifications** from an action group, accept the default value of **Yes** and provide a valid email address. If you select **No** but decide at a later date that you want to receive email notifications, you can update the [action group](#) that is created with valid email addresses separated by a comma.

IMPORTANT

The default value for **Target ResourceGroup Names** is a *. This targets all VMs in a subscription. If you do not want the solution to target all the VMs in your subscription this value needs to be updated to a list of resource group names prior to enabling the schedules.

9. After you have configured the initial settings required for the solution, click **OK** to close the **Parameters** page and select **Create**. After all settings are validated, the solution is deployed to your subscription. This process can take several seconds to finish, and you can track its progress under **Notifications** from the menu.

Scenarios

The solution contains three distinct scenarios. These scenarios are:

Scenario 1: Start/Stop VMs on a schedule

This is the default configuration when you first deploy the solution. For example, you can configure it to stop all VMs across a subscription when you leave work in the evening, and start them in the morning when you are back in the office. When you configure the schedules **Scheduled-StartVM** and **Scheduled-StopVM** during deployment, they start and stop targeted VMs. Configuring this solution to just stop VMs is supported, see [Modify the startup and shutdown schedules](#) to learn how to configure a custom schedule.

NOTE

The time zone is your current time zone when you configure the schedule time parameter. However, it is stored in UTC format in Azure Automation. You do not have to do any time zone conversion as this is handled during the deployment.

You control which VMs are in scope by configuring the following variables:

External_Start_ResourceGroupNames, **External_Stop_ResourceGroupNames**, and **External_ExcludeVMNames**.

You can enable either targeting the action against a subscription and resource group, or targeting a specific list of VMs, but not both.

Target the start and stop actions against a subscription and resource group

1. Configure the **External_Stop_ResourceGroupNames** and **External_ExcludeVMNames** variables to specify

the target VMs.

2. Enable and update the **Scheduled-StartVM** and **Scheduled-StopVM** schedules.
3. Run the **ScheduledStartStop_Parent** runbook with the ACTION parameter set to **start** and the WHATIF parameter set to **True** to preview your changes.

Target the start and stop action by VM list

1. Run the **ScheduledStartStop_Parent** runbook with the ACTION parameter set to **start**, add a comma-separated list of VMs in the **VMList** parameter, and then set the WHATIF parameter to **True**. Preview your changes.
2. Configure the **External_ExcludeVMNames** parameter with a comma-separated list of VMs (VM1,VM2,VM3).
3. This scenario does not honor the **External_Start_ResourceGroupNames** and **External_Stop_ResourceGroupNames** variables. For this scenario, you need to create your own Automation schedule. For details, see [Scheduling a runbook in Azure Automation](#).

NOTE

The value for **Target ResourceGroup Names** is stored as the value for both **External_Start_ResourceGroupNames** and **External_Stop_ResourceGroupNames**. For further granularity, you can modify each of these variables to target different resource groups. For start action, use **External_Start_ResourceGroupNames**, and for stop action, use **External_Stop_ResourceGroupNames**. VMs are automatically added to the start and stop schedules.

Scenario 2: Start/Stop VMS in sequence by using tags

In an environment that includes two or more components on multiple VMs supporting a distributed workload, supporting the sequence in which components are started and stopped in order is important. You can accomplish this by performing the following steps:

Target the start and stop actions against a subscription and resource group

1. Add a **SequenceStart** and a **SequenceStop** tag with a positive integer value to VMs that are targeted in **External_Start_ResourceGroupNames** and **External_Stop_ResourceGroupNames** variables. The start and stop actions are performed in ascending order. To learn how to tag a VM, see [Tag a Windows Virtual Machine in Azure](#) and [Tag a Linux Virtual Machine in Azure](#).
2. Modify the schedules **Sequenced-StartVM** and **Sequenced-StopVM** to the date and time that meet your requirements and enable the schedule.
3. Run the **SequencedStartStop_Parent** runbook with the ACTION parameter set to **start** and the WHATIF parameter set to **True** to preview your changes.
4. Preview the action and make any necessary changes before implementing against production VMs. When ready, manually execute the runbook with the parameter set to **False**, or let the Automation schedule **Sequenced-StartVM** and **Sequenced-StopVM** run automatically following your prescribed schedule.

Target the start and stop action by VM list

1. Add a **SequenceStart** and a **SequenceStop** tag with a positive integer value to VMs you plan to add to the **VMList** variable.
2. Run the **SequencedStartStop_Parent** runbook with the ACTION parameter set to **start**, add a comma-separated list of VMs in the **VMList** parameter, and then set the WHATIF parameter to **True**. Preview your changes.
3. Configure the **External_ExcludeVMNames** parameter with a comma-separated list of VMs (VM1,VM2,VM3).
4. This scenario does not honor the **External_Start_ResourceGroupNames** and **External_Stop_ResourceGroupNames** variables. For this scenario, you need to create your own Automation schedule. For details, see [Scheduling a runbook in Azure Automation](#).
5. Preview the action and make any necessary changes before implementing against production VMs. When ready, manually execute the monitoring-and-diagnostics/monitoring-action-groupsrunbook with the parameter set to **False**, or let the Automation schedule **Sequenced-StartVM** and **Sequenced-StopVM** run automatically

following your prescribed schedule.

Scenario 3: Start/Stop automatically based on CPU utilization

This solution can help manage the cost of running virtual machines in your subscription by evaluating Azure VMs that aren't used during non-peak periods, such as after hours, and automatically shutting them down if processor utilization is less than x%.

By default, the solution is pre-configured to evaluate the percentage CPU metric to see if average utilization is 5 percent or less. This is controlled by the following variables and can be modified if the default values do not meet your requirements:

- External_AutoStop_MetricName
- External_AutoStop_Threshold
- External_AutoStop_TimeAggregationOperator
- External_AutoStop_TimeWindow

You can enable either targeting the action against a subscription and resource group, or targeting a specific list of VMs, but not both.

Target the stop action against a subscription and resource group

1. Configure the **External_Stop_ResourceGroupNames** and **External_ExcludeVMNames** variables to specify the target VMs.
2. Enable and update the **Schedule_AutoStop_CreateAlert_Parent** schedule.
3. Run the **AutoStop_CreateAlert_Parent** runbook with the ACTION parameter set to **start** and the WHATIF parameter set to **True** to preview your changes.

Target the start and stop action by VM list

1. Run the **AutoStop_CreateAlert_Parent** runbook with the ACTION parameter set to **start**, add a comma-separated list of VMs in the **VMList** parameter, and then set the WHATIF parameter to **True**. Preview your changes.
2. Configure the **External_ExcludeVMNames** parameter with a comma-separated list of VMs (VM1,VM2,VM3).
3. This scenario does not honor the **External_Start_ResourceGroupNames** and **External_Stop_ResourceGroupNames** variables. For this scenario, you need to create your own Automation schedule. For details, see [Scheduling a runbook in Azure Automation](#).

Now that you have a schedule for stopping VMs based on CPU utilization, you need to enable one of the following schedules to start them.

- Target start action by subscription and resource group. See the steps in [Scenario 1](#) for testing and enabling **Scheduled-StartVM** schedules.
- Target start action by subscription, resource group, and tag. See the steps in [Scenario 2](#) for testing and enabling **Sequenced-StartVM** schedules.

Solution components

This solution includes preconfigured runbooks, schedules, and integration with Log Analytics so you can tailor the startup and shutdown of your virtual machines to suit your business needs.

Runbooks

The following table lists the runbooks deployed to your Automation account by this solution. You should not make changes to the runbook code. Instead, write your own runbook for new functionality.

IMPORTANT

Do not directly run any runbook with "child" appended to its name.

All parent runbooks include the *WhatIf* parameter. When set to **True**, *WhatIf* supports detailing the exact behavior the runbook takes when run without the *WhatIf* parameter and validates the correct VMs are being targeted. A runbook only performs its defined actions when the *WhatIf* parameter is set to **False**.

| RUNBOOK | PARAMETERS | DESCRIPTION |
|-----------------------------|--|--|
| AutoStop_CreateAlert_Child | VMObject
AlertAction
WebHookURI | Called from the parent runbook. This runbook creates alerts on a per-resource basis for the AutoStop scenario. |
| AutoStop_CreateAlert_Parent | VMList
<i>WhatIf</i> : True or False | Creates or updates Azure alert rules on VMs in the targeted subscription or resource groups.
VMList: Comma-separated list of VMs. For example, <i>vm1,vm2,vm3</i> .
<i>WhatIf</i> validates the runbook logic without executing. |
| AutoStop_Disable | none | Disables AutoStop alerts and default schedule. |
| AutoStop_StopVM_Child | WebHookData | Called from the parent runbook. Alert rules call this runbook to stop the VM. |
| Bootstrap_Main | none | Used one time to set up bootstrap configurations such as webhookURI, which are typically not accessible from Azure Resource Manager. This runbook is removed automatically upon successful deployment. |
| ScheduledStartStop_Child | VMName
Action: Start or Stop
ResourceGroupName | Called from the parent runbook. Executes a start or stop action for the scheduled stop. |
| ScheduledStartStop_Parent | Action: Start or Stop
VMList
<i>WhatIf</i> : True or False | This affects all VMs in the subscription. Edit the External_Start_ResourceGroupNames and External_Stop_ResourceGroupNames to only execute on these targeted resource groups. You can also exclude specific VMs by updating the External_ExcludeVMNames variable. VMList: Comma-separated list of VMs. For example, <i>vm1,vm2,vm3</i> .
<i>WhatIf</i> validates the runbook logic without executing. |

| RUNBOOK | PARAMETERS | DESCRIPTION |
|---------------------------|--|---|
| SequencedStartStop_Parent | Action: Start or Stop
WhatIf: True or False
VMList | Create tags named SequenceStart and SequenceStop on each VM for which you want to sequence start/stop activity. The value of the tag should be a positive integer (1, 2, 3) that corresponds to the order in which you want to start or stop.
VMList: Comma-separated list of VMs. For example, <i>vm1,vm2,vm3</i> .
<i>WhatIf</i> validates the runbook logic without executing.
Note: VMs must be within resource groups defined as External_Start_ResourceGroupNames, External_Stop_ResourceGroupNames, and External_ExcludeVMNames in Azure Automation variables. They must have the appropriate tags for actions to take effect. |

Variables

The following table lists the variables created in your Automation account. You should only modify variables prefixed with **External**. Modifying variables prefixed with **Internal** causes undesirable effects.

| VARIABLE | DESCRIPTION |
|---|---|
| External_AutoStop_Condition | The conditional operator required for configuring the condition before triggering an alert. Acceptable values are GreaterThan , GreaterThanOrEqual , LessThan , and LessThanOrEqual . |
| External_AutoStop_Description | The alert to stop the VM if the CPU percentage exceeds the threshold. |
| External_AutoStop_MetricName | The name of the performance metric for which the Azure Alert rule is to be configured. |
| External_AutoStop_Threshold | The threshold for the Azure Alert rule specified in the variable <i>External_AutoStop_MetricName</i> . Percentage values can range from 1 to 100. |
| External_AutoStop_TimeAggregationOperator | The time aggregation operator, which is applied to the selected window size to evaluate the condition. Acceptable values are Average , Minimum , Maximum , Total , and Last . |
| External_AutoStop_TimeWindow | The window size during which Azure analyzes selected metrics for triggering an alert. This parameter accepts input in timespan format. Possible values are from 5 minutes to 6 hours. |
| External_ExcludeVMNames | Enter VM names to be excluded, separating names by using a comma with no spaces. |
| External_Start_ResourceGroupNames | Specifies one or more resource groups, separating values by using a comma, targeted for start actions. |

| VARIABLE | DESCRIPTION |
|----------------------------------|---|
| External_Stop_ResourceGroupNames | Specifies one or more resource groups, separating values by using a comma, targeted for stop actions. |
| Internal_AutomationAccountName | Specifies the name of the Automation account. |
| Internal_AutoSnooze_WebhookUri | Specifies Webhook URI called for the AutoStop scenario. |
| Internal_AzureSubscriptionId | Specifies the Azure Subscription ID. |
| Internal_ResourceGroupName | Specifies the Automation account resource group name. |

Across all scenarios, the **External_Start_ResourceGroupNames**, **External_Stop_ResourceGroupNames**, and **External_ExcludeVMNames** variables are necessary for targeting VMs, with the exception of providing a comma-separated list of VMs for the **AutoStop_CreateAlert_Parent**, **SequencedStartStop_Parent**, and **ScheduledStartStop_Parent** runbooks. That is, your VMs must reside in target resource groups for start and stop actions to occur. The logic works similar to Azure policy, in that you can target the subscription or resource group and have actions inherited by newly created VMs. This approach avoids having to maintain a separate schedule for every VM and manage starts and stops in scale.

Schedules

The following table lists each of the default schedules created in your Automation account. You can modify them or create your own custom schedules. By default, each of these are disabled except for **Scheduled_StartVM** and **Scheduled_StopVM**.

You should not enable all schedules, because this might create overlapping schedule actions. It's best to determine which optimizations you want to perform and modify accordingly. See the example scenarios in the overview section for further explanation.

| SCHEDULE NAME | FREQUENCY | DESCRIPTION |
|--------------------------------------|---------------------|--|
| Schedule_AutoStop_CreateAlert_Parent | Every 8 hours | Runs the AutoStop_CreateAlert_Parent runbook every 8 hours, which in turn stops the VM-based values in External_Start_ResourceGroupNames, External_Stop_ResourceGroupNames, and External_ExcludeVMNames in Azure Automation variables. Alternatively, you can specify a comma-separated list of VMs by using the VMList parameter. |
| Scheduled_StopVM | User defined, daily | Runs the Scheduled_Parent runbook with a parameter of <i>Stop</i> every day at the specified time. Automatically stops all VMs that meet the rules defined by asset variables. You should enable the related schedule, Scheduled-StartVM . |
| Scheduled_StartVM | User defined, daily | Runs the Scheduled_Parent runbook with a parameter of <i>Start</i> every day at the specified time. Automatically starts all VMs that meet the rules defined by the appropriate variables. You should enable the related schedule, Scheduled-StopVM . |

| SCHEDULE NAME | FREQUENCY | DESCRIPTION |
|-------------------|-----------------------------|--|
| Sequenced-StopVM | 1:00 AM (UTC), every Friday | Runs the Sequenced_Parent runbook with a parameter of <i>Stop</i> every Friday at the specified time. Sequentially (ascending) stops all VMs with a tag of SequenceStop defined by the appropriate variables. Refer to the Runbooks section for more details on tag values and asset variables. You should enable the related schedule, Sequenced-StartVM . |
| Sequenced-StartVM | 1:00 PM (UTC), every Monday | Runs the Sequenced_Parent runbook with a parameter of <i>Start</i> every Monday at the specified time. Sequentially (descending) starts all VMs with a tag of SequenceStart defined by the appropriate variables. Refer to the Runbooks section for more details on tag values and asset variables. You should enable the related schedule, Sequenced-StopVM . |

Log Analytics records

Automation creates two types of records in the Log Analytics workspace: job logs and job streams.

Job logs

| PROPERTY | DESCRIPTION |
|------------------|---|
| Caller | Who initiated the operation. Possible values are either an email address or system for scheduled jobs. |
| Category | Classification of the type of data. For Automation, the value is JobLogs. |
| CorrelationId | GUID that is the Correlation ID of the runbook job. |
| JobId | GUID that is the ID of the runbook job. |
| operationName | Specifies the type of operation performed in Azure. For Automation, the value is Job. |
| resourceId | Specifies the resource type in Azure. For Automation, the value is the Automation account associated with the runbook. |
| ResourceGroup | Specifies the resource group name of the runbook job. |
| ResourceProvider | Specifies the Azure service that supplies the resources you can deploy and manage. For Automation, the value is Azure Automation. |
| ResourceType | Specifies the resource type in Azure. For Automation, the value is the Automation account associated with the runbook. |

| PROPERTY | DESCRIPTION |
|-------------------|---|
| resultType | The status of the runbook job. Possible values are:
- Started
- Stopped
- Suspended
- Failed
- Succeeded |
| resultDescription | Describes the runbook job result state. Possible values are:
- Job is started
- Job Failed
- Job Completed |
| RunbookName | Specifies the name of the runbook. |
| SourceSystem | Specifies the source system for the data submitted. For Automation, the value is OpsManager |
| StreamType | Specifies the type of event. Possible values are:
- Verbose
- Output
- Error
- Warning |
| SubscriptionId | Specifies the subscription ID of the job. |
| Time | Date and time when the runbook job executed. |

Job streams

| PROPERTY | DESCRIPTION |
|------------------|---|
| Caller | Who initiated the operation. Possible values are either an email address or system for scheduled jobs. |
| Category | Classification of the type of data. For Automation, the value is JobStreams. |
| JobId | GUID that is the ID of the runbook job. |
| operationName | Specifies the type of operation performed in Azure. For Automation, the value is Job. |
| ResourceGroup | Specifies the resource group name of the runbook job. |
| resourceId | Specifies the resource ID in Azure. For Automation, the value is the Automation account associated with the runbook. |
| ResourceProvider | Specifies the Azure service that supplies the resources you can deploy and manage. For Automation, the value is Azure Automation. |
| ResourceType | Specifies the resource type in Azure. For Automation, the value is the Automation account associated with the runbook. |

| PROPERTY | DESCRIPTION |
|-------------------|--|
| resultType | The result of the runbook job at the time the event was generated. A possible value is:
- InProgress |
| resultDescription | Includes the output stream from the runbook. |
| RunbookName | The name of the runbook. |
| SourceSystem | Specifies the source system for the data submitted. For Automation, the value is OpsManager. |
| StreamType | The type of job stream. Possible values are:
- Progress
- Output
- Warning
- Error
- Debug
- Verbose |
| Time | Date and time when the runbook job executed. |

When you perform any log search that returns category records of **JobLogs** or **JobStreams**, you can select the **JobLogs** or **JobStreams** view, which displays a set of tiles summarizing the updates returned by the search.

Sample log searches

The following table provides sample log searches for job records collected by this solution.

| QUERY | DESCRIPTION |
|---|--|
| Find jobs for runbook ScheduledStartStop_Parent that have finished successfully | search Category == "JobLogs" where (RunbookName_s == "ScheduledStartStop_Parent") where (ResultType == "Completed") summarize AggregatedValue = count() by ResultType, bin(TimeGenerated, 1h) sort by TimeGenerated desc |
| Find jobs for runbook SequencedStartStop_Parent that have finished successfully | search Category == "JobLogs" where (RunbookName_s == "SequencedStartStop_Parent") where (ResultType == "Completed") summarize AggregatedValue = count() by ResultType, bin(TimeGenerated, 1h) sort by TimeGenerated desc |

Viewing the solution

To access the solution, navigate to your Automation Account, select **Workspace** under **RELATED RESOURCES**. On the Log Analytics page, select **Solutions** under **GENERAL**. On the **Solutions** page, select the solution **Start-Stop-VM[workspace]** from the list.

Selecting the solution displays the **Start-Stop-VM[workspace]** solution page. Here you can review important details such as the **StartStopVM** tile. As in your Log Analytics workspace, this tile displays a count and a graphical representation of the runbook jobs for the solution that have started and have finished successfully.

The screenshot shows the OMS Portal interface for the 'Start-Stop-VM[Workspace-e7e3582f]' solution. The left sidebar contains navigation links like Overview, Activity log, Access control (IAM), Diagnose and solve problems, Locks, Automation script, OMS Workspace, Properties, and Saved searches. The main content area has tabs for Essentials and Summary. Under Essentials, it lists the Resource group (testazureauto), Status (Active), Location (East US), Subscription name (Microsoft Azure), and Management services (Operations logs). Under Summary, there's a donut chart titled 'StartStopVMView' showing 60 total items with 30 Started and 30 Completed. To the right, there's a section for Solution Resources with one item named 'StartStopVMView'.

From here, you can perform further analysis of the job records by clicking the donut tile. The solution dashboard shows job history and pre-defined log search queries. Switch to the Log Analytics Advanced portal to search based on your search queries.

Configure email notifications

To change email notifications after the solution is deployed, modify action group that was created during deployment.

In the Azure portal, navigate to Monitor -> Action groups. Select the action group titled

StartStop_VM_Notification.

The screenshot shows the Azure portal's Monitor - Action groups page. The left sidebar includes options like Create a resource, All services, Favorites (Virtual machines, Load balancers, Storage accounts, Virtual networks, Azure Active Directory), and Monitor (which is selected and highlighted with a red box). The main content area shows a list of action groups. One action group is listed: 'StartStop_VM_Notification' (Short name: StStAlert, Subscription: Microsoft Azure, Resource group: ExampleAutoAccount, Status: Enabled, Actions: 1 Email).

On the **StartStop_VM_Notification** page, click **Edit details** under **Details**. This opens the **Email/SMS/Push/Voice** page. Update the email address and click **OK** to save your changes.

Email/SMS/Push/Voice

Name
Emailnt3m3rhm56gq4

Email
xlkjciekc321c3@contoso.com ✓

SMS

Country code * Phone number
1 1234567890

i Carrier charges may apply.

Azure app Push Notifications
Learn about the connecting to your Azure resources using the Azure app.
email@example.com

This is the email you use to log into your Azure account.

Voice

Country code * Phone number
1 1234567890

OK

Alternatively you can add additional actions to the action group, to learn more about action groups, see [action groups](#)

The following is an example email that is sent when the solution shuts down virtual machines.

File Message Help Tell me what you want to do

Mon 6/11/2018 2:25 PM

Azure Email Service No Reply Account

Start/Stop VMs during off-hours Runbook: ScheduledStartStop_Parent has attempted an action

To Administrator

If there are problems with how this message is displayed, click here to view it in a web browser.

We are notifying you because there are 1 counts of "ScheduledStartStop_Parent"

| | |
|----------------------------|--|
| NAME | ScheduledStartStop_Parent |
| SEVERITY | Informational |
| RESOURCE | StartStopWorkspaceExample |
| SEARCH INTERVAL START TIME | 6/11/2018 9:14:24 PM (UTC) |
| SEARCH INTERVAL DURATION | 5 min |
| SEARCH QUERY | AzureDiagnostics where (RunbookName_s == "ScheduledStartStop_Parent") where (ResultDescription hasprefix "~") extend output = substring(ResultDescription,1) summarize by ResultDescription, output project output |
| SEARCH RESULTS | 1 result(s) |
| DESCRIPTION | Start/Stop VMs during off-hours Runbook: ScheduledStartStop_Parent has attempted an action |

Top 1 result(s)

Attempted the stop action on the following VMs: aks-agentpool-42602657-0 aks-agentpool-42602657-1 aks-agentpool-42602657-2 LinuxVM1 LinuxVM2 LinuxVM3 WinVM1 WinVM2 WinVM3 WinVM4 WinVM5 LinuxVM4 LinuxVM5 LinuxVM6 WinVM6

Modify the startup and shutdown schedules

Managing the startup and shutdown schedules in this solution follows the same steps as outlined in [Scheduling a runbook in Azure Automation](#).

Configuring the solution to just stop VMs at a certain time is supported. To do this, you need to:

1. Ensure you have added the resource groups for the VMs to shut down in the **External_Start_ResourceGroupNames** variable.
2. Create your own schedule for the time you want to shut down the VMs.
3. Navigate to the **ScheduledStartStop_Parent** runbook and click **Schedule**. This allows you to select the schedule you created in the preceding step.
4. Select **Parameters and run settings** and set the ACTION parameter to "Stop".
5. Click **OK** to save your changes.

Update the solution

If you have deployed a previous version of this solution, you must first delete it from your account before deploying an updated release. Follow the steps to [remove the solution](#) and then follow the steps above to [deploy the solution](#).

Remove the solution

If you decide you no longer need to use the solution, you can delete it from the Automation account. Deleting the solution only removes the runbooks. It does not delete the schedules or variables that were created when the solution was added. Those assets you need to delete manually if you are not using them with other runbooks.

To delete the solution, perform the following steps:

1. From your Automation account, select **Workspace** from the left page.
2. On the **Solutions** page, select the solution **Start-Stop-VM[Workspace]**. On the **VMMManagementSolution[Workspace]** page, from the menu, select **Delete**.

The screenshot shows the Azure portal's 'Solutions' section. A specific solution named 'VMMManagementSolution[Workspace]' is selected. The page provides a summary of the solution's configuration, such as the resource group ('testazureauto'), status ('Active'), location ('East US'), and subscription ('Microsoft Azure'). The 'Delete' button is prominently displayed at the top right of the solution's card.

3. In the **Delete Solution** window, confirm that you want to delete the solution.
4. While the information is verified and the solution is deleted, you can track its progress under **Notifications** from the menu. You are returned to the **Solutions** page after the process to remove the solution starts.

The Automation account and Log Analytics workspace are not deleted as part of this process. If you do not want to retain the Log Analytics workspace, you need to manually delete it. This can be accomplished from the Azure portal:

1. From the Azure portal home screen, select **Log Analytics**.
2. On the **Log Analytics** page, select the workspace.
3. Select **Delete** from the menu on the workspace settings page.

Next steps

- To learn more about how to construct different search queries and review the Automation job logs with Log Analytics, see [Log searches in Log Analytics](#).
- To learn more about runbook execution, how to monitor runbook jobs, and other technical details, see [Track a runbook job](#).
- To learn more about Log Analytics and data collection sources, see [Collecting Azure storage data in Log Analytics overview](#).

Azure Automation scenario - automate removal of resource groups

5/21/2018 • 2 minutes to read • [Edit Online](#)

Many customers create more than one resource group. Some might be used for managing production applications, and others might be used as development, testing, and staging environments. Automating the deployment of these resources is one thing, but being able to decommission a resource group with a click of the button is another. You can streamline this common management task by using Azure Automation. This is helpful if you are working with an Azure subscription that has a spending limit through a member offer like MSDN or the Microsoft Partner Network Cloud Essentials program.

This scenario is based on a PowerShell runbook and is designed to remove one or more resource groups that you specify from your subscription. The default setting of the runbook is to test before proceeding. This ensures that you don't accidentally delete the resource group before you're ready to complete this procedure.

Getting the scenario

This scenario consists of a PowerShell runbook that you can download from the [PowerShell Gallery](#). You can also import it directly from the [Runbook Gallery](#) in the Azure portal.

| RUNBOOK | DESCRIPTION |
|----------------------|---|
| Remove-ResourceGroup | Removes one or more Azure resource groups and associated resources from the subscription. |

The following input parameters are defined for this runbook:

| PARAMETER | DESCRIPTION |
|------------------------|---|
| NameFilter (Required) | Specifies a name filter to limit the resource groups that you intend on deleting. You can pass multiple values using a comma-separated list.
The filter is not case-sensitive and matches any resource group that contains the string. |
| PreviewMode (Optional) | Executes the runbook to see which resource groups would be deleted, but takes no action.
The default is true to help avoid accidental deletion of one or more resource groups passed to the runbook. |

Install and configure this scenario

Prerequisites

This runbook authenticates using the [Azure Run As account](#).

Install and publish the runbooks

After you download the runbook, you can import it by using the procedure in [Importing runbook procedures](#). Publish the runbook after it has been successfully imported into your Automation account.

Using the runbook

The following steps walk you through the execution of this runbook and help you become familiar with how it works. You are testing the runbook in this example, not actually deleting the resource group.

1. From the Azure portal, open your Automation account and click **Runbooks**.
2. Select the **Remove-ResourceGroup** runbook and click **Start**.
3. When you start the runbook, the **Start Runbook** page opens and you can configure the parameters. Enter the names of resource groups in your subscription that you can use for testing and causes no harm if accidentally deleted.

NOTE

Make sure **Previewmode** is set to **true** to avoid deleting the selected resource groups. This runbook does not remove the resource group that contains the Automation account that is running this runbook.

4. After you have configured all the parameter values, click **OK**, and the runbook will be queued for execution.

To view the details of the **Remove-ResourceGroup** runbook job in the Azure portal, under **Resource**, select **Jobs** in the runbook. Select the job you want to view. The job summary displays the input parameters and the output stream in addition to general information about the job and any exceptions that occurred.



Remove-ResourceGroup 2/2/2018, 1:06 PM

Job

▶ Resume ■ Stop || Suspend

Essentials ^

| | |
|--------------------------------------|--------------------------------------|
| Job Id | Created |
| 6bb8e9f3-6f7d-4567-a2d4-dbaa68111241 | 2/2/2018, 1:06 PM |
| Job status | Last Update |
| Completed | 2/2/2018, 1:07 PM |
| Run As | Runbook |
| User | Remove-ResourceGroup |
| Ran on | Source snapshot |
| Azure | View source snapshot |

Overview

Input

1 ➔

Output

➔ Output



All Logs

Errors

0 ✘

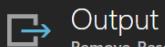
Warnings

0 !

Exception

None

The **Job Summary** includes messages from the output, warning, and error streams. Select **Output** to view detailed results from the runbook execution.



Output

Remove-ResourceGroup 2/2/2018, 1:06 PM

```
The resource group for this runbook job will not be removed. Resource group: TestAzureAuto
```

```
Preview Mode: The following resource groups would be removed:
```

```
ContosoVM
```

```
Preview Mode: The following resources would be removed:
```

```
Name : ContosoVM1  
ResourceId : /subscriptions/  
            pute/virtualMachines/ContosoVM1  
/resourceGroups/ContosoVM/providers/Microsoft.  
ResourceName : ContosoVM1  
ResourceType : Microsoft.Compute/virtualMachines  
ResourceGroupName : ContosoVM  
Location : eastus  
SubscriptionId :  
Tags : {Schedule}  
  
Name : ContosoVM1/Microsoft.Powershell.DSC  
ResourceId : /subscriptions/  
            pute/virtualMachines/ContosoVM1/extensions/Microsoft.Powershell.DSC  
/resourceGroups/ContosoVM/providers/Microsoft.  
ResourceName : ContosoVM1/Microsoft.Powershell.DSC  
ResourceType : Microsoft.Compute/virtualMachines/extensions  
ResourceGroupName : ContosoVM  
Location : eastus  
SubscriptionId :  
Tags : {AutomationAccountARMINID}
```

Next steps

- To get started creating your own runbook, see [Creating or importing a runbook in Azure Automation](#).
- To get started with PowerShell Workflow runbooks, see [My first PowerShell Workflow runbook](#).

Azure Automation scenario - Automation source control integration with GitHub Enterprise

6/12/2018 • 4 minutes to read • [Edit Online](#)

Automation currently supports source control integration, which allows you to associate runbooks in your Automation account to a GitHub source control repository. However, customers who have deployed [GitHub Enterprise](#) to support their DevOps practices, also want to use it to manage the lifecycle of runbooks that are developed to automate business processes and service management operations.

In this scenario, you have a Windows computer in your data center configured as a Hybrid Runbook Worker with the Azure Resource Manager modules and Git tools installed. The Hybrid worker machine has a clone of the local Git repository. When the runbook is run on the hybrid worker, the Git directory is synchronized and the runbook file contents are imported into the Automation account.

This article describes how to set up this configuration in your Azure Automation environment. You start by configuring Automation with the security credentials, runbooks required to support this scenario, and deployment of a Hybrid Runbook Worker in your data center to run the runbooks and access your GitHub Enterprise repository to synchronize runbooks with your Automation account.

Getting the scenario

This scenario consists of two PowerShell runbooks that you can import directly from the [Runbook Gallery](#) in the Azure portal or download from the [PowerShell Gallery](#).

Runbooks

| RUNBOOK | DESCRIPTION |
|--|---|
| Export-RunAsCertificateToHybridWorker | Runbook exports a RunAs certificate from an Automation account to a hybrid worker so that runbooks on the worker can authenticate with Azure in order to import runbooks into the Automation account. |
| Sync-LocalGitFolderToAutomationAccount | Runbook syncs the local Git folder on the hybrid machine and then import the runbook files (*.ps1) into the Automation account. |

Credentials

| CREDENTIAL | DESCRIPTION |
|------------------|--|
| GitHRWCredential | Credential asset you create to contain the username and password for a user with permissions to the hybrid worker. |

Installing and configuring this scenario

Prerequisites

1. The Sync-LocalGitFolderToAutomationAccount runbook authenticates using the [Azure Run As account](#).
2. A Log Analytics workspace with the Azure Automation solution enabled and configured is also required. If you do not have one that is associated with the Automation account used to install and configure this

scenario, it is created and configured for you when you execute the **New-OnPremiseHybridWorker.ps1** script from the hybrid runbook worker.

NOTE

Currently the following regions only support Automation integration with Log Analytics - **Australia Southeast, East US 2, Southeast Asia, and West Europe**.

3. A computer that can serve as a dedicated Hybrid Runbook Worker that also hosts the GitHub software and maintain the runbook files (*runbook.ps1*) in a source directory on the file system to synchronize between GitHub and your Automation account.

Import and publish the runbooks

To import the *Export-RunAsCertificateToHybridWorker* and *Sync-LocalGitFolderToAutomationAccount* runbooks from the Runbook Gallery from your Automation account in the Azure portal, follow the procedures in [Import Runbook from the Runbook Gallery](#). Publish the runbooks after they have been successfully imported into your Automation account.

Deploy and Configure Hybrid Runbook Worker

If you do not have a Hybrid Runbook Worker already deployed in your data center, you should review the requirements and follow the automated installation steps using the procedure in [Azure Automation Hybrid Runbook Workers - Automate Install and Configuration for Windows or Linux](#). Once you have successfully installed the hybrid worker on a computer, perform the following steps to complete its configuration to support this scenario.

1. Sign on to the computer hosting the Hybrid Runbook Worker role with an account that has local administrative rights and create a directory to hold the Git runbook files. Clone the internal Git repository to the directory.
2. If you do not already have a RunAs account created or you want to create a new one dedicated for this purpose, from the Azure portal navigate to Automation accounts, select your Automation account, and create a [credential asset](#) that contains the username and password for a user with permissions to the hybrid worker.
3. From your Automation account, [edit the runbook Export-RunAsCertificateToHybridWorker](#) and modify the value for the variable *\$Password* with a strong password. After you modify the value, click **Publish** to have the draft version of the runbook published.
4. Start the runbook **Export-RunAsCertificateToHybridWorker**, and in the **Start Runbook** blade, under the option **Run settings** select the option **Hybrid Worker** and in the drop-down list select the Hybrid worker group you created earlier for this scenario.

This exports a certificate to the hybrid worker so that runbooks on the worker can authenticate with Azure using the Run As connection in order to manage Azure resources (in particular for this scenario - import runbooks to the Automation account).

5. From your Automation account, select the Hybrid worker group created earlier and [specify a RunAs account](#) for the Hybrid worker group, and chose the credential asset you just or already have created. This assures that the Sync runbook can run Git commands.
6. Start the runbook **Sync-LocalGitFolderToAutomationAccount**, provide the following required input parameter values and in the **Start Runbook** blade, under the option **Run settings** select the option **Hybrid Worker** and in the drop-down list select the Hybrid worker group you created earlier for this scenario:
 - *ResourceGroup* - the name of your resource group associated with your Automation account
 - *AutomationAccountName* - the name of your Automation account
 - *GitPath* - The local folder or file on the Hybrid Runbook Worker where Git is set up to pull latest changes into

This syncs the local Git folder on the hybrid worker computer and then imports the .ps1 files from the source directory to the Automation account.

7. View job summary details for the runbook by selecting it from the **Runbooks** blade in your Automation account, and then select the **Jobs** tile. Confirm it completed successfully by selecting the **All logs** tile and reviewing the detailed log stream.

Next steps

- To know more about runbook types, their advantages and limitations, see [Azure Automation runbook types](#)
- For more information on PowerShell script support feature, see [Native PowerShell script support in Azure Automation](#)

Azure Automation scenario - Automation source control integration with Visual Studio Team Services

5/21/2018 • 3 minutes to read • [Edit Online](#)

In this scenario, you have a Visual Studio Team Services project that you are using to manage Azure Automation runbooks or DSC configurations under source control. This article describes how to integrate VSTS with your Azure Automation environment so that continuous integration happens for each check-in.

Getting the scenario

This scenario consists of two PowerShell runbooks that you can import directly from the [Runbook Gallery](#) in the Azure portal or download from the [PowerShell Gallery](#).

Runbooks

| RUNBOOK | DESCRIPTION |
|--------------|---|
| Sync-VSTS | Import runbooks or configurations from VSTS source control when a check-in is done. If run manually, it imports and publishes all runbooks or configurations into the Automation account. |
| Sync-VSTSGit | Import runbooks or configurations from VSTS under Git source control when a check-in is done. If run manually, it imports and publishes all runbooks or configurations into the Automation account. |

Variables

| VARIABLE | DESCRIPTION |
|----------|---|
| VSToken | Secure variable asset you create that contains the VSTS personal access token. You can learn how to create a VSTS personal access token on the VSTS authentication page . |

Installing and configuring this scenario

Create a [personal access token](#) in VSTS that you use to sync the runbooks or configurations into your automation account.

The screenshot shows the 'Personal access tokens' section of the VSTS settings. On the left, there's a sidebar with 'Security', 'Notifications', and 'Usage'. Under 'Personal access tokens', it says: 'Personal access tokens can be used instead of a password to allow applications outside the browser access to the resources stored in your account.' Below that, it says: 'Your tokens are like passwords. Keep them secret.' There are three buttons at the bottom: 'Add', 'Revoke All', and 'Show Revoked Tokens'. A table lists the tokens:

| Description | Expiration | Status ↑ | Actions |
|-------------|---------------------|----------|---------|
| ContosoDev | 5/6/2018 3:10:03 PM | Active | X |

Create a [secure variable](#) in your automation account to hold the personal access token so that the runbook can authenticate to VSTS and sync the runbooks or configurations into the Automation account. You can name this

VSToken.

The screenshot shows the 'Variables' blade in an Azure Automation account named 'TestAzureAuto'. On the left, there's a sidebar with 'RELATED RESOURCES' including 'Workspace', 'Unlink workspace', and 'Event grid'. The main area displays a table with one row:

| NAME | TYPE | VALUE | LAST MODIFIED |
|---------|---------------------|-------|------------------|
| VSToken | Unknown (encryp...) | ***** | 2/5/2018 7:11 AM |

Import the runbook that syncs your runbooks or configurations into the automation account. You can use the [VSTS sample runbook](#) or the [VSTS with Git sample runbook](#) from the PowerShellGallery.com depending on if you use VSTS source control or VSTS with Git and deploy to your automation account.

The screenshot shows the PowerShell Gallery item page for 'Sync-VSTS'. The page includes the following details:

- Icon:** A gear with a lightning bolt.
- Title:** Integrate Azure Automation with Visual Studio Team Services Source Control 1.0
- Downloads:** 0 Downloads, 0 Downloads of 1.0
- Last published:** 2017-01-23
- Project Site:** [Project Site](#)
- License:** [License](#)
- Contact Owners:** [Contact Owners](#)
- Report Abuse:** [Report Abuse](#)
- How to Download:** [How to Download](#)
- Script Statistics:** [Script Statistics](#)
- Share this item:** Buttons for Facebook (0), LinkedIn (0), and Twitter (0).
- Inspect:** PS> Save-Script -Name Sync-VSTS -Path <path>
- Install:** PS> Install-Script -Name Sync-VSTS
- Deploy:** A button labeled 'Deploy to Azure Automation' with an icon of a gear and a lightning bolt.
- See Documentation for more details.**
- Owners:** AzureAutomation

You can now [publish](#) this runbook so you can create a webhook.

The screenshot shows the Azure Automation Runbook Editor interface. The title bar says "Edit PowerShell Runbook" and "Sync-VSTS". The left sidebar has icons for CMDLETS, RUNBOOKS, and ASSETS. The main area contains the following PowerShell script:

```

1 <#
2 .SYNOPSIS
3     This Azure Automation runbook syncs runbook and configurations from VSTS source control. It requires that a
4     service hook be set up in VSTS to trigger this runbook when changes are made.
5
6 .DESCRIPTION
7     This Azure Automation runbook syncs runbook and configurations from VSTS source control. It requires that a
8     service hook be set up in VSTS to trigger this runbook when changes are made. It can also be run without a
9     service hook to force a sync of everything from VSTS folder.
10    It requires that you have the RunAs account configured in the automation service.
11
12    This enables continuous integration with VSTS source control and an automation account.
13
14 .PARAMETER WebhookData
15     Optional. This will contain the change that was made in VSTS and sent over to the runbook through a
16     service hook call.
17
18 .PARAMETER ResourceGroup
19     Required. The name of the resource group the automation account is in.
20
21 .PARAMETER AutomationAccountName
22     Required. The name of the Automation account to sync all the runbooks and configurations to
23
24 .PARAMETER VSFolder
25     Required. The name of the folder in VSTS where the runbooks and configurations exist.
26     This should look like '$/ContosoDev/AutomationScriptsConfigurations'

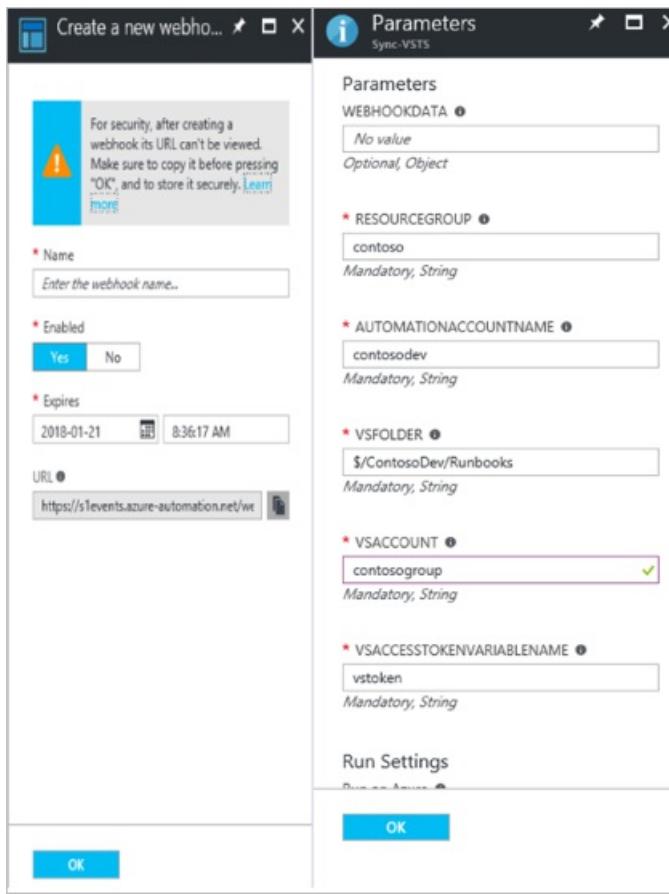
```

Create a webhook for this Sync-VSTS runbook and fill in the parameters as shown below. Make sure you copy the webhook url as you need it for a service hook in VSTS. The VSAccessTokenVariableName is the name (VSToken) of the secure variable that you created earlier to hold the personal access token.

Integrating with VSTS (Sync-VSTS.ps1) takes the following parameters:

Sync-VSTS Parameters

| PARAMETER | DESCRIPTION |
|---------------------------|---|
| WebhookData | This contains the checkin information sent from the VSTS service hook. You should leave this parameter blank. |
| ResourceGroup | This is the name of the resource group that the automation account is in. |
| AutomationAccountName | The name of the automation account that syncs with VSTS. |
| VSFolder | The name of the folder in VSTS where the runbooks and configurations exist. |
| VSAccount | The name of the Visual Studio Team Services account. |
| VSAccessTokenVariableName | The name of the secure variable (VSToken) that holds the VSTS personal access token. |



If you are using VSTS with GIT (Sync-VSTSGit.ps1) it will take the following parameters.

| PARAMETER | DESCRIPTION |
|---------------------------|---|
| WebhookData | This will contain the checkin information sent from the VSTS service hook. You should leave this parameter blank. |
| AutomationAccountName | The name of the automation account that syncs with VSTS. |
| VSAccount | The name of the Visual Studio Team Services account. |
| VSProject | The name of the project in VSTS where the runbooks and configurations exist. |
| GitRepo | The name of the Git repository. |
| GitBranch | The name of the branch in VSTS Git repository. |
| Folder | The name of the folder in VSTS Git branch. |
| VSAccessTokenVariableName | The name of the secure variable (VSToken) that holds the VSTS personal access token. |

Add Webhook

Start a runbook via a simple HTTP POST to a URL

Webhook
VSTSGitFinance

Parameters and run settings
Configure parameters and run settings

Parameters
Sync-VSTSGit

* VSPROJECT ⓘ
MyFirstProject ✓
Mandatory, String

* GITREPO ⓘ
MyFirstProject ✓
Mandatory, String

* GITBRANCH ⓘ
master ✓
Mandatory, String

* FOLDER ⓘ
runbooks ✓
Mandatory, String

* VSACCESTOKENVARIABLENAME ⓘ
vstoken ✓
Mandatory, String

Run Settings
Run on ⓘ
Azure Hybrid Worker

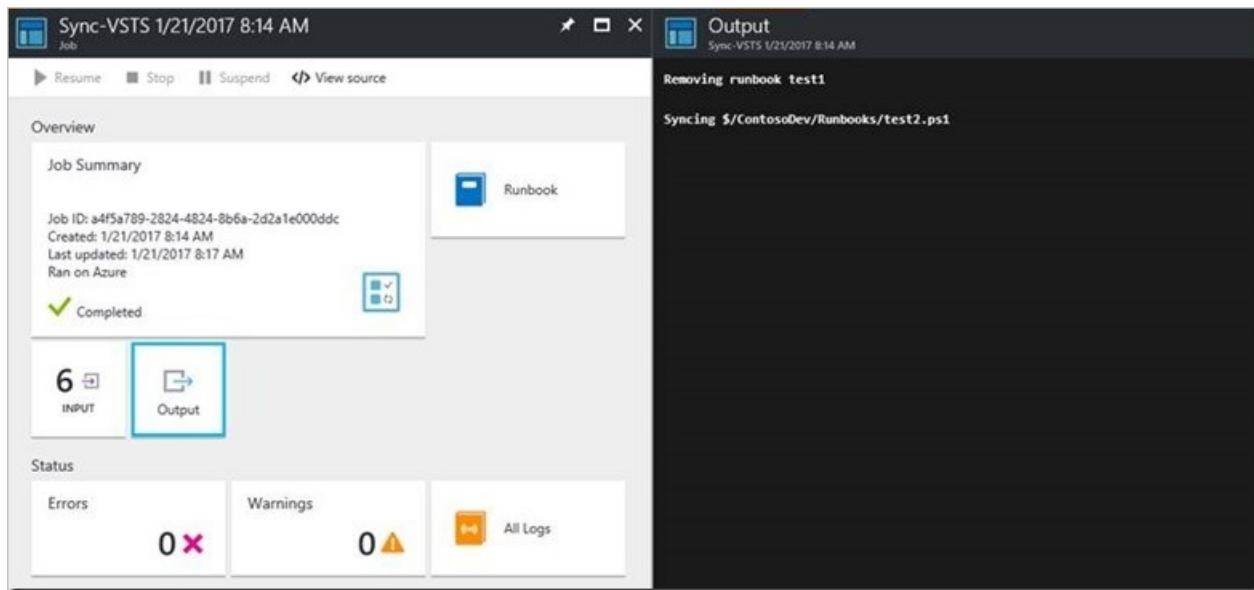
Create
OK

Create a service hook in VSTS for check-ins to the folder that triggers this webhook on code check-in. Select **Web Hooks** as the service to integrate with when you create a new subscription. You can learn more about service hooks on [VSTS Service Hooks documentation](#).

The screenshot shows the 'Service Hooks' section of the VSTS interface. A new subscription is being configured for the 'Web Hooks' consumer, specifically for the 'Code checked in' event. The 'Path' is set to '\$/ContosoDev'. A modal window titled 'EDIT SERVICE HOOKS SUBSCRIPTION' is open, showing the 'Action' configuration. The action selected is 'Post via HTTP', which posts a JSON object representation of the event to a specified URL. The 'SETTINGS' section includes fields for 'URL' (set to 'https://s1events.azure-automation.net/webhooks?token=%2F%2bU3ZwDV'), 'HTTP headers', 'Basic authentication username', 'Basic authentication password', and 'Resource details to send'. Navigation buttons at the bottom of the modal include 'Previous', 'Next', 'Test', 'Finish', and 'Cancel'.

You should now be able to do all check-ins of your runbooks and configurations into VSTS and have these

automatically synched into your automation account.



If you run this runbook manually without being triggered by VSTS, you can leave the webhookdata parameter empty and it does a full sync from the VSTS folder specified.

If you wish to uninstall the scenario, remove the service hook from VSTS, delete the runbook, and the VSToken variable.

Call an Azure Automation runbook from a Log Analytics alert

6/20/2018 • 4 minutes to read • [Edit Online](#)

You can configure an alert in Azure Log Analytics to create an alert record when results match your criteria. That alert can then automatically run an Azure Automation runbook in an attempt to auto-remediate the issue.

For example, an alert might indicate a prolonged spike in processor utilization. Or it might indicate when an application process that's critical to the functionality of a business application fails. A runbook can then write a corresponding event in the Windows event log.

There are two options to call a runbook in the alert configuration:

- Use a webhook.
 - This is the only option available if your Log Analytics workspace is not linked to an Automation account.
 - If you already have an Automation account linked to a Log Analytics workspace, this option is still available.
- Select a runbook directly.
 - This option is available only if the Log Analytics workspace is linked to an Automation account.

Calling a runbook by using a webhook

You can use a webhook to start a particular runbook in Azure Automation through a single HTTP request. Before you configure the [Webhook action for log alerts](#) to call the runbook by using a webhook as an alert action, you need to [create a webhook](#) for the runbook that's called through this method. Remember to record the webhook URL so you can reference it while configuring the alert rule.

Calling a runbook directly

You can install and configure the Automation and Control offering in your Log Analytics workspace. While you're configuring the runbook actions option for the alert, you can view all runbooks from the **Select a runbook** drop-down list and select the specific runbook that you want to run in response to the alert. The selected runbook can run in an Azure workspace or on a hybrid runbook worker.

After you create the alert by using the runbook option, a webhook is created for the runbook. You can see the webhook if you go to the Automation account and open the webhook pane of the selected runbook.

If you delete the alert, the webhook is not deleted. This is not a problem. The webhook just becomes an orphaned item that you should eventually delete manually, to maintain an organized Automation account.

Characteristics of a runbook

Both methods for calling the runbook from the Log Analytics alert have characteristics that you need to understand before you configure your alert rules.

The alert data is in JSON format in a single property called **SearchResult**. This format is for runbook and webhook actions with a standard payload. For webhook actions with custom payloads (including **IncludeSearchResults:True** in **RequestBody**), the property is **SearchResults**.

You must have a runbook input parameter called **WebhookData** that is an **Object** type. It can be mandatory or

optional. The alert passes the search results to the runbook by using this input parameter.

```
param
(
    [Parameter (Mandatory=$true)]
    [object] $WebhookData
)
```

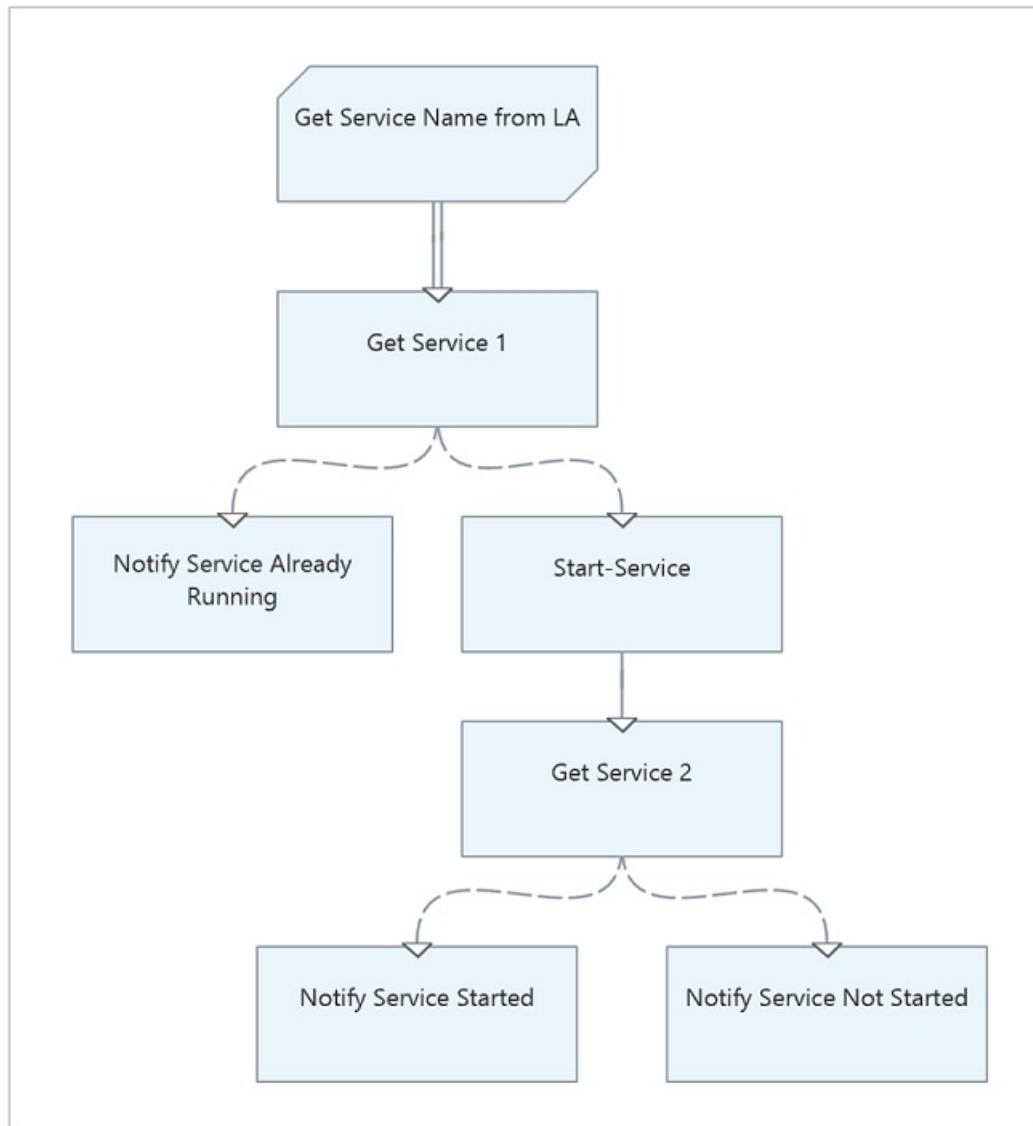
You must also have code to convert **WebhookData** to a PowerShell object.

```
$SearchResult = (ConvertFrom-Json $WebhookData.RequestBody).SearchResult.value
```

\$SearchResult is an array of objects. Each object contains the fields with values from one search result.

Example walkthrough

The following example of a graphical runbook demonstrates how this process works. It starts a Windows service.



The runbook has one input parameter of type **Object** that is called **WebhookData**. It includes the webhook data passed from the alert that contains **SearchResult**.

* Name !
WebhookData

Description !

Type !
Object

Mandatory !
 Yes No

Default value
Mandatory parameters can't have a default value

For this example, we created two custom fields in Log Analytics: **SvcDisplayName_CF** and **SvcState_CF**. These fields extract the service display name and the state of the service (that is, running or stopped) from the event that's written to the system event log. We then created an alert rule with the following search query, so that we can detect when the Print Spooler service is stopped on the Windows system:

```
Type=Event SvcDisplayName_CF="Print Spooler" SvcState_CF="stopped"
```

It can be any service of interest. For this example, we're referencing one of the pre-existing services that are included with the Windows OS. The alert action is configured to execute the runbook used in this example and run on the hybrid runbook worker, which is enabled on the target system.

The runbook code activity **Get Service Name from LA** converts the JSON-formatted string into an object type and filters on the item **SvcDisplayName_CF**. It extracts the display name of the Windows service and passes this value to the next activity, which verifies that the service is stopped before attempting to restart it.

SvcDisplayName_CF is a [custom field](#) that we created in Log Analytics to demonstrate this example.

```
$SearchResult = (ConvertFrom-Json $WebhookData.RequestBody).SearchResult.value  
$SearchResult.SvcDisplayName_CF
```

When the service stops, the alert rule in Log Analytics detects a match, triggers the runbook, and sends the alert context to the runbook. The runbook tries to verify that the service is stopped. If so, the runbook attempts to restart the service, verify that it started correctly, and display the results.

Alternatively, if you don't have your Automation account linked to your Log Analytics workspace, you can configure the alert rule with a webhook action. The webhook action triggers the runbook. It also configures the runbook to convert the JSON-formatted string and filter on **SearchResult** by following the guidance mentioned earlier.

NOTE

If your workspace has been upgraded to the [new Log Analytics query language](#), the webhook payload has changed. Details of the format are in the [Azure Log Analytics REST API](#).

Next steps

- To learn more about creating an Azure Alert using a log search, see [Log alerts in Azure](#).
- To understand how to trigger runbooks by using a webhook, see [Azure Automation webhooks](#).

Deploy an Azure Resource Manager template in an Azure Automation PowerShell runbook

5/21/2018 • 4 minutes to read • [Edit Online](#)

You can write an [Azure Automation PowerShell runbook](#) that deploys an Azure resource by using an [Azure Resource Management template](#).

By doing this, you can automate deployment of Azure resources. You can maintain your Resource Manager templates in a central, secure location such as Azure Storage.

In this topic, we create a PowerShell runbook that uses an Resource Manager template stored in [Azure Storage](#) to deploy a new Azure Storage account.

Prerequisites

To complete this tutorial, you need the following:

- Azure subscription. If you don't have one yet, you can [activate your MSDN subscriber benefits](#) or [\[sign up for a free account\]\(https://azure.microsoft.com/free/\)](#).
- [Automation account](#) to hold the runbook and authenticate to Azure resources. This account must have permission to start and stop the virtual machine.
- [Azure Storage account](#) in which to store the Resource Manager template
- Azure Powershell installed on a local machine. See [Install and configure Azure Powershell](#) for information about how to get Azure PowerShell.

Create the Resource Manager template

For this example, we use an Resource Manager template that deploys a new Azure Storage account.

In a text editor, copy the following text:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "storageAccountType": {
            "type": "string",
            "defaultValue": "Standard_LRS",
            "allowedValues": [
                "Standard_LRS",
                "Standard_GRS",
                "Standard_ZRS",
                "Premium_LRS"
            ],
            "metadata": {
                "description": "Storage Account type"
            }
        }
    },
    "variables": {
        "storageAccountName": "[concat(uniquestring(resourceGroup().id), 'standardsa')]"
    },
    "resources": [
        {
            "type": "Microsoft.Storage/storageAccounts",
            "name": "[variables('storageAccountName')]",
            "apiVersion": "2016-01-01",
            "location": "[resourceGroup().location]",
            "sku": {
                "name": "[parameters('storageAccountType')]"
            },
            "kind": "Storage",
            "properties": {}
        }
    ],
    "outputs": {
        "storageAccountName": {
            "type": "string",
            "value": "[variables('storageAccountName')]"
        }
    }
}
```

Save the file locally as `TemplateTest.json`.

Save the Resource Manager template in Azure Storage

Now we use PowerShell to create an Azure Storage file share and upload the `TemplateTest.json` file. For instructions on how to create a file share and upload a file in the Azure portal, see [Get started with Azure File storage on Windows](#).

Launch PowerShell on your local machine, and run the following commands to create a file share and upload the Resource Manager template to that file share.

```
# Login to Azure
Connect-AzureRmAccount

# Get the access key for your storage account
$key = Get-AzureRmStorageAccountKey -ResourceGroupName 'MyAzureAccount' -Name 'MyStorageAccount'

# Create an Azure Storage context using the first access key
$context = New-AzureStorageContext -StorageAccountName 'MyStorageAccount' -StorageAccountKey $key[0].value

# Create a file share named 'resource-templates' in your Azure Storage account
$fileShare = New-AzureStorageShare -Name 'resource-templates' -Context $context

# Add the TemplateTest.json file to the new file share
# "TemplatePath" is the path where you saved the TemplateTest.json file
$templateFile = 'C:\TemplatePath'
Set-AzureStorageFileContent -ShareName $fileShare.Name -Context $context -Source $templateFile
```

Create the PowerShell runbook script

Now we create a PowerShell script that gets the `TemplateTest.json` file from Azure Storage and deploys the template to create a new Azure Storage account.

In a text editor, paste the following text:

```

param (
    [Parameter(Mandatory=$true)]
    [string]
    $ResourceGroupName,
    [Parameter(Mandatory=$true)]
    [string]
    $StorageAccountName,
    [Parameter(Mandatory=$true)]
    [string]
    $StorageAccountKey,
    [Parameter(Mandatory=$true)]
    [string]
    $StorageFileName
)

# Authenticate to Azure if running from Azure Automation
$ServicePrincipalConnection = Get-AutomationConnection -Name "AzureRunAsConnection"
Connect-AzureRmAccount `-
    -ServicePrincipal `-
    -TenantId $ServicePrincipalConnection.TenantId `-
    -ApplicationId $ServicePrincipalConnection.ApplicationId `-
    -CertificateThumbprint $ServicePrincipalConnection.CertificateThumbprint | Write-Verbose

#Set the parameter values for the Resource Manager template
$Parameters = @{
    "storageAccountType"="Standard_LRS"
}

# Create a new context
$Context = New-AzureStorageContext -StorageAccountName $StorageAccountName -StorageAccountKey
$StorageAccountKey

Get-AzureStorageFileContent -ShareName 'resource-templates' -Context $Context -path 'TemplateTest.json' -Destination 'C:\Temp'

$TemplateFile = Join-Path -Path 'C:\Temp' -ChildPath $StorageFileName

# Deploy the storage account
New-AzureRmResourceGroupDeployment -ResourceGroupName $ResourceGroupName -TemplateFile $TemplateFile -TemplateParameterObject $Parameters

```

Save the file locally as `DeployTemplate.ps1`.

Import and publish the runbook into your Azure Automation account

Now we use PowerShell to import the runbook into your Azure Automation account, and then publish the runbook. For information about how to import and publish a runbook in the Azure portal, see [Creating or importing a runbook in Azure Automation](#).

To import `DeployTemplate.ps1` into your Automation account as a PowerShell runbook, run the following PowerShell commands:

```

# MyPath is the path where you saved DeployTemplate.ps1
# MyResourceGroup is the name of the Azure ResourceGroup that contains your Azure Automation account
# MyAutomationAccount is the name of your Automation account
$importParams = @{
    Path = 'C:\MyPath\DeployTemplate.ps1'
    ResourceGroupName = 'MyResourceGroup'
    AutomationAccountName = 'MyAutomationAccount'
    Type = 'PowerShell'
}
Import-AzureRmAutomationRunbook @importParams

# Publish the runbook
$publishParams = @{
    ResourceGroupName = 'MyResourceGroup'
    AutomationAccountName = 'MyAutomationAccount'
    Name = 'DeployTemplate'
}
Publish-AzureRmAutomationRunbook @publishParams

```

Start the runbook

Now we start the runbook by calling the [Start-AzureRmAutomationRunbook](#) cmdlet.

For information about how to start a runbook in the Azure portal, see [Starting a runbook in Azure Automation](#).

Run the following commands in the PowerShell console:

```

# Set up the parameters for the runbook
$runbookParams = @{
    ResourceGroupName = 'MyResourceGroup'
    StorageAccountName = 'MyStorageAccount'
    StorageAccountKey = $key[0].Value # We got this key earlier
    StorageFileName = 'TemplateTest.json'
}

# Set up parameters for the Start-AzureRmAutomationRunbook cmdlet
$startParams = @{
    ResourceGroupName = 'MyResourceGroup'
    AutomationAccountName = 'MyAutomationAccount'
    Name = 'DeployTemplate'
    Parameters = $runbookParams
}

# Start the runbook
$job = Start-AzureRmAutomationRunbook @startParams

```

The runbook runs, and you can check its status by running `$job.Status`.

The runbook gets the Resource Manager template and uses it to deploy a new Azure Storage account. You can see that the new storage account was created by running the following command:

```
Get-AzureRmStorageAccount
```

Summary

That's it! Now you can use Azure Automation and Azure Storage, and Resource Manager templates to deploy all your Azure resources.

Next steps

- To learn more about Resource Manager templates, see [Azure Resource Manager overview](#)
- To get started with Azure Storage, see [Introduction to Azure Storage](#).
- To find other useful Azure Automation runbooks, see [Runbook and module galleries for Azure Automation](#).
- To find other useful Resource Manager templates, see [Azure Quickstart Templates](#)

Integrate Azure Automation with Event Grid and Microsoft Teams

12/6/2017 • 2 minutes to read • [Edit Online](#)

In this tutorial, you learn how to:

- Import an Event Grid sample runbook.
- Create an optional Microsoft Teams webhook.
- Create a webhook for the runbook.
- Create an Event Grid subscription.
- Create a VM that triggers the runbook.

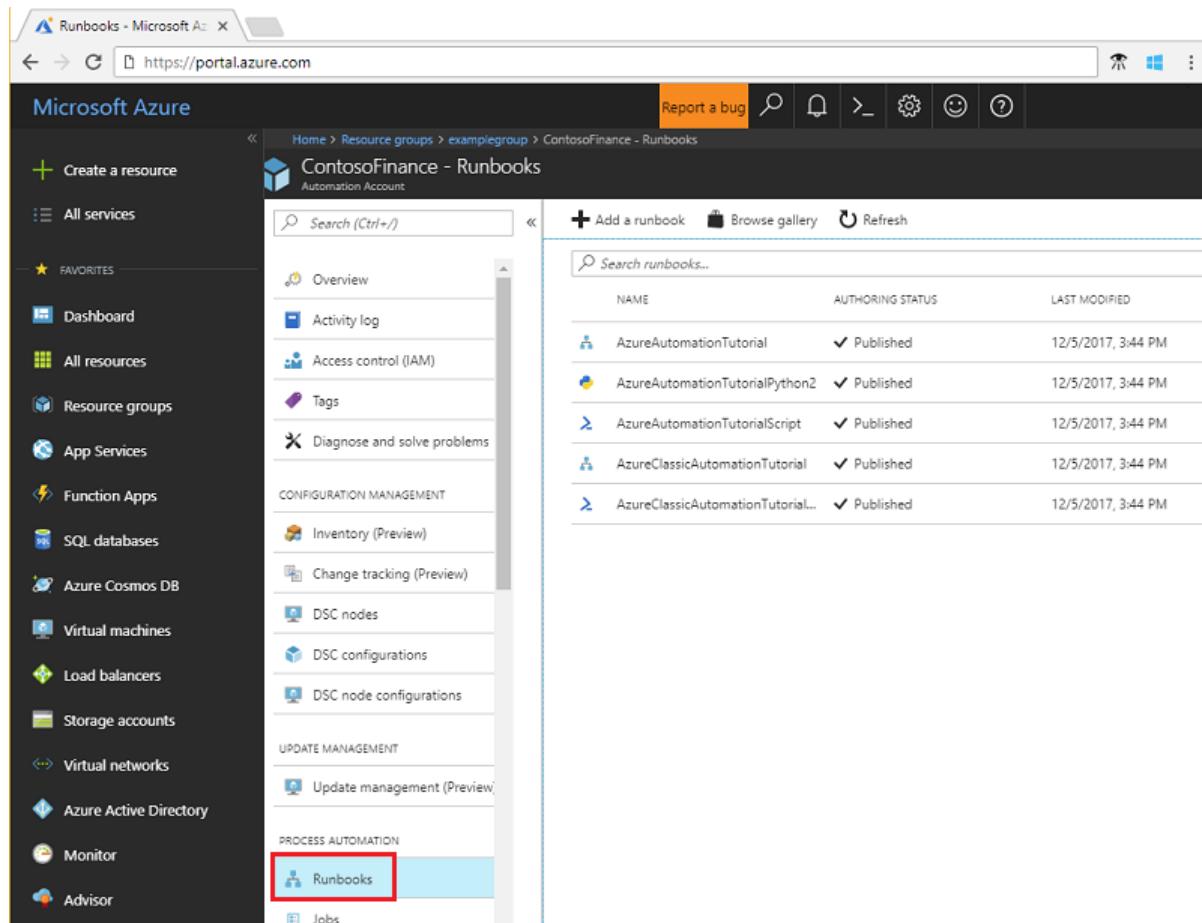
If you don't have an Azure subscription, create a [free account](#) before you begin.

Prerequisites

To complete this tutorial, an [Azure Automation account](#) is required to hold the runbook that is triggered from the Azure Event Grid subscription.

Import an Event Grid sample runbook

1. Select your Automation account, and select the **Runbooks** page.



The screenshot shows the Microsoft Azure portal interface. The URL in the address bar is <https://portal.azure.com>. The page title is "ContosoFinance - Runbooks" under the "Automation Account" section. The left sidebar lists various Azure services like Dashboard, All resources, Resource groups, App Services, Function Apps, SQL databases, Azure Cosmos DB, Virtual machines, Load balancers, Storage accounts, Virtual networks, Azure Active Directory, Monitor, and Advisor. The "Runbooks" link in the sidebar is highlighted with a red box. The main content area displays the "Runbooks" page with a search bar at the top. Below the search bar is a table listing six runbooks: "AzureAutomationTutorial", "AzureAutomationTutorialPython2", "AzureAutomationTutorialScript", "AzureClassicAutomationTutorial", and two partially visible entries. The columns in the table are NAME, AUTHORIZING STATUS, and LAST MODIFIED. All runbooks are listed as Published and were last modified on 12/5/2017, 3:44 PM.

| NAME | AUTHORIZING STATUS | LAST MODIFIED |
|-----------------------------------|--------------------|--------------------|
| AzureAutomationTutorial | ✓ Published | 12/5/2017, 3:44 PM |
| AzureAutomationTutorialPython2 | ✓ Published | 12/5/2017, 3:44 PM |
| AzureAutomationTutorialScript | ✓ Published | 12/5/2017, 3:44 PM |
| AzureClassicAutomationTutorial | ✓ Published | 12/5/2017, 3:44 PM |
| AzureClassicAutomationTutorial... | ✓ Published | 12/5/2017, 3:44 PM |

2. Select the **Browse gallery** button.

3. Search for **Event Grid**, and select **Integrating Azure Automation with Event grid**.

The screenshot shows the 'Browse Gallery' interface. A search bar at the top contains the text 'Event Grid'. To the right of the search bar is a dropdown menu set to 'Last updated'. Below the search bar, a list item is displayed: 'Integrating Azure Automation with Event grid' (PowerShell Runbook). To the left of the title is a blue icon resembling a greater than sign (>). The description below the title states: 'This sample Automation runbook integrates with Azure event grid subscriptions to get notified when a write command is performed against an Azure VM. The runbook adds a cost tag to the VM if it doesn't exist. It also sends an optional notification to a Microsoft Tag: Azure Automation, Microsoft Azure Virtual Machines, Event Grid'. To the right of the description, there is contact information: 'Created by: SC Automation Product Team', '0 downloads', and 'Last updated: 11/28/2017'.

4. Select **Import** and name it **Watch-VMWrite**.
5. After it has imported, select **Edit** to view the runbook source. Select the **Publish** button.

Create an optional Microsoft Teams webhook

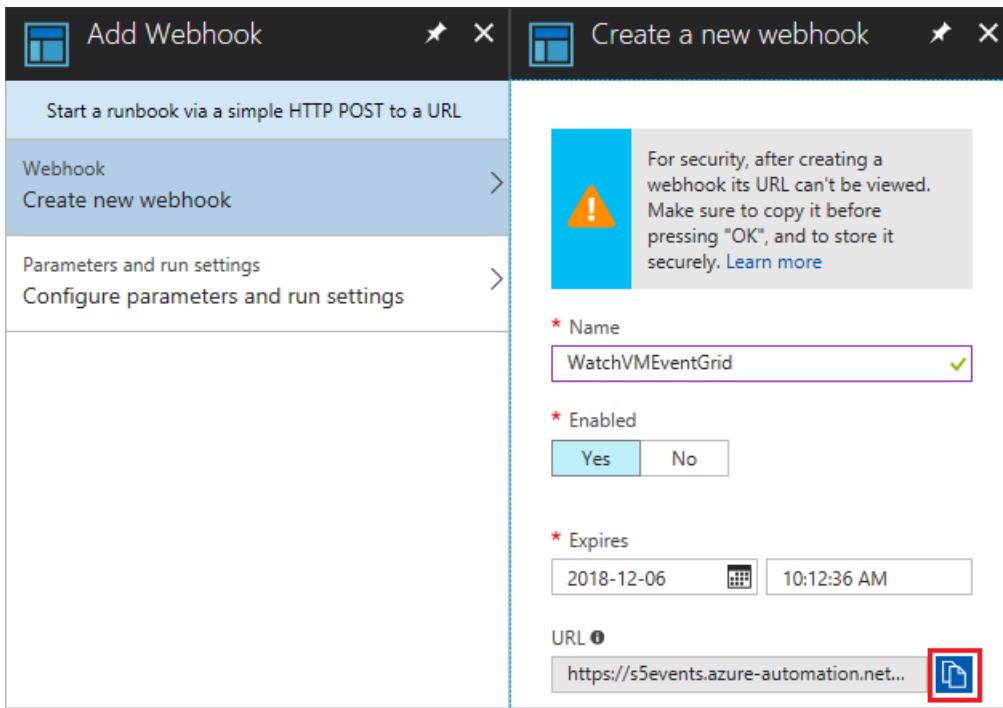
1. In Microsoft Teams, select **More Options** next to the channel name, and then select **Connectors**.

The screenshot shows a Microsoft Teams channel named 'AzureAutomationCloudEvents'. The 'General' tab is selected. A context menu is open over the channel name, with the 'Connectors' option highlighted with a red box. Other options in the menu include 'Follow this channel', 'Open in SharePoint', 'Add channel', 'Get email address', 'Get link to channel', and 'Leave the team'.

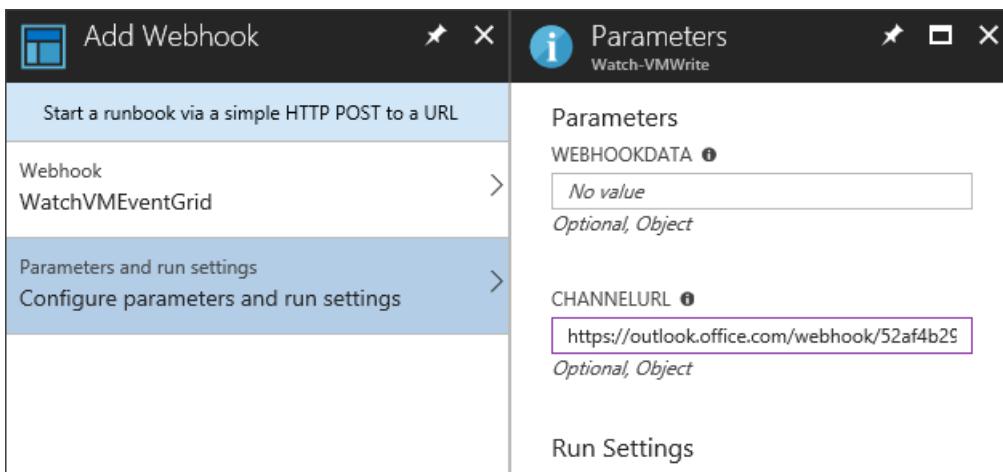
2. Scroll through the list of connectors to **Incoming Webhook**, and select **Add**.
3. Enter **AzureAutomationIntegration** for the name, and select **Create**.
4. Copy the webhook to the clipboard, and save it. The webhook URL is used to send information to Microsoft Teams.
5. Select **Done** to save the webhook.

Create a webhook for the runbook

1. Open the Watch-VMWrite runbook.
2. Select **Webhooks**, and select the **Add Webhook** button.
3. Enter **WatchVMEventGrid** for the name. Copy the URL to the clipboard, and save it.



4. Select **Configure parameters and run settings**, and enter the Microsoft Teams webhook URL for **CHANNELURL**. Leave **WEBHOOKDATA** blank.



5. Select **OK** to create the Automation runbook webhook.

Create an Event Grid subscription

1. On the **Automation Account** overview page, select **Event grid**.

2. Select the **+ Event Subscription** button.
3. Configure the subscription with the following information:

- Enter **AzureAutomation** for the name.
- In **Topic Type**, select **Azure Subscriptions**.
- Clear the **Subscribe to all event types** check box.
- In **Event Types**, select **Resource Write Success**.
- In **Subscriber Endpoint**, enter the webhook URL for the Watch-VMWrite runbook.
- In **Prefix Filter**, enter the subscription and resource group where you want to look for the new VMs created. It should look like:

```
/subscriptions/<subscription-id>/resourcegroups/<resource-group-name>/providers/Microsoft.Compute/virtualMachines
```

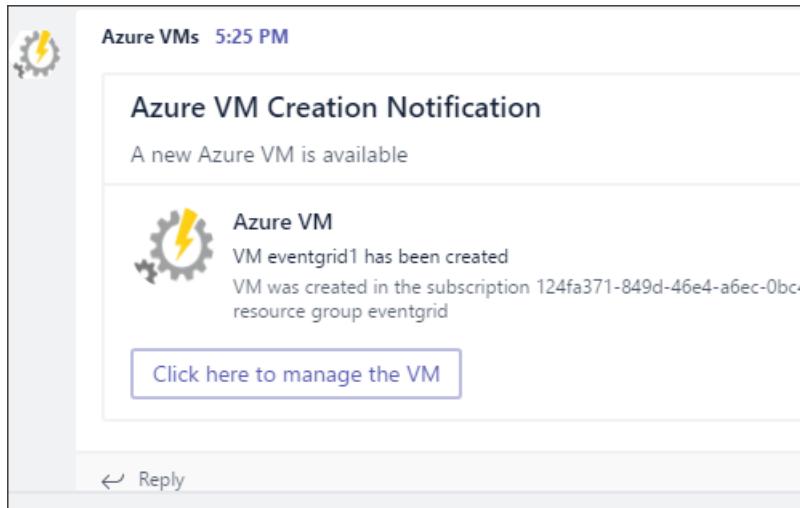
4. Select **Create** to save the Event Grid subscription.

Create a VM that triggers the runbook

1. Create a new VM in the resource group you specified in the Event Grid subscription prefix filter.
2. The Watch-VMWrite runbook should be called and a new tag added to the VM.

| | |
|------|-------------------------------|
| Name | Value |
| Cost | {"AutoShutdownStartup": true} |

3. A new message is sent to the Microsoft Teams channel.



Next steps

In this tutorial, you set up integration between Event Grid and Automation. You learned how to:

- Import an Event Grid sample runbook.
- Create an optional Microsoft Teams webhook.
- Create a webhook for the runbook.
- Create an Event Grid subscription.
- Create a VM that triggers the runbook.

[Create and route custom events with Event Grid](#)

Forward job status and job streams from Automation to Log Analytics

6/20/2018 • 7 minutes to read • [Edit Online](#)

Automation can send runbook job status and job streams to your Log Analytics workspace. Job logs and job streams are visible in the Azure portal, or with PowerShell, for individual jobs and this allows you to perform simple investigations. Now with Log Analytics you can:

- Get insight on your Automation jobs.
- Trigger an email or alert based on your runbook job status (for example, failed or suspended).
- Write advanced queries across your job streams.
- Correlate jobs across Automation accounts.
- Visualize your job history over time.

Prerequisites and deployment considerations

To start sending your Automation logs to Log Analytics, you need:

- The November 2016 or later release of [Azure PowerShell](#) (v2.3.0).
- A Log Analytics workspace. For more information, see [Get started with Log Analytics](#).
- The ResourceId for your Azure Automation account.

To find the ResourceId for your Azure Automation account:

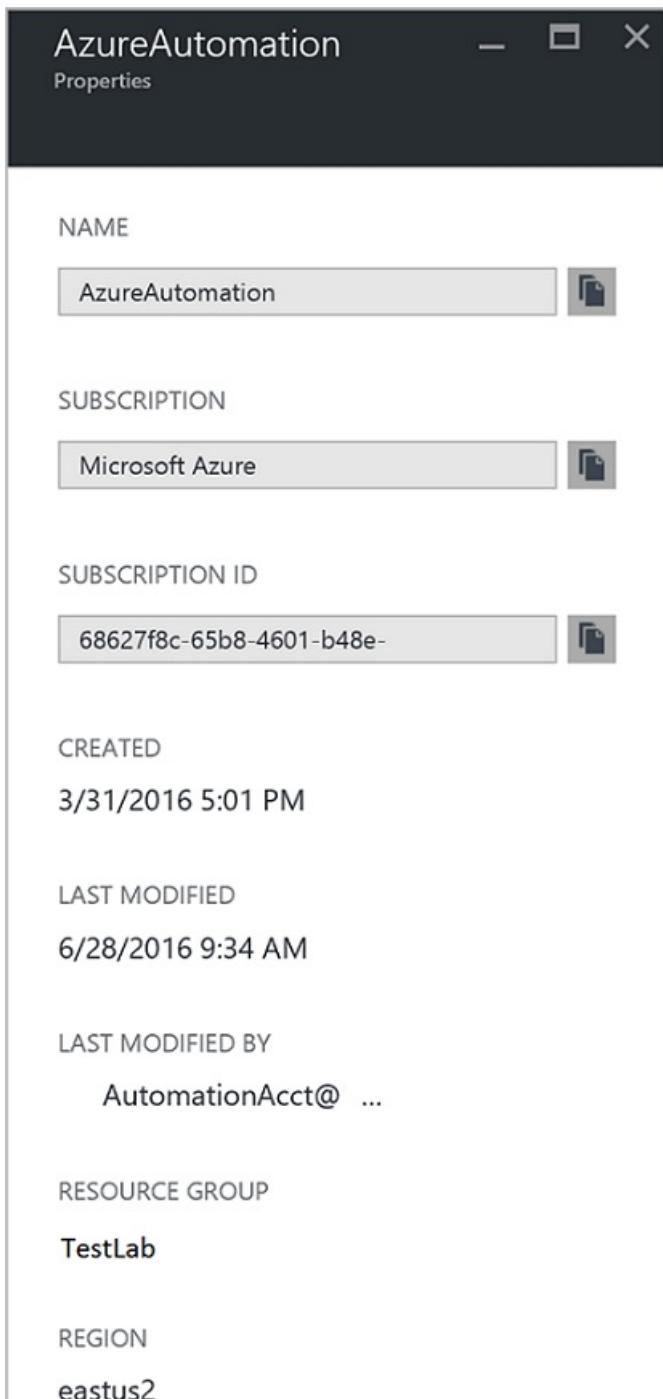
```
# Find the ResourceId for the Automation Account  
Find-AzureRmResource -ResourceType "Microsoft.Automation/automationAccounts"
```

To find the ResourceId for your Log Analytics workspace, run the following PowerShell:

```
# Find the ResourceId for the Log Analytics workspace  
Find-AzureRmResource -ResourceType "Microsoft.OperationalInsights/workspaces"
```

If you have more than one Automation accounts, or workspaces, in the output of the preceding commands, find the *Name* you need to configure and copy the value for *ResourceId*.

If you need to find the *Name* of your Automation account, in the Azure portal select your Automation account from the **Automation account** blade and select **All settings**. From the **All settings** blade, under **Account Settings** select **Properties**. In the **Properties** blade, you can note these values.



Set up integration with Log Analytics

1. On your computer, start **Windows PowerShell** from the **Start** screen.
2. Run the following PowerShell, and edit the value for the `[your resource id]` and `[resource id of the log analytics workspace]` with the values from the preceding step.

```
$workspaceId = "[resource id of the log analytics workspace]"
$automationAccountId = "[resource id of your automation account]"

Set-AzureRmDiagnosticSetting -ResourceId $automationAccountId -WorkspaceId $workspaceId -Enabled $true
```

After running this script, you'll see records in Log Analytics within 10 minutes of new JobLogs or JobStreams being written.

To see the logs, run the following query in Log Analytics log search:

```
AzureDiagnostics | where ResourceProvider == "MICROSOFT.AUTOMATION"
```

Verify configuration

To confirm that your Automation account is sending logs to your Log Analytics workspace, check that diagnostics are correctly configured on the Automation account by using the following PowerShell:

```
Get-AzureRmDiagnosticSetting -ResourceId $automationAccountId
```

In the output ensure that:

- Under *Logs*, the value for *Enabled* is *True*.
- The value of *WorkspaceId* is set to the ResourceId of your Log Analytics workspace.

Log Analytics records

Diagnostics from Azure Automation creates two types of records in Log Analytics and are tagged as **AzureDiagnostics**. The following queries use the upgraded query language to Log Analytics. For information on common queries between legacy query language and the new Azure Log Analytics query language visit [Legacy to new Azure Log Analytics Query Language cheat sheet](#)

Job Logs

| PROPERTY | DESCRIPTION |
|---------------|--|
| TimeGenerated | Date and time when the runbook job executed. |
| RunbookName_s | The name of the runbook. |
| Caller_s | Who initiated the operation. Possible values are either an email address or system for scheduled jobs. |
| Tenant_g | GUID that identifies the tenant for the Caller. |
| JobId_g | GUID that is the Id of the runbook job. |
| ResultType | The status of the runbook job. Possible values are:
- New
- Started
- Stopped
- Suspended
- Failed
- Completed |
| Category | Classification of the type of data. For Automation, the value is JobLogs. |
| OperationName | Specifies the type of operation performed in Azure. For Automation, the value is Job. |
| Resource | Name of the Automation account |
| SourceSystem | How Log Analytics collected the data. Always Azure for Azure diagnostics. |

| PROPERTY | DESCRIPTION |
|-------------------|---|
| ResultDescription | Describes the runbook job result state. Possible values are:
- Job is started
- Job Failed
- Job Completed |
| CorrelationId | GUID that is the Correlation Id of the runbook job. |
| ResourceId | Specifies the Azure Automation account resource id of the runbook. |
| SubscriptionId | The Azure subscription Id (GUID) for the Automation account. |
| ResourceGroup | Name of the resource group for the Automation account. |
| ResourceProvider | MICROSOFT.AUTOMATION |
| ResourceType | AUTOMATIONACCOUNTS |

Job Streams

| PROPERTY | DESCRIPTION |
|---------------|--|
| TimeGenerated | Date and time when the runbook job executed. |
| RunbookName_s | The name of the runbook. |
| Caller_s | Who initiated the operation. Possible values are either an email address or system for scheduled jobs. |
| StreamType_s | The type of job stream. Possible values are:
- Progress
- Output
- Warning
- Error
- Debug
- Verbose |
| Tenant_g | GUID that identifies the tenant for the Caller. |
| JobId_g | GUID that is the Id of the runbook job. |
| ResultType | The status of the runbook job. Possible values are:
- In Progress |
| Category | Classification of the type of data. For Automation, the value is JobStreams. |
| OperationName | Specifies the type of operation performed in Azure. For Automation, the value is Job. |
| Resource | Name of the Automation account |

| PROPERTY | DESCRIPTION |
|-------------------|---|
| SourceSystem | How Log Analytics collected the data. Always Azure for Azure diagnostics. |
| ResultDescription | Includes the output stream from the runbook. |
| CorrelationId | GUID that is the Correlation Id of the runbook job. |
| ResourceId | Specifies the Azure Automation account resource id of the runbook. |
| SubscriptionId | The Azure subscription Id (GUID) for the Automation account. |
| ResourceGroup | Name of the resource group for the Automation account. |
| ResourceProvider | MICROSOFT.AUTOMATION |
| ResourceType | AUTOMATIONACCOUNTS |

Viewing Automation Logs in Log Analytics

Now that you started sending your Automation job logs to Log Analytics, let's see what you can do with these logs inside Log Analytics.

To see the logs, run the following query: `AzureDiagnostics | where ResourceProvider == "MICROSOFT.AUTOMATION"`

Send an email when a runbook job fails or suspends

One of the top customer asks is for the ability to send an email or a text when something goes wrong with a runbook job.

To create an alert rule, you start by creating a log search for the runbook job records that should invoke the alert. Click the **Alert** button to create and configure the alert rule.

1. From the Log Analytics Overview page, click **Log Search**.

2. Create a log search query for your alert by typing the following search into the query field:

```
AzureDiagnostics | where ResourceProvider == "MICROSOFT.AUTOMATION" and Category == "JobLogs" and
(ResultType == "Failed" or ResultType == "Suspended")
```

You can also group by the RunbookName by using:

```
AzureDiagnostics | where ResourceProvider == "MICROSOFT.AUTOMATION" and Category == "JobLogs" and
(ResultType == "Failed" or ResultType == "Suspended") | summarize AggregatedValue = count() by
RunbookName_s
```

If you set up logs from more than one Automation account or subscription to your workspace, you can group your alerts by subscription and Automation account. Automation account name can be found in the Resource field in the search of JobLogs.

3. To open the **Create rule** screen, click **+ New Alert Rule** at the top of the page. For more information on the options to configure the alert, see [Log alerts in Azure](#).

Find all jobs that have completed with errors

In addition to alerting on failures, you can find when a runbook job has a non-terminating error. In these cases PowerShell produces an error stream, but the non-terminating errors don't cause your job to suspend or fail.

1. In your Log Analytics workspace, click **Log Search**.
2. In the query field, type

```
AzureDiagnostics | where ResourceProvider == "MICROSOFT.AUTOMATION" and Category == "JobStreams" and StreamType_s == "Error" | summarize AggregatedValue = count() by JobId_g
```

and then click the **Search** button.

View job streams for a job

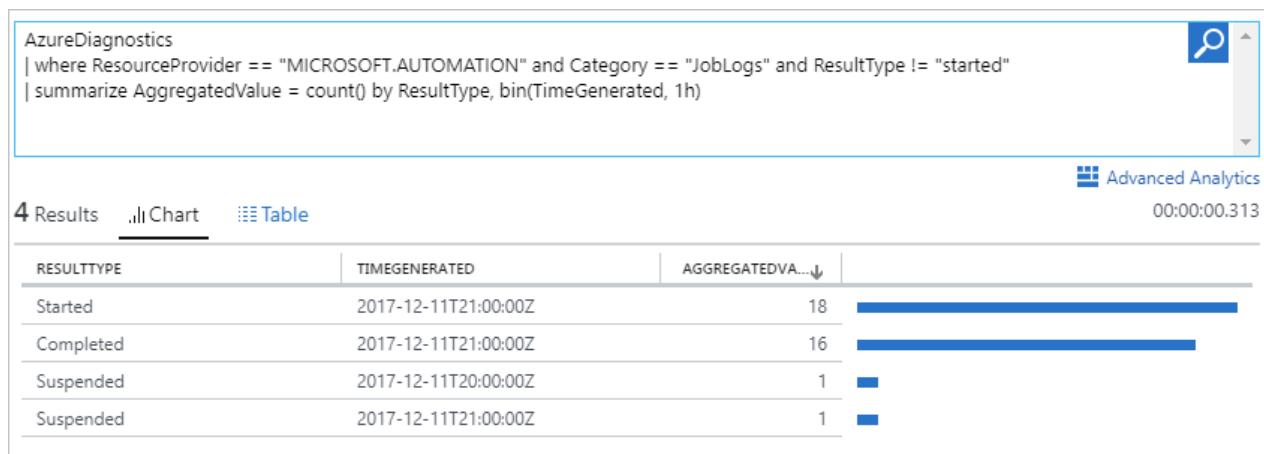
When you're debugging a job, you may also want to look into the job streams. The following query shows all the streams for a single job with GUID 2ebd22ea-e05e-4eb9-9d76-d73cbd4356e0:

```
AzureDiagnostics | where ResourceProvider == "MICROSOFT.AUTOMATION" and Category == "JobStreams" and JobId_g == "2ebd22ea-e05e-4eb9-9d76-d73cbd4356e0" | sort by TimeGenerated asc | project ResultDescription
```

View historical job status

Finally, you may want to visualize your job history over time. You can use this query to search for the status of your jobs over time.

```
AzureDiagnostics | where ResourceProvider == "MICROSOFT.AUTOMATION" and Category == "JobLogs" and ResultType != "started" | summarize AggregatedValue = count() by ResultType, bin(TimeGenerated, 1h)
```



Summary

By sending your Automation job status and stream data to Log Analytics, you can get better insight into the status of your Automation jobs by:

- Setting up alerts to notify you when there is an issue.
- Using custom views and search queries to visualize your runbook results, runbook job status, and other related key indicators or metrics.

Log Analytics provides greater operational visibility to your Automation jobs and can help address incidents quicker.

Next steps

- To learn more about how to construct different search queries and review the Automation job logs with Log Analytics, see [Log searches in Log Analytics](#).
- To understand how to create and retrieve output and error messages from runbooks, see [Runbook output and messages](#).
- To learn more about runbook execution, how to monitor runbook jobs, and other technical details, see [Track a runbook job](#).
- To learn more about Log Analytics and data collection sources, see [Collecting Azure storage data in Log Analytics overview](#).

How to unlink your Automation account from a Log Analytics workspace

5/21/2018 • 2 minutes to read • [Edit Online](#)

Azure Automation integrates with Log Analytics to not only support monitoring of your runbook jobs across all of your Automation accounts, but is also required when you import the following solutions that are dependent on Log Analytics:

- [Update Management](#)
- [Change Tracking](#)
- [Start/Stop VMs during off-hours](#)

If you decide you no longer wish to integrate your Automation account with Log Analytics, you can unlink your account directly from the Azure portal. Before you proceed, you first need to remove the solutions mentioned earlier, otherwise this process will be prevented from proceeding. Review the topic for the particular solution you have imported to understand the steps required to remove it.

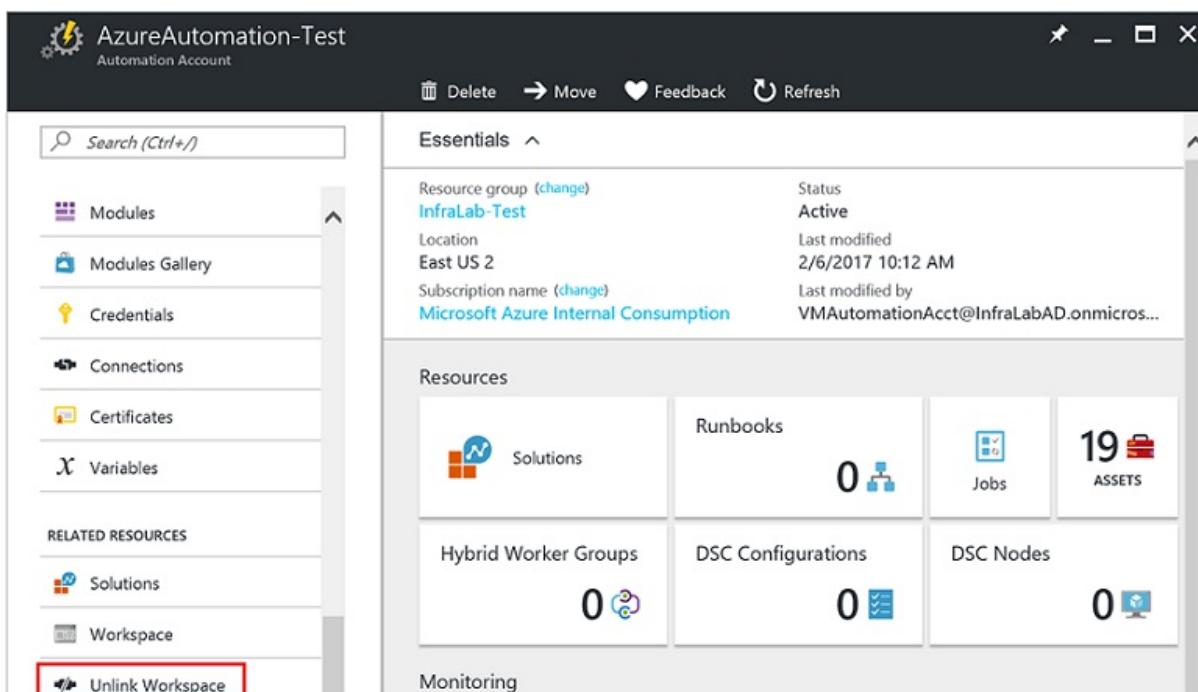
After you remove these solutions you can perform the following steps to unlink your Automation account.

NOTE

Some solutions including earlier versions of the Azure SQL monitoring solution may have created automation assets and may also need to be removed prior to unlinking the workspace.

Unlink workspace

1. From the Azure portal, open your Automation account, and on the Automation account page select **Unlink workspace** under the section **Related Resources** on the left.



2. On the Unlink workspace page, click **Unlink workspace**.

The screenshot shows the Azure Automation - Unlink Workspace page. The left sidebar contains a search bar and links to Modules, Modules Gallery, Credentials, Connections, Certificates, Variables, Solutions, Workspace, and Unlink Workspace. The 'Unlink Workspace' link is highlighted with a blue background. The main content area is titled 'Unlink Workspace' and contains instructions for unlinking the Automation account and Log Analytics workspace. It lists solutions that must be removed first: Update Management, ChangeTracking, and Start/Stop VMs during off-hours. After removing these, the 'Unlink Workspace' button can be clicked to complete the unlinking. There is also a note about optional removal of items after removing the Update Management solution, such as update schedules and hybrid worker groups.

You will receive a prompt verifying you wish to proceed.

3. While Azure Automation attempts to unlink the account your Log Analytics workspace, you can track the progress under **Notifications** from the menu.

If you used the Update Management solution, optionally you may want to remove the following items that are no longer needed after you remove the solution.

- Update schedules. Each will have names that match the update deployments you created)
- Hybrid worker groups created for the solution. Each will be named similarly to machine1.contoso.com_9ceb8108-26c9-4051-b6b3-227600d715c8).

If you used the Start/Stop VMs during off-hours solution, optionally you may want to remove the following items that are no longer needed after you remove the solution.

- Start and stop VM runbook schedules
- Start and stop VM runbooks
- Variables

Next steps

To reconfigure your Automation account to integrate with Log Analytics, see [Forward job status and job streams from Automation to Log Analytics](#).

Migrating from Orchestrator to Azure Automation (Beta)

5/21/2018 • 9 minutes to read • [Edit Online](#)

Runbooks in [System Center Orchestrator](#) are based on activities from integration packs that are written specifically for Orchestrator while runbooks in Azure Automation are based on Windows PowerShell. [Graphical runbooks](#) in Azure Automation have a similar appearance to Orchestrator runbooks with their activities representing PowerShell cmdlets, child runbooks, and assets.

The [System Center Orchestrator Migration Toolkit](#) includes tools to assist you in converting runbooks from Orchestrator to Azure Automation. In addition to converting the runbooks themselves, you must convert the integration packs with the activities that the runbooks use to integration modules with Windows PowerShell cmdlets.

Following is the basic process for converting Orchestrator runbooks to Azure Automation. Each of these steps is described in detail in the sections below.

1. Download the [System Center Orchestrator Migration Toolkit](#) which contains the tools and modules discussed in this article.
2. Import [Standard Activities Module](#) into Azure Automation. This includes converted versions of standard Orchestrator activities that may be used by converted runbooks.
3. Import [System Center Orchestrator Integration Modules](#) into Azure Automation for those integration packs used by your runbooks that access System Center.
4. Convert custom and third party integration packs using the [Integration Pack Converter](#) and import into Azure Automation.
5. Convert Orchestrator runbooks using the [Runbook Converter](#) and install in Azure Automation.
6. Manually create required Orchestrator assets in Azure Automation since the Runbook Converter does not convert these resources.
7. Configure a [Hybrid Runbook Worker](#) in your local data center to run converted runbooks that will access local resources.

Service Management Automation

[Service Management Automation](#) (SMA) stores and runs runbooks in your local data center like Orchestrator, and it uses the same integration modules as Azure Automation. The [Runbook Converter](#) converts Orchestrator runbooks to graphical runbooks though which are not supported in SMA. You can still install the [Standard Activities Module](#) and [System Center Orchestrator Integration Modules](#) into SMA, but you must manually [rewrite](#) your runbooks.

Hybrid Runbook Worker

Runbooks in Orchestrator are stored on a database server and run on runbook servers, both in your local data center. Runbooks in Azure Automation are stored in the Azure cloud and can run in your local data center using a [Hybrid Runbook Worker](#). This is how you will usually run runbooks converted from Orchestrator since they are designed to run on local servers.

Integration Pack Converter

The Integration Pack Converter converts integration packs that were created using the [Orchestrator Integration](#)

[Toolkit \(OIT\)](#) to integration modules based on Windows PowerShell that can be imported into Azure Automation or Service Management Automation.

When you run the Integration Pack Converter, you are presented with a wizard that will allow you to select an integration pack (.oip) file. The wizard then lists the activities included in that integration pack and allows you to select which will be migrated. When you complete the wizard, it creates an integration module that includes a corresponding cmdlet for each of the activities in the original integration pack.

Parameters

Any properties of an activity in the integration pack are converted to parameters of the corresponding cmdlet in the integration module. Windows PowerShell cmdlets have a set of [common parameters](#) that can be used with all cmdlets. For example, the -Verbose parameter causes a cmdlet to output detailed information about its operation. No cmdlet may have a parameter with the same name as a common parameter. If an activity does have a property with the same name as a common parameter, the wizard will prompt you to provide another name for the parameter.

Monitor activities

Monitor runbooks in Orchestrator start with a [monitor activity](#) and run continuously waiting to be invoked by a particular event. Azure Automation does not support monitor runbooks, so any monitor activities in the integration pack will not be converted. Instead, a placeholder cmdlet is created in the integration module for the monitor activity. This cmdlet has no functionality, but it allows any converted runbook that uses it to be installed. This runbook will not be able to run in Azure Automation, but it can be installed so that you can modify it.

Integration packs that cannot be converted

Integration packs that were not created with OIT cannot be converted with the Integration Pack Converter. There are also some integration packs provided by Microsoft that cannot currently be converted with this tool. Converted versions of these integration packs have been [provided for download](#) so that they can be installed in Azure Automation or Service Management Automation.

Standard Activities Module

Orchestrator includes a set of [standard activities](#) that are not included in an integration pack but are used by many runbooks. The Standard Activities module is an integration module that includes a cmdlet equivalent for each of these activities. You must install this integration module in Azure Automation before importing any converted runbooks that use a standard activity.

In addition to supporting converted runbooks, the cmdlets in the standard activities module can be used by someone familiar with Orchestrator to build new runbooks in Azure Automation. While the functionality of all of the standard activities can be performed with cmdlets, they may operate differently. The cmdlets in the converted standard activities module will work the same as their corresponding activities and use the same parameters. This can help the existing Orchestrator runbook author in their transition to Azure Automation runbooks.

System Center Orchestrator Integration Modules

Microsoft provides [integration packs](#) for building runbooks to automate System Center components and other products. Some of these integration packs are currently based on OIT but cannot currently be converted to integration modules because of known issues. [System Center Orchestrator Integration Modules](#) includes converted versions of these integration packs that can be imported into Azure Automation and Service Management Automation.

By the RTM version of this tool, updated versions of the integration packs based on OIT that can be converted with the Integration Pack Converter will be published. Guidance will also be provided to assist you in converting runbooks using activities from the integration packs not based on OIT.

Runbook Converter

The Runbook Converter converts Orchestrator runbooks into [graphical runbooks](#) that can be imported into Azure Automation.

Runbook Converter is implemented as a PowerShell module with a cmdlet called **ConvertFrom-SCORunbook** that performs the conversion. When you install the tool, it will create a shortcut to a PowerShell session that loads the cmdlet.

Following is the basic process to convert an Orchestrator runbook and import it into Azure Automation. The following sections provide further details on using the tool and working with converted runbooks.

1. Export one or more runbooks from Orchestrator.
2. Obtain integration modules for all activities in the runbook.
3. Convert the Orchestrator runbooks in the exported file.
4. Review information in logs to validate the conversion and to determine any required manual tasks.
5. Import converted runbooks into Azure Automation.
6. Create any required assets in Azure Automation.
7. Edit the runbook in Azure Automation to modify any required activities.

Using Runbook Converter

The syntax for **ConvertFrom-SCORunbook** is as follows:

```
ConvertFrom-SCORunbook -RunbookPath <string> -Module <string[]> -OutputFolder <string>
```

- RunbookPath - Path to the export file containing the runbooks to convert.
- Module - Comma delimited list of integration modules containing activities in the runbooks.
- OutputFolder - Path to the folder to create converted graphical runbooks.

The following example command converts the runbooks in an export file called **MyRunbooks.ois_export**. These runbooks use the Active Directory and Data Protection Manager integration packs.

```
ConvertFrom-SCORunbook -RunbookPath "c:\runbooks\MyRunbooks.ois_export" -Module  
c:\ip\SystemCenter_IntegrationModule_ActiveDirectory.zip,c:\ip\SystemCenter_IntegrationModule_DPM.zip -  
OutputFolder "c:\runbooks"
```

Log files

The Runbook Converter will create the following log files in the same location as the converted runbook. If the files already exist, then they will be overwritten with information from the last conversion.

| FILE | CONTENTS |
|----------------------------------|--|
| Runbook Converter - Progress.log | Detailed steps of the conversion including information for each activity successfully converted and warning for each activity not converted. |
| Runbook Converter - Summary.log | Summary of the last conversion including any warnings and follow up tasks that you need to perform such as creating a variable required for the converted runbook. |

Exporting runbooks from Orchestrator

The Runbook Converter works with an export file from Orchestrator that contains one or more runbooks. It will create a corresponding Azure Automation runbook for each Orchestrator runbook in the export file.

To export a runbook from Orchestrator, right-click the name of the runbook in Runbook Designer and select **Export**. To export all runbooks in a folder, right-click the name of the folder and select **Export**.

Runbook activities

The Runbook Converter converts each activity in the Orchestrator runbook to a corresponding activity in Azure Automation. For those activities that can't be converted, a placeholder activity is created in the runbook with warning text. After you import the converted runbook into Azure Automation, you must replace any of these activities with valid activities that perform the required functionality.

Any Orchestrator activities in the [Standard Activities Module](#) will be converted. There are some standard Orchestrator activities that are not in this module though and are not converted. For example, **Send Platform Event** has no Azure Automation equivalent since the event is specific to Orchestrator.

[Monitor activities](#) are not converted since there is no equivalent to them in Azure Automation. The exception are monitor activities in [converted integration packs](#) that will be converted to the placeholder activity.

Any activity from a [converted integration pack](#) will be converted if you provide the path to the integration module with the **modules** parameter. For System Center Integration Packs, you can use the [System Center Orchestrator Integration Modules](#).

Orchestrator resources

The Runbook Converter only converts runbooks, not other Orchestrator resources such as counters, variables, or connections. Counters are not supported in Azure Automation. Variables and connections are supported, but you must create them manually. The log files will inform you if the runbook requires such resources and specify corresponding resources that you need to create in Azure Automation for the converted runbook to operate properly.

For example, a runbook may use a variable to populate a particular value in an activity. The converted runbook will convert the activity and specify a variable asset in Azure Automation with the same name as the Orchestrator variable. This will be noted in the **Runbook Converter - Summary.log** file that is created after the conversion. You will need to manually create this variable asset in Azure Automation before using the runbook.

Input parameters

Runbooks in Orchestrator accept input parameters with the **Initialize Data** activity. If the runbook being converted includes this activity, then an [input parameter](#) in the Azure Automation runbook is created for each parameter in the activity. A [Workflow Script control](#) activity is created in the converted runbook that retrieves and returns each parameter. Any activities in the runbook that use an input parameter refer to the output from this activity.

The reason that this strategy is used is to best mirror the functionality in the Orchestrator runbook. Activities in new graphical runbooks should refer directly to input parameters using a Runbook input data source.

Invoke Runbook activity

Runbooks in Orchestrator start other runbooks with the **Invoke Runbook** activity. If the runbook being converted includes this activity and the **Wait for completion** option is set, then a runbook activity is created for it in the converted runbook. If the **Wait for completion** option is not set, then a Workflow Script activity is created that uses **Start-AzureAutomationRunbook** to start the runbook. After you import the converted runbook into Azure Automation, you must modify this activity with the information specified in the activity.

Related articles

- [System Center 2012 - Orchestrator](#)
- [Service Management Automation](#)
- [Hybrid Runbook Worker](#)
- [Orchestrator Standard Activities](#)

- [Download System Center Orchestrator Migration Toolkit](#)

Migrate Automation Account and resources

5/21/2018 • 2 minutes to read • [Edit Online](#)

For Automation accounts and its associated resources (that is, assets, runbooks, modules, etc.) that you have created in the Azure portal and want to migrate from one resource group to another or from one subscription to another, you can accomplish this easily with the [move resources](#) feature available in the Azure portal. However, before proceeding with this action, you should first review the following [checklist before moving resources](#) and additionally, the following list specific to Automation.

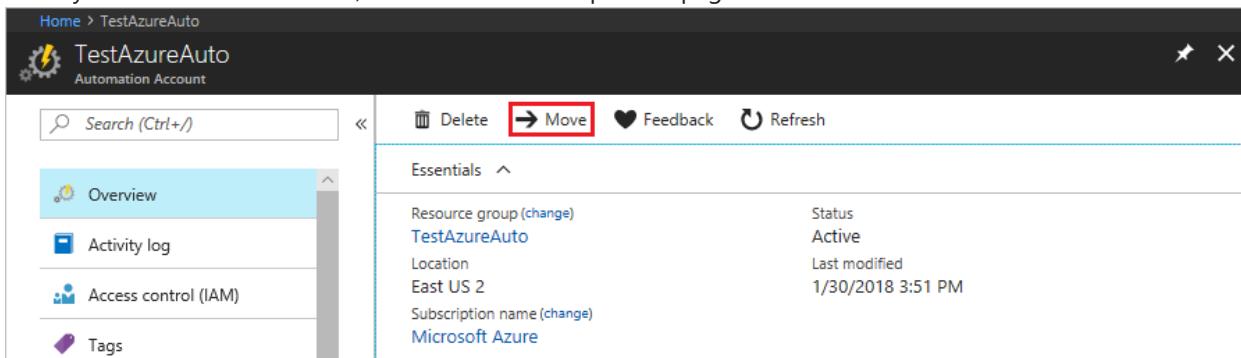
1. The destination subscription/resource group must be in same region as the source. Meaning, Automation accounts cannot be moved across regions.
2. When moving resources (for example, runbooks, jobs, etc.), both the source group and the target group are locked for the duration of the operation. Write and delete operations are blocked on the groups until the move completes.
3. Any runbooks or variables, which reference a resource or subscription ID from the existing subscription, should be updated after migration is completed.

NOTE

This feature does not support moving Classic automation resources.

To move the Automation Account using the portal

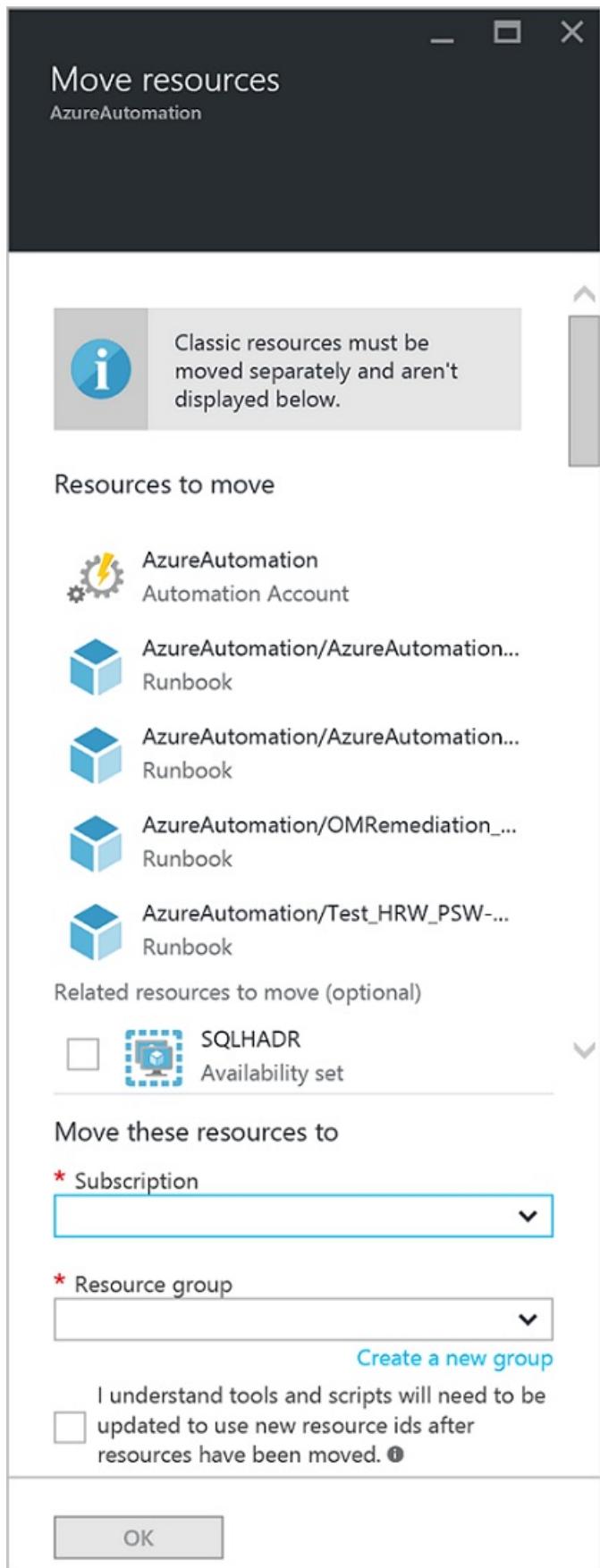
1. From your Automation account, click **Move** at the top of the page.



The screenshot shows the Azure portal interface for an Automation account named 'TestAzureAuto'. The top navigation bar includes 'Home > TestAzureAuto', a search bar, and buttons for 'Delete', 'Move' (which is highlighted with a red box), 'Feedback', and 'Refresh'. The main content area is titled 'Essentials' and displays the following information:

| Resource group (change) | Status |
|----------------------------|-------------------|
| TestAzureAuto | Active |
| Location | Last modified |
| East US 2 | 1/30/2018 3:51 PM |
| Subscription name (change) | |
| Microsoft Azure | |

2. Choose to move the automation account to another resource group, or another subscription.
3. On the **Move resources** pane, notice it presents resources related to both your Automation account and your resource group(s). Select the **subscription** and **resource group** from the drop-down lists, or select the option **create a new resource group** and enter a new resource group name in the field provided.
4. Review and select the checkbox to acknowledge you *understand tools and scripts will need to be updated to use new resource IDs after resources have been moved* and then click **OK**.



This action can take several minutes to complete. In **Notifications**, you are presented with a status of each action that takes place - validation, migration, and then finally when it is completed.

To move the Automation Account using PowerShell

To move existing Automation resources to another resource group or subscription, use the **Get-AzureRmResource** cmdlet to get the specific Automation account and then **Move-AzureRmResource** cmdlet to

perform the move.

The first example shows how to move an Automation account to a new resource group.

```
$resource = Get-AzureRmResource -ResourceName "TestAutomationAccount" -ResourceGroupName "ResourceGroup01"
Move-AzureRmResource -ResourceId $resource.ResourceId -DestinationResourceGroupName "NewResourceGroup"
```

After you execute the above code example, you are prompted to verify you want to perform this action. Once you click **Yes** and allow the script to proceed, you do not receive any notifications while it's performing the migration.

To move to a new subscription, include a value for the *DestinationSubscriptionId* parameter.

```
$resource = Get-AzureRmResource -ResourceName "TestAutomationAccount" -ResourceGroupName "ResourceGroup01"
Move-AzureRmResource -ResourceId $resource.ResourceId -DestinationResourceGroupName "NewResourceGroup" -
DestinationSubscriptionId "SubscriptionId"
```

As with the previous example, you are prompted to confirm the move.

Next steps

- For more information about moving resources to new resource group or subscription, see [Move resources to new resource group or subscription](#)
- For more information about Role-based Access Control in Azure Automation, see [Role-based access control in Azure Automation](#).
- To learn about PowerShell cmdlets for managing your subscription, see [Using Azure PowerShell with Resource Manager](#)
- To learn about portal features for managing your subscription, see [Using the Azure portal to manage resources](#).

Troubleshoot errors when onboarding solutions

6/27/2018 • 4 minutes to read • [Edit Online](#)

You may encounter errors when onboarding solutions like Update Management or Change Tracking and Inventory. This article describes the various errors that may occur and how to resolve them.

General Errors

Scenario: ComputerGroupQueryFormatError

Issue

This error code means that the saved search computer group query used to target the solution was not formatted correctly.

Cause

You may have altered the query, or it may have been altered by the system.

Resolution

You can delete the query for this solution, and reonboard the solution, which recreates the query. The query can be found within your workspace, under **Saved searches**. The name of the query is **MicrosoftDefaultComputerGroup**, and the category of the query is the name of the solution associated with this query. If multiple solutions are enabled, the **MicrosoftDefaultComputerGroup** shows multiple times under **Saved Searches**.

Scenario: PolicyViolation

Issue

This error code means that the deployment failed due to violation of one or more policies.

Cause

A policy is in place that is blocking the operation from completing.

Resolution

In order to successfully deploy the solution, you need to consider altering the indicated policy. As there are many different types of policies that can be defined, the specific changes required depend on the policy that is violated. For example, if a policy was defined on a resource group that denied permission to change the contents of certain types of resources within that resource group, you could, for example, do any of the following:

- Remove the policy altogether.
- Try to onboard to a different resource group.
- Revise the policy, by, for example:
 - Re-targeting the policy to a specific resource (such as to a specific Automation account).
 - Revising the set of resources that policy was configured to deny.

Check the notifications in the top right corner of the Azure portal or navigate to the resource group that contains your automation account and select **Deployments** under **Settings** to view the failed deployment. To learn more about Azure Policy visit: [Overview of Azure Policy](#).

MMA Extension failures

When deploying a solution, a variety of related resources are deployed. One of those resources is the Microsoft Monitoring Agent Extension or OMS Agent for Linux. These are Virtual Machine Extensions installed by the virtual machine's Guest Agent that is responsible for communicating with the configured Operations Management Suite (OMS) Workspace, for the purpose of later coordination of the downloading of binaries and other files that the

solution you are onboarding depend on once it begins execution. You typically first become aware of MMA or OMS Agent for Linux installation failures from a notification appearing in the Notifications Hub. Clicking on that notification gives further information about the specific failure. Navigation to the Resource Groups resource, and then to the Deployments element within it also provides details on the deployment failures that occurred. Installation of the MMA or OMS Agent for Linux can fail for a variety of reasons, and the steps to take to address these failures vary, depending on the issue. Specific troubleshooting steps follow.

The following section describes various issues that you can encounter when onboarding that cause a failure in the deployment of the MMA extension.

Scenario: An exception occurred during a WebClient request

The MMA extension on the virtual machine is unable to communicate with external resources and deployment fails.

Issue

The following are examples of error messages that are returned:

```
Please verify the VM has a running VM agent, and can establish outbound connections to Azure storage.
```

```
'Manifest download error from  
https://<endpoint>/<endpointId>/Microsoft.EnterpriseCloud.Monitoring_MicrosoftMonitoringAgent_australiaeast_manifest.xml. Error: UnknownError. An exception occurred during a WebClient request.'
```

Cause

Some potential causes to this error are:

- There is a proxy configured in the VM, that only allows specific ports.
- A firewall setting has blocked access to the required ports and addresses.

Resolution

Ensure that you have the proper ports and addresses open for communication. For a list of ports and addresses, see [planning your network](#).

Scenario: Install failed due to transient environment issues

The installation of the Microsoft Monitoring Agent extension failed during deployment due to another installation or action blocking the installation

Issue

The following are examples of error messages may be returned:

```
The Microsoft Monitoring Agent failed to install on this machine. Please try to uninstall and reinstall the extension. If the issue persists, please contact support.
```

```
'Install failed for plugin (name: Microsoft.EnterpriseCloud.Monitoring.MicrosoftMonitoringAgent, version 1.0.11081.4) with exception Command  
C:\Packages\Plugins\Microsoft.EnterpriseCloud.Monitoring.MicrosoftMonitoringAgent\1.0.11081.4\MMAExtensionInsta  
ll.exe of Microsoft.EnterpriseCloud.Monitoring.MicrosoftMonitoringAgent has exited with Exit code: 1618'
```

```
'Install failed for plugin (name: Microsoft.EnterpriseCloud.Monitoring.MicrosoftMonitoringAgent, version 1.0.11081.2) with exception Command  
C:\Packages\Plugins\Microsoft.EnterpriseCloud.Monitoring.MicrosoftMonitoringAgent\1.0.11081.2\MMAExtensionInsta  
ll.exe of Microsoft.EnterpriseCloud.Monitoring.MicrosoftMonitoringAgent has exited with Exit code: 1601'
```

Cause

Some potential causes to this error are:

- Another installation is in progress
- The system was triggered to reboot during template deployment

Resolution

This error is a transient error in nature. Retry the deployment to install the extension.

Scenario: Installation timeout

The installation of the MMA extension did not complete due to a timeout.

Issue

The following is an example of an error message that may be returned:

```
Install failed for plugin (name: Microsoft.EnterpriseCloud.Monitoring.MicrosoftMonitoringAgent, version 1.0.11081.4) with exception Command C:\Packages\Plugins\Microsoft.EnterpriseCloud.Monitoring.MicrosoftMonitoringAgent\1.0.11081.4\MMAExtensionInstaller.exe of Microsoft.EnterpriseCloud.Monitoring.MicrosoftMonitoringAgent has exited with Exit code: 15614
```

Cause

This error is due to the virtual machine being under a heavy load during installation.

Resolution

Attempt to install the MMA extension when the VM is under a lower load.

Next steps

If you did not see your problem or are unable to solve your issue, visit one of the following channels for more support:

- Get answers from Azure experts through [Azure Forums](#)
- Connect with [@AzureSupport](#) – the official Microsoft Azure account for improving customer experience by connecting the Azure community to the right resources: answers, support, and experts.
- If you need more help, you can file an Azure support incident. Go to the [Azure support site](#) and select **Get Support**.

Troubleshoot errors with runbooks

7/9/2018 • 8 minutes to read • [Edit Online](#)

Authentication errors when working with Azure Automation runbooks

Scenario: Sign in to Azure Account failed

Issue

You receive the following error when working with the `Add-AzureAccount` or `Connect-AzureRmAccount` cmdlets.:

```
Unknown_user_type: Unknown User Type
```

Cause

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

Resolution

In order to determine what's wrong, take the following steps:

1. 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.
2. Check that you can use the username and password that are stored in the Azure Automation credential in your local PowerShell ISE editor. You can do this by running the following cmdlets in the PowerShell ISE:

```
$Cred = Get-Credential  
#Using Azure Service Management  
Add-AzureAccount -Credential $Cred  
#Using Azure Resource Manager  
Connect-AzureRmAccount -Credential $Cred
```

3. If your authentication fails locally, this means that you haven't set up your Azure Active Directory credentials properly. Refer to [Authenticating to Azure using Azure Active Directory](#) blog post to get the Azure Active Directory account set up correctly.

Scenario: Unable to find the Azure subscription

Issue

You receive the following error when working with the `Select-AzureSubscription` or `Select-AzureRmSubscription` cmdlets.:

```
The subscription named <subscription name> cannot be found.
```

Error

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.

Resolution

In order to determine if you have properly authenticated to Azure and have access to the subscription you are trying to select, take the following steps:

1. Make sure that you run the `Add-AzureAccount` before running the `Select-AzureSubscription` cmdlet.
2. 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. Now verify if the output of `Get-AzureSubscription` contains your subscription details.

- If you don't see any subscription details in the output, this means that the subscription isn't initialized yet.
- If you do see the subscription details in the output, confirm that you are using the correct subscription name or ID with the **Select-AzureSubscription** cmdlet.

Scenario: Authentication to Azure failed because multi-factor authentication is enabled

Issue

You receive the the following error when authenticating to Azure with your Azure username and password:

```
Add-AzureAccount: AADSTS50079: Strong authentication enrollment (proof-up) is required
```

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.

Resolution

To use a certificate with the Azure classic deployment model cmdlets, refer to [creating and adding a certificate to manage Azure services](#). To use a service principal with Azure Resource Manager cmdlets, refer to [creating service principal using Azure portal](#) and [authenticating a service principal with Azure Resource Manager](#).

Common errors when working with runbooks

Scenario: The runbook job start was attempted three times, but it failed to start each time

Issue

Your runbook fails with the error:

```
The job was tried three times but it failed
```

Cause

This error can be caused by the following reasons:

1. Memory Limit. There are documented limits on how much memory allocated to a Sandbox [Automation service limits](#) so a job may fail if it is using more than 400 MB of memory.
2. Module Incompatible. This can occur if module dependencies are not correct and if they are not, your runbook typically returns a "Command not found" or "Cannot bind parameter" message.

Resolution

Any of the following solutions fix the problem:

- Suggested methods to work within the memory limit are to split the workload between multiple runbooks, not process as much data in memory, not to write unnecessary output from your runbooks, or consider how many checkpoints you write into your PowerShell workflow runbooks.
- Update your Azure modules by following the steps [How to update Azure PowerShell modules in Azure Automation](#).

Scenario: Runbook fails because of deserialized object

Issue

Your runbook fails with the error:

Cannot bind parameter <ParameterName>.

Cannot convert the <ParameterType> value of type Deserialized <ParameterType> to type <ParameterType>.

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.

Resolution

Any of the following three solutions fix this problem:

1. If you are piping complex objects from one cmdlet to another, wrap these cmdlets in an **InlineScript**.
2. Pass the name or value that you need from the complex object instead of passing the entire object.
3. Use a PowerShell runbook instead of a PowerShell Workflow runbook.

Scenario: Runbook job failed because the allocated quota exceeded

Issue

Your runbook job fails with the error:

```
The quota for the monthly total job run time has been reached for this subscription
```

Cause

This error occurs when the job execution exceeds the 500-minute free quota for your account. This quota 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. To learn more about pricing for Automation, see [Automation pricing](#).

Resolution

If you want to use more than 500 minutes of processing per month, you need to change your subscription from the Free tier to the Basic tier. You can upgrade to the Basic tier by taking the following steps:

1. Sign in to your Azure subscription
2. Select the Automation account you wish to upgrade
3. Click on **Settings > Pricing**.
4. Click **Enable** on page bottom to upgrade your account to the **Basic** tier.

Scenario: Cmdlet not recognized when executing a runbook

Issue

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.
```

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.

Resolution

Any of the following solutions fix the problem:

- Check that you have entered the cmdlet name correctly.
- Make sure the cmdlet exists in your Automation account and that there are no conflicts. To verify if the cmdlet is present, open a runbook in edit mode and search for the cmdlet you want to find in the library or run

`Get-Command <CommandName>`. Once you have validated that the cmdlet is available to the account, and that there are no name conflicts with other cmdlets or runbooks, add it to the canvas and ensure that you are using a valid parameter set in your runbook.

- 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. For example, you can use **ModuleName\CmdletName**.
- If you are executing the runbook on-premises in a hybrid worker group, then make sure that the module/cmdlet is installed on the machine that hosts the hybrid worker.

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

Issue

This behavior is by design due to the "Fair Share" monitoring of processes within Azure Automation, which automatically suspends a runbook if it executes longer than three hours. However, the error message returned does not provide "what next" options.

Cause

A runbook can be suspended for a number of reasons. Suspensions happen mostly due to errors. For example, an uncaught exception in a runbook, a network failure, or a crash on the Runbook Worker running the runbook, all cause the runbook to be suspended and start from its last checkpoint when resumed.

Resolution

The documented solution to avoid this issue is to use Checkpoints in a workflow. To learn more, refer to [Learning PowerShell Workflows](#). A more thorough explanation of "Fair Share" and Checkpoint can be found in this blog article [Using Checkpoints in Runbooks](#).

Scenario: A long running runbook fails to complete

Issue

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

Cause

The runbook ran over the 3 hour limit allowed by fair share in an Azure Sandbox

Resolution

The recommended solution is to run the runbook on a [Hybrid Runbook Worker](#). Hybrid Workers are not limited by the [fair share](#) 3 hour runbook limit that Azure sandboxes are.

Common errors when importing modules

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

Issue

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

Cause

Some common reasons that a module might not successfully import to Azure Automation are:

- The structure does not match the structure that Automation needs it to be in.
- The module is dependent on another module that has not been deployed to your Automation account.
- The module is missing its dependencies in the folder.
- The `New-AzureRmAutomationModule` cmdlet is being used to upload the module, and you have not given the full storage path or have not loaded the module by using a publicly accessible URL.

Resolution

Any of the following solutions fix the problem:

- Make sure that the module follows the following format: `ModuleName.Zip -> ModuleName` or `Version`

Number -> (ModuleName.psm1, ModuleName.psd1)

- Open the .psd1 file and see if the module has any dependencies. If it does, upload these modules to the Automation account.
- Make sure that any referenced .dlls are present in the module folder.

Next steps

If you did not see your problem or are unable to solve your issue, visit one of the following channels for more support:

- Get answers from Azure experts through [Azure Forums](#)
- Connect with [@AzureSupport](#) – the official Microsoft Azure account for improving customer experience by connecting the Azure community to the right resources: answers, support, and experts.
- If you need more help, you can file an Azure support incident. Go to the [Azure support site](#) and select **Get Support**.

Troubleshoot Desired State Configuration (DSC)

6/27/2018 • 2 minutes to read • [Edit Online](#)

This article provides information on troubleshooting issues with Desired State Configuration (DSC).

Common errors when working with Desired State Configuration (DSC)

Scenario: Node is in failed status with a "Not found" error

Issue

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.
```

Cause

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

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.
 - In order to assign a node configuration to a node using Azure portal, open the **DSC Nodes** page, then select a node and click on **Assign node configuration** button.
 - In order to assign a node configuration to a node using PowerShell cmdlet, use **Set-AzureRmAutomationDscNode** cmdlet

Scenario: No node configurations (MOF files) were produced when a configuration is compiled

Issue

Your DSC compilation job suspends with the error:

```
Compilation completed successfully, but no node configuration.mofs were generated.
```

Cause

When the expression following the **Node** keyword in the DSC configuration evaluates to `$null`, then no node configurations are produced.

Resolution

Any of the following solutions fix the problem:

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

Scenario: The DSC node report becomes stuck "in progress" state

Issue

The DSC agent outputs:

No instance found with given property values

Cause

You have upgraded your WMF version and have corrupted WMI.

Resolution

To fix the issue follow the instructions in the [DSC known issues and limitations](#) article.

Scenario: Unable to use a credential in a DSC configuration

Issue

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.
```

Cause

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

Resolution

- Make sure to pass in the proper **ConfigurationData** to set **PSDscAllowPlainTextPassword** to true for each node configuration mentioned in the configuration. For more information, see [assets in Azure Automation DSC](#).

Next steps

If you did not see your problem or are unable to solve your issue, visit one of the following channels for more support:

- Get answers from Azure experts through [Azure Forums](#)
- Connect with [@AzureSupport](#) – the official Microsoft Azure account for improving customer experience by connecting the Azure community to the right resources: answers, support, and experts.
- If you need more help, you can file an Azure support incident. Go to the [Azure support site](#) and select **Get Support**.

Troubleshoot Hybrid Runbook Workers

6/27/2018 • 5 minutes to read • [Edit Online](#)

This article provides information on troubleshooting issues with Hybrid Runbook Workers.

General

The Hybrid Runbook Worker depends on an agent to communicate with your Automation account to register the worker, receive runbook jobs, and report status. For Windows, this agent is the Microsoft Monitoring Agent. For Linux, it is the OMS Agent for Linux.

Scenario: Runbook execution fails

Issue

Runbook execution fails and you receive the following error:

```
"The job action 'Activate' cannot be run, because the process stopped unexpectedly. The job action was attempted three times."
```

Your runbook is suspended shortly after attempting to execute it three times. There are conditions, which may interrupt the runbook from completing successfully and the related error message does not include any additional information indicating why.

Cause

The following are potential possible causes:

- The runbooks can't authenticate with local resources
- The hybrid worker is behind a proxy or firewall
- The runbooks can't authenticate with local resources
- The computer designated to run the Hybrid Runbook Worker feature meets the minimum hardware requirements.

Resolution

Verify the computer has outbound access to *.azure-automation.net on port 443.

Computers running the Hybrid Runbook Worker should meet the minimum hardware requirements before designating it to host this feature. Otherwise, depending on the resource utilization of other background processes and contention caused by runbooks during execution, the computer becomes over utilized and cause runbook job delays or timeouts.

Confirm the computer designated to run the Hybrid Runbook Worker feature meets the minimum hardware requirements. If it does, monitor CPU and memory utilization to determine any correlation between the performance of Hybrid Runbook Worker processes and Windows. If there is memory or CPU pressure, this may indicate the need to upgrade or add additional processors, or increase memory to address the resource bottleneck and resolve the error. Alternatively, select a different compute resource that can support the minimum requirements and scale when workload demands indicate an increase is necessary.

Check the **Microsoft-SMA** event log for a corresponding event with description *Win32 Process Exited with code [4294967295]*. The cause of this error is you haven't configured authentication in your runbooks or specified the Run As credentials for the Hybrid worker group. Review [Runbook permissions](#) to confirm you have correctly configured authentication for your runbooks.

Linux

The Linux Hybrid Runbook Worker depends on the OMS Agent for Linux to communicate with your Automation account to register the worker, receive runbook jobs, and report status. If registration of the worker fails, here are some possible causes for the error:

Scenario: The OMS Agent for Linux is not running

If the OMS Agent for Linux is not running, this prevents the Linux Hybrid Runbook Worker from communicating with Azure Automation. Verify the agent is running by entering the following command: `ps -ef | grep python`. You should see output similar to the following, the python processes with **nxautomation** user account. If the Update Management or Azure Automation solutions are not enabled, none of the following processes are running.

```
nxautom+ 8567      1  0 14:45 ?          00:00:00 python
/opt/microsoft/omsconfig/modules/nxOMSAutomationWorker/DSCResources/MSFT_nxOMSAutomationWorkerResource/automationworker/worker/main.py /var/opt/microsoft/omsagent/state/automationworker/oms.conf rworkspace:<workspaceId>
<Linux hybrid worker version>
nxautom+ 8593      1  0 14:45 ?          00:00:02 python
/opt/microsoft/omsconfig/modules/nxOMSAutomationWorker/DSCResources/MSFT_nxOMSAutomationWorkerResource/automationworker/worker/hybridworker.py /var/opt/microsoft/omsagent/state/automationworker/worker.conf managed
rworkspace:<workspaceId> rversion:<Linux hybrid worker version>
nxautom+ 8595      1  0 14:45 ?          00:00:02 python
/opt/microsoft/omsconfig/modules/nxOMSAutomationWorker/DSCResources/MSFT_nxOMSAutomationWorkerResource/automationworker/worker/hybridworker.py
/var/opt/microsoft/omsagent/<workspaceId>/state/automationworker/diy/worker.conf managed rworkspace:
<workspaceId> rversion:<Linux hybrid worker version>
```

The following list shows the processes that are started for a Linux Hybrid Runbook Worker. They are all located in the `/var/opt/microsoft/omsagent/state/automationworker/` directory.

- **oms.conf** - This is the worker manager process, this is started directly from DSC.
- **worker.conf** - This process is the Auto Registered Hybrid worker process, it is started by the worker manager. This process is used by Update Management and is transparent to the user. This process is not present if the Update Management solution is not enabled on the machine.
- **diy/worker.conf** - This process is the DIY hybrid worker process. The DIY hybrid worker process is used to execute user runbooks on the Hybrid Runbook Worker. It only differs from the Auto registered Hybrid worker process in the key detail that it uses a different configuration. This process is not present if the Azure Automation solution is not enabled, and the DIY Linux Hybrid Worker is not registered.

If the OMS Agent for Linux is not running, run the following command to start the service:

```
sudo /opt/microsoft/omsagent/bin/service_control restart .
```

Scenario: The specified class does not exist

If you see the error: **The specified class does not exist..** in the `/var/opt/microsoft/omsconfig/omsconfig.log` then the OMS Agent for Linux needs to be updated. Run the following command to reinstall the OMS Agent:

```
wget https://raw.githubusercontent.com/Microsoft/OMS-Agent-for-Linux/master/installer/scripts/onboard_agent.sh
&& sh onboard_agent.sh -w <WorkspaceID> -s <WorkspaceKey>
```

Windows

The Windows Hybrid Runbook Worker depends on the Microsoft Monitoring Agent to communicate with your Automation account to register the worker, receive runbook jobs, and report status. If registration of the worker fails, here are some possible causes for the error:

Scenario: The Microsoft Monitoring Agent is not running

Issue

The `healthservice` service is not running on the Hybrid Runbook Worker machine.

Cause

If the Microsoft Monitoring Agent Windows service is not running, this prevents the Hybrid Runbook Worker from communicating with Azure Automation.

Resolution

Verify the agent is running by entering the following command in PowerShell: `Get-Service healthservice`. If the service is stopped, enter the following command in PowerShell to start the service: `Start-Service healthservice`.

Event 4502 in Operations Manager log

Issue

In the **Application and Services Logs\Operations Manager** event log, you see event 4502 and EventMessage containing **Microsoft.EnterpriseManagement.HealthService.AzureAutomation.HybridAgent** with the following description: *The certificate presented by the service <wsid>.oms.opinsights.azure.com was not issued by a certificate authority used for Microsoft services. Please contact your network administrator to see if they are running a proxy that intercepts TLS/SSL communication. The article KB3126513 has additional troubleshooting information for connectivity issues.*

Cause

This can be caused by your proxy or network firewall blocking communication to Microsoft Azure. Verify the computer has outbound access to *.azure-automation.net on ports 443.

Resolution

Logs are stored locally on each hybrid worker at C:\ProgramData\Microsoft\System Center\Orchestrator\7.2\SMA\Sandboxes. You can check if there are any warning or error events written to the **Application and Services Logs\Microsoft-SMA\Operations and Application and Services Logs\Operations Manager** event log that would indicate a connectivity or other issue affecting onboarding of the role to Azure Automation or issue while performing normal operations.

[Runbook output and messages](#) are sent to Azure Automation from hybrid workers just like runbook jobs run in the cloud. You can also enable the Verbose and Progress streams the same way you would for other runbooks.

Next steps

If you did not see your problem or are unable to solve your issue, visit one of the following channels for more support:

- Get answers from Azure experts through [Azure Forums](#)
- Connect with [@AzureSupport](#) – the official Microsoft Azure account for improving customer experience by connecting the Azure community to the right resources: answers, support, and experts.
- If you need more help, you can file an Azure support incident. Go to the [Azure support site](#) and select **Get Support**.

Troubleshooting issues with Update Management

6/27/2018 • 3 minutes to read • [Edit Online](#)

This article discusses solutions to resolve issues that you may encounter when using Update Management.

Windows

If you encounter issues while attempting to onboard the solution on a virtual machine, check the **Operations Manager** event log under **Application and Services Logs** on the local machine for events with event ID **4502** and event message containing **Microsoft.EnterpriseManagement.HealthService.AzureAutomation.HybridAgent**.

The following section highlights specific error messages and a possible resolution for each. For other onboarding issues see, [troubleshoot solution onboarding](#).

Scenario: Machine is already registered to a different account

Issue

You receive the following error message:

```
Unable to Register Machine for Patch Management, Registration Failed with Exception  
System.InvalidOperationException: {"Message":"Machine is already registered to a different account."}
```

Cause

The machine is already onboarded to another workspace for Update Management.

Resolution

Perform cleanup of old artifacts on the machine by [deleting the hybrid runbook group](#) and try again.

Scenario: Machine is unable to communicate with the service

Issue

You receive one of the following error messages:

```
Unable to Register Machine for Patch Management, Registration Failed with Exception  
System.Net.Http.HttpRequestException: An error occurred while sending the request. --->  
System.Net.WebException: The underlying connection was closed: An unexpected error occurred on a receive. --->  
System.ComponentModel.Win32Exception: The client and server can't communicate, because they do not possess a  
common algorithm
```

```
Unable to Register Machine for Patch Management, Registration Failed with Exception  
Newtonsoft.Json.JsonReaderException: Error parsing positive infinity value.
```

```
The certificate presented by the service <wsid>.oms.opinsights.azure.com was not issued by a certificate  
authority used for Microsoft services. Contact your network administrator to see if they are running a proxy  
that intercepts TLS/SSL communication.
```

Cause

There may be a proxy, gateway or firewall blocking network communication.

Resolution

Review your networking and ensure appropriate ports and addresses are allowed. See [network requirements](#), for a list of ports and addresses that are required by Update Management and Hybrid Runbook Workers.

Scenario: Unable to create self-signed certificate

Issue

You receive one of the following error messages:

```
Unable to Register Machine for Patch Management, Registration Failed with Exception  
AgentService.HybridRegistration. PowerShell.Certificates.CertificateCreationException: Failed to create a  
self-signed certificate. ---> System.UnauthorizedAccessException: Access is denied.
```

Cause

The Hybrid Runbook Worker was not able to generate a self-signed certificate

Resolution

Verify system account has read access to folder **C:\ProgramData\Microsoft\Crypto\RSA** and try again.

Linux

Scenario: Update run fails to start

Issue

An update runs fail to start on a Linux machine.

Cause

The Linux Hybrid Worker is unhealthy.

Resolution

Make a copy of the following log file and preserve it for troubleshooting purposes:

```
/var/opt/microsoft/omsagent/run/automationworker/worker.log
```

Scenario: Update run starts, but encounters errors

Issue

An update run starts, but encounters errors during the run.

Cause

Possible causes could be:

- Package manager is unhealthy
- Specific packages may interfere with cloud based patching
- Other reasons

Resolution

If failures occur during an update run after it starts successfully on Linux, check the job output from the affected machine in the run. You may find specific error messages from your machine's package manager that you can research and take action on. Update Management requires the package manager to be healthy for successful update deployments.

In some cases, package updates can interfere with Update Management preventing an update deployment from completing. If you see that, you'll have to either exclude these packages from future update runs or install them manually yourself.

If you can't resolve a patching issue, make a copy of the following log file and preserve it **before** the next update deployment starts for troubleshooting purposes:

```
/var/opt/microsoft/omsagent/run/automationworker/omsupdategmt.log
```

Next steps

If you did not see your problem or are unable to solve your issue, visit one of the following channels for more support:

- Get answers from Azure experts through [Azure Forums](#)
- Connect with [@AzureSupport](#) – the official Microsoft Azure account for improving customer experience by connecting the Azure community to the right resources: answers, support, and experts.
- If you need more help, you can file an Azure support incident. Go to the [Azure support site](#) and select **Get Support**.