
Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.
Table of Contents

What are the AWS Tools for PowerShell? ................................................................. 1
  Maintenance and support for SDK major versions .............................................. 2
  AWS.Tools .............................................................................................................. 2
  AWSPowerShell.NetCore ...................................................................................... 3
  AWSPowerShell ..................................................................................................... 3
  How to use this guide ............................................................................................ 4

Installation ................................................................................................................. 5
  Installing on Windows .............................................................................................. 5
    Prerequisites ......................................................................................................... 6
    Install AWS.Tools .................................................................................................. 6
    Install AWSPowerShell.NetCore .......................................................................... 8
    Install AWSPowerShell ......................................................................................... 10
    Enable Script Execution ....................................................................................... 10
    Versioning ............................................................................................................. 12
  Updating AWS Tools for PowerShell ....................................................................... 14

Installing on Linux or macOS ...................................................................................... 15
  Overview of Setup .................................................................................................. 15
    Prerequisites ......................................................................................................... 6
    Install AWS.Tools .................................................................................................. 16
    Install AWSPowerShell.NetCore .......................................................................... 19
    Script Execution .................................................................................................... 10
    Configuring the PowerShell Console .................................................................... 21
    Initialize Your PowerShell Session ...................................................................... 21
    Versioning ............................................................................................................. 12
  Updating the AWS Tools for PowerShell on Linux or macOS ................................ 23
  Related Information ............................................................................................... 24

Migrating from AWS Tools for PowerShell Version 3.3 to Version 4 ......................... 24
  New Fully Modularized AWS.Tools Version ......................................................... 24
  New Get-AWSService cmdlet .................................................................................. 25
  New -Select Parameter to Control the Object Returned by a Cmdlet .................... 25
  More Consistent Limiting of the Number of Items in the Output ......................... 27
  Easier to Use Stream Parameters .......................................................................... 28
  Extending the Pipe by Property Name .................................................................... 28
  Static Common Parameters ..................................................................................... 29
Get started .......................................................................................................................... 31

Configure tool authentication ......................................................................................... 31
  Enable and configure IAM Identity Center ................................................................. 32
  Configure the Tools for PowerShell to use IAM Identity Center ............................ 32
  Start an AWS access portal session .......................................................................... 34
  Example ....................................................................................................................... 35
  Additional information ............................................................................................... 35
  Use the AWS CLI ......................................................................................................... 36

Specify AWS Regions ....................................................................................................... 40
  Specifying a Custom or Nonstandard Endpoint ....................................................... 41
  Additional information ............................................................................................... 42

Configure federated identity ........................................................................................... 42
  Prerequisites ............................................................................................................... 42
  How an Identity-Federated User Gets Federated Access to AWS Service APIs ....... 43
  How SAML Support Works in the AWS Tools for PowerShell .............................. 44
  How to Use the PowerShell SAML Configuration Cmdlets ..................................... 45
  Additional Reading .................................................................................................... 50

Cmdlet discovery and aliases .......................................................................................... 50
  Cmdlet Discovery ........................................................................................................ 50
  Cmdlet Naming and Aliases ....................................................................................... 56

Pipelining and $AWSHistory ......................................................................................... 61
  $AWSHistory ............................................................................................................ 61

Credential and profile resolution .................................................................................... 65
  Credentials Search Order ........................................................................................... 65

Users and roles ................................................................................................................ 66
  Users and permission sets ......................................................................................... 66
  Service roles ............................................................................................................... 67

Using legacy credentials ............................................................................................... 68
  Important warnings and guidelines ......................................................................... 68
  AWS Credentials ...................................................................................................... 69
  Shared Credentials ................................................................................................... 78

Work with AWS services ............................................................................................... 84
  PowerShell File Concatenation Encoding ................................................................ 84
Returned Objects for the PowerShell Tools ................................................................. 85
Amazon EC2 ............................................................................................................. 85
Amazon S3 ............................................................................................................... 85
AWS Lambda and AWS Tools for PowerShell ......................................................... 86
Amazon SNS and Amazon SQS ............................................................................... 86
CloudWatch ............................................................................................................. 86
See Also .................................................................................................................. 86
Topics ....................................................................................................................... 86
Amazon S3 and Tools for Windows PowerShell ..................................................... 87
  Create an Amazon S3 Bucket, Verify Its Region, and Optionally Remove It .......... 88
  Configure an Amazon S3 Bucket as a Website and Enable Logging .................. 89
  Upload Objects to an Amazon S3 Bucket ............................................................. 89
  Delete Amazon S3 Objects and Buckets ............................................................. 91
  Upload In-Line Text Content to Amazon S3 ....................................................... 93
Amazon EC2 and Tools for Windows PowerShell .................................................. 93
  Create a Key Pair ................................................................................................. 94
  Create a Security Group ...................................................................................... 96
  Find an AMI ......................................................................................................... 100
  Launch an Instance .............................................................................................. 104
AWS Lambda and AWS Tools for PowerShell ....................................................... 108
  Prerequisites ....................................................................................................... 6
  Install the AWSLambdaPSCore Module ............................................................. 109
See Also .................................................................................................................. 86
Amazon SQS, Amazon SNS and Tools for Windows PowerShell ......................... 110
  Create an Amazon SQS queue and get queue ARN ......................................... 110
  Create an Amazon SNS topic ............................................................................. 110
  Give permissions to the SNS topic ..................................................................... 111
  Subscribe the queue to the SNS topic ............................................................... 111
  Give permissions ............................................................................................... 112
  Verify results ...................................................................................................... 112
CloudWatch from the AWS Tools for Windows PowerShell ................................. 113
  Publish a Custom Metric to Your CloudWatch Dashboard ............................... 113
See Also .................................................................................................................. 86
Using ClientConfig ................................................................................................. 114
  Using the ClientConfig parameter ..................................................................... 114
  Using an undefined property ............................................................................. 115
<table>
<thead>
<tr>
<th>Code examples</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actions and scenarios</td>
<td>117</td>
</tr>
<tr>
<td>ACM</td>
<td>119</td>
</tr>
<tr>
<td>Application Auto Scaling</td>
<td>124</td>
</tr>
<tr>
<td>AppStream 2.0</td>
<td>130</td>
</tr>
<tr>
<td>Aurora</td>
<td>156</td>
</tr>
<tr>
<td>Auto Scaling</td>
<td>158</td>
</tr>
<tr>
<td>AWS Budgets</td>
<td>193</td>
</tr>
<tr>
<td>AWS Cloud9</td>
<td>194</td>
</tr>
<tr>
<td>AWS CloudFormation</td>
<td>201</td>
</tr>
<tr>
<td>CloudFront</td>
<td>214</td>
</tr>
<tr>
<td>CloudTrail</td>
<td>222</td>
</tr>
<tr>
<td>CloudWatch</td>
<td>227</td>
</tr>
<tr>
<td>CodeCommit</td>
<td>231</td>
</tr>
<tr>
<td>CodeDeploy</td>
<td>237</td>
</tr>
<tr>
<td>CodePipeline</td>
<td>255</td>
</tr>
<tr>
<td>Amazon Cognito Identity</td>
<td>273</td>
</tr>
<tr>
<td>AWS Config</td>
<td>277</td>
</tr>
<tr>
<td>Device Farm</td>
<td>295</td>
</tr>
<tr>
<td>AWS Directory Service</td>
<td>296</td>
</tr>
<tr>
<td>AWS DMS</td>
<td>321</td>
</tr>
<tr>
<td>DynamoDB</td>
<td>322</td>
</tr>
<tr>
<td>Amazon EC2</td>
<td>337</td>
</tr>
<tr>
<td>Amazon ECR</td>
<td>464</td>
</tr>
<tr>
<td>Amazon ECS</td>
<td>465</td>
</tr>
<tr>
<td>Amazon EFS</td>
<td>471</td>
</tr>
<tr>
<td>Amazon EKS</td>
<td>478</td>
</tr>
<tr>
<td>Elastic Load Balancing - Version 1</td>
<td>491</td>
</tr>
<tr>
<td>Elastic Load Balancing - Version 2</td>
<td>510</td>
</tr>
<tr>
<td>Amazon FSx</td>
<td>534</td>
</tr>
<tr>
<td>AWS Glue</td>
<td>542</td>
</tr>
<tr>
<td>AWS Health</td>
<td>543</td>
</tr>
<tr>
<td>IAM</td>
<td>545</td>
</tr>
<tr>
<td>Kinesis</td>
<td>617</td>
</tr>
<tr>
<td>Lambda</td>
<td>620</td>
</tr>
</tbody>
</table>
What are the AWS Tools for PowerShell?

The AWS Tools for PowerShell are a set of PowerShell modules that are built on the functionality exposed by the AWS SDK for .NET. The AWS Tools for PowerShell enable you to script operations on your AWS resources from the PowerShell command line.

The cmdlets provide an idiomatic PowerShell experience for specifying parameters and handling results even though they are implemented using the various AWS service HTTP query APIs. For example, the cmdlets for the AWS Tools for PowerShell support PowerShell pipelining—that is, you can pipe PowerShell objects in and out of the cmdlets.

The AWS Tools for PowerShell are flexible in how they enable you to handle credentials, including support for the AWS Identity and Access Management (IAM) infrastructure. You can use the tools with IAM user credentials, temporary security tokens, and IAM roles.

The AWS Tools for PowerShell support the same set of services and AWS Regions that are supported by the SDK. You can install the AWS Tools for PowerShell on computers running Windows, Linux, or macOS operating systems.

Note

AWS Tools for PowerShell version 4 is the latest major release, and is a backward-compatible update to AWS Tools for PowerShell version 3.3. It adds significant improvements while maintaining existing cmdlet behavior. Your existing scripts should continue to work after upgrading to the new version, but we do recommend that you test them thoroughly before upgrading. For more information about the changes in version 4, see Migrating from AWS Tools for PowerShell Version 3.3 to Version 4.

The AWS Tools for PowerShell are available as the following three distinct packages:

- [AWS.Tools](#)
- [AWSPowerShell.NetCore](#)
- [AWSPowerShell](#)
Maintenance and support for SDK major versions

For information about maintenance and support for SDK major versions and their underlying dependencies, see the following in the AWS SDKs and Tools Reference Guide:

- AWS SDKs and tools maintenance policy
- AWS SDKs and tools version support matrix

AWS.Tools - A modularized version of the AWS Tools for PowerShell

This version of AWS Tools for PowerShell is the recommended version for any computer running PowerShell in a production environment. Because it's modularized, you need to download and load only the modules for the services you want to use. This reduces download times, memory usage, and, in most cases, enables auto-importing of AWS.Tools cmdlets without the need to manually call Import-Module first.

This is the latest version of AWS Tools for PowerShell and runs on all supported operating systems, including Windows, Linux, and macOS. This package provides one installation module, AWS.Tools.Installer, one common module, AWS.Tools.Common, and one module for each AWS service, for example, AWS.Tools.EC2, AWS.Tools.IdentityManagement, AWS.Tools.S3, and so on.

The AWS.Tools.Installer module provides cmdlets that enable you to install, update, and remove the modules for each of the AWS services. The cmdlets in this module automatically ensure that you have all the dependent modules required to support the modules you want to use.

The AWS.Tools.Common module provides cmdlets for configuration and authentication that are not service specific. To use the cmdlets for an AWS service, you just run the command. PowerShell automatically imports the AWS.Tools.Common module and the module for the AWS service whose cmdlet you want to run. This module is automatically installed if you use the AWS.Tools.Installer module to install the service modules.
You can install this version of AWS Tools for PowerShell on computers that are running:

- PowerShell Core 6.0 or later on Windows, Linux, or macOS.
- Windows PowerShell 5.1 or later on Windows with the .NET Framework 4.7.2 or later.

Throughout this guide, when we need to specify this version only, we refer to it by its module name: **AWS.Tools**.

**AWSPowerShell.NetCore - A single-module version of the AWS Tools for PowerShell**

This version consists of a single, large module that contains support for all AWS services. Before you can use this module, you must manually import it.

You can install this version of AWS Tools for PowerShell on computers that are running:

- PowerShell Core 6.0 or later on Windows, Linux, or macOS.
- Windows PowerShell 3.0 or later on Windows with the .NET Framework 4.7.2 or later.

Throughout this guide, when we need to specify this version only, we refer to it by its module name: **AWSPowerShell.NetCore**.

**AWSPowerShell - A single-module version for Windows PowerShell**

This version of AWS Tools for PowerShell is compatible with and installable on only Windows computers that are running Windows PowerShell versions 2.0 through 5.1. It is not compatible with PowerShell Core 6.0 or later, or any other operating system (Linux or macOS). This version consists of a single, large module that contains support for all AWS services.
Throughout this guide, when we need to specify this version only, we refer to it by its module name: \textit{AWSPowerShell}.

\section*{How to use this guide}

The guide is divided into the following major sections.

\textbf{Installing the AWS Tools for PowerShell}

This section explains how to install the AWS Tools for PowerShell. It includes how to sign up for AWS if you don't already have an account, and how to create an IAM user that you can use to run the cmdlets.

\textbf{Get started with the AWS Tools for Windows PowerShell}

This section describes the fundamentals of using the AWS Tools for PowerShell, such as specifying credentials and AWS Regions, finding cmdlets for a particular service, and using aliases for cmdlets.

\textbf{Work with AWS services in the AWS Tools for PowerShell}

This section includes information about using the AWS Tools for PowerShell to perform some of the most common AWS tasks.
Installing the AWS Tools for PowerShell

To successfully install and use the AWS Tools for PowerShell cmdlets, see the steps in the following topics.

Topics

• Installing the AWS Tools for PowerShell on Windows
• Installing AWS Tools for PowerShell on Linux or macOS
• Migrating from AWS Tools for PowerShell Version 3.3 to Version 4

Installing the AWS Tools for PowerShell on Windows

A Windows-based computer can run any of the AWS Tools for PowerShell package options:

• **AWS.Tools** - The modularized version of AWS Tools for PowerShell. Each AWS service is supported by its own individual, small module, with shared support modules `AWS.Tools.Common` and `AWS.Tools.Installer`.

• **AWS.PowerShell.NetCore** - The single, large-module version of AWS Tools for PowerShell. All AWS services are supported by this single, large module.

  ⚠️ **Note**

  Be aware that the single module might be too large to use with **AWS Lambda** functions. Instead, use the modularized version shown above.

• **AWS.PowerShell** - The legacy Windows-specific, single, large-module version of AWS Tools for PowerShell. All AWS services are supported by this single, large module.

The package you choose depends on the release and edition of Windows that you're running.

⚠️ **Note**

The Tools for Windows PowerShell (AWS.PowerShell module) are installed by default on all Windows-based Amazon Machine Images (AMIs).
Setting up the AWS Tools for PowerShell involves the following high-level tasks, described in detail in this topic.

1. Install the AWS Tools for PowerShell package option that's appropriate for your environment.
2. Verify that script execution is enabled by running the Get-ExecutionPolicy cmdlet.
3. Import the AWS Tools for PowerShell module into your PowerShell session.

**Prerequisites**

Newer versions of PowerShell, including PowerShell Core, are available as downloads from Microsoft at [Installing various versions of PowerShell](#) on Microsoft's Web site.

**Install AWS . Tools on Windows**

You can install the modularized version of AWS Tools for PowerShell on computers that are running Windows with Windows PowerShell 5.1, or PowerShell Core 6.0 or later. For information about how to install PowerShell Core, see [Installing various versions of PowerShell](#) on Microsoft's Web site.

You can install AWS . Tools in one of three ways:

- **Using the cmdlets in the AWS . Tools . Installer module.** This module simplifies the installation and update of other AWS . Tools modules. AWS . Tools . Installer requires PowerShellGet, and automatically downloads and installs an updated version of it. AWS . Tools . Installer automatically keeps your module versions in sync. When you install or update to a newer version of one module, the cmdlets in AWS . Tools . Installer automatically update all of your other AWS . Tools modules to the same version.

  This method is described in the procedure that follows.

- **Downloading the modules from AWS . Tools.zip and extracting them in one of the module folders.** You can discover your module folders by displaying the value of the PSMODULEPATH environment variable.

- **Installing each service module from the PowerShell Gallery using the Install-Module cmdlet.**

**To install AWS . Tools on Windows using the AWS . Tools . Installer module**

1. Start a PowerShell session.
Note

We recommend that you don’t run PowerShell as an administrator with elevated permissions except when required by the task at hand. This is because of the potential security risk and is inconsistent with the principle of least privilege.

2. To install the modularized AWS.Tools package, run the following command.

```
PS > Install-Module -Name AWS.Tools.Installer
```

Untrusted repository
You are installing the modules from an untrusted repository. If you trust this repository, change its InstallationPolicy value by running the Set-PSRepository cmdlet. Are you sure you want to install the modules from 'PSGallery'? [Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "N"): y

If you are notified that the repository is "untrusted", it asks you if you want to install anyway. Enter y to allow PowerShell to install the module. To avoid the prompt and install the module without trusting the repository, you can run the command with the -Force parameter.

```
PS > Install-Module -Name AWS.Tools.Installer -Force
```

3. You can now install the module for each AWS service that you want to use by using the Install-AWSToolsModule cmdlet. For example, the following command installs the Amazon EC2 and Amazon S3 modules. This command also installs any dependent modules that are required for the specified module to work. For example, when you install your first AWS.Tools service module, it also installs AWS.Tools.Common. This is a shared module required by all AWS service modules. It also removes older versions of the modules, and updates other modules to the same newer version.

```
PS > Install-AWSToolsModule AWS.Tools.EC2,AWS.Tools.S3 -CleanUp
```

Confirm
Are you sure you want to perform this action? Performing the operation "Install-AWSToolsModule" on target "AWS Tools version 4.0.0.0". [Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): y

Install AWS.Tools
Installing module AWS.Tools.Common version 4.0.0.0
Installing module AWS.Tools.EC2 version 4.0.0.0
Installing module AWS.Tools.Glacier version 4.0.0.0
Installing module AWS.Tools.S3 version 4.0.0.0

Uninstalling AWS.Tools version 3.3.618.0
Uninstalling module AWS.Tools.Glacier
Uninstalling module AWS.Tools.S3
Uninstalling module AWS.Tools.SimpleNotificationService
Uninstalling module AWS.Tools.SQS
Uninstalling module AWS.Tools.Common

**Note**

The `Install-AWSToolsModule` cmdlet downloads all requested modules from the PSRepository named PSGallery ([https://www.powershellgallery.com/](https://www.powershellgallery.com/)) and considers it a trusted source. Use the command `Get-PSRepository -Name PSGallery` for more information about this PSRepository.

By default, the previous command installs modules into the `%USERPROFILE%\Documents \WindowsPowerShell\Modules` folder. To install the AWS Tools for PowerShell for all users of a computer, you must run the following command in a PowerShell session that you started as an administrator. For example, the following command installs the IAM module to the `%ProgramFiles%\WindowsPowerShell\Modules` folder that is accessible by all users.

```powershell
PS > Install-AWSToolsModule AWS.Tools.IdentityManagement -Scope AllUsers
```

To install other modules, run similar commands with the appropriate module names, as found in the [PowerShell Gallery](https://www.powershellgallery.com/).

**Install AWSPowerShell.NetCore on Windows**

You can install the AWSPowerShell.NetCore on computers that are running Windows with PowerShell version 3 through 5.1, or PowerShell Core 6.0 or later. For information about how to install PowerShell Core, see [Installing various versions of PowerShell](https://docs.microsoft.com/en-us/powershell/scripting/install/installing-various-versions-of-powershell) on the Microsoft PowerShell website.
You can install AWSPowerShell.NetCore in one of two ways

- Downloading the module from AWSPowerShell.NetCore.zip and extracting it in one of the module directories. You can discover your module directories by displaying the value of the PSMODULEPATH environment variable.

- Installing from the PowerShell Gallery using the Install-Module cmdlet, as described in the following procedure.

**To install AWSPowerShell.NetCore from the PowerShell Gallery using the Install-Module cmdlet**

To install the AWSPowerShell.NetCore from the PowerShell Gallery, your computer must be running PowerShell 5.0 or later, or running PowerShellGet on PowerShell 3 or later. Run the following command.

```
PS > Install-Module -name AWSPowerShell.NetCore
```

If you're running PowerShell as administrator, the previous command installs AWS Tools for PowerShell for all users on the computer. If you're running PowerShell as a standard user without administrator permissions, that same command installs AWS Tools for PowerShell for only the current user.

To install for only the current user when that user has administrator permissions, run the command with the -Scope CurrentUser parameter set, as follows.

```
PS > Install-Module -name AWSPowerShell.NetCore -Scope CurrentUser
```

Although PowerShell 3.0 and later releases typically load modules into your PowerShell session the first time you run a cmdlet in the module, the AWSPowerShell.NetCore module is too large to support this functionality. You must instead explicitly load the AWSPowerShell.NetCore Core module into your PowerShell session by running the following command.

```
PS > Import-Module AWSPowerShell.NetCore
```

To load the AWSPowerShell.NetCore module into a PowerShell session automatically, add that command to your PowerShell profile. For more information about editing your PowerShell profile, see About Profiles in the PowerShell documentation.
Install AWSPowerShell on Windows PowerShell

You can install the AWS Tools for Windows PowerShell in one of two ways:

- Downloading the module from AWSPowerShell.zip and extracting it in one of the module directories. You can discover your module directories by displaying the value of the PSModulePath environment variable.
- Installing from the PowerShell Gallery using the Install-Module cmdlet as described in the following procedure.

To install AWSPowerShell from the PowerShell Gallery using the Install-Module cmdlet

You can install the AWSPowerShell from the PowerShell Gallery if you're running PowerShell 5.0 or later, or have installed PowerShellGet on PowerShell 3 or later. You can install and update AWSPowerShell from Microsoft's PowerShell Gallery by running the following command.

```
PS > Install-Module -Name AWSPowerShell
```

To load the AWSPowerShell module into a PowerShell session automatically, add the previous import-module cmdlet to your PowerShell profile. For more information about editing your PowerShell profile, see About Profiles in the PowerShell documentation.

⚠️ Note

The Tools for Windows PowerShell are installed by default on all Windows-based Amazon Machine Images (AMIs).

Enable Script Execution

To load the AWS Tools for PowerShell modules, you must enable PowerShell script execution. To enable script execution, run the Set-ExecutionPolicy cmdlet to set a policy of RemoteSigned. For more information, see About Execution Policies on the Microsoft Technet website.

⚠️ Note

This is a requirement only for computers that are running Windows. The ExecutionPolicy security restriction is not present on other operating systems.
To enable script execution

1. Administrator rights are required to set the execution policy. If you are not logged in as a user with administrator rights, open a PowerShell session as Administrator. Choose Start, and then choose All Programs. Choose Accessories, and then choose Windows PowerShell. Right-click Windows PowerShell, and on the context menu, choose Run as administrator.

2. At the command prompt, enter the following.

   ```powershell
   PS > Set-ExecutionPolicy RemoteSigned
   ```

   **Note**

   On a 64-bit system, you must do this separately for the 32-bit version of PowerShell, Windows PowerShell (x86).

If you don't have the execution policy set correctly, PowerShell shows the following error whenever you try to run a script, such as your profile.

```
File C:\Users\username\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1 cannot be loaded because the execution of scripts is disabled on this system. Please see "get-help about_signing" for more details.
At line:1 char:2
+ . 'C:\Users\username\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1'
+ CategoryInfo : NotSpecified: (:) [], PSSecurityException
+ FullyQualifiedErrorId : RuntimeException
```

The Tools for Windows PowerShell installer automatically updates the PSMODULEPATH to include the location of the directory that contains the AWSPowerShell module.

Because the PSMODULEPATH includes the location of the AWS module's directory, the Get-Module -ListAvailable cmdlet shows the module.

```
PS > Get-Module -ListAvailable

<table>
<thead>
<tr>
<th>ModuleType</th>
<th>Name</th>
<th>ExportedCommands</th>
</tr>
</thead>
</table>
```

Enable Script Execution
### Versioning

AWS releases new versions of the AWS Tools for PowerShell periodically to support new AWS services and features. To determine the version of the Tools that you have installed, run the `Get-AWSPowerShellVersion` cmdlet.

```powershell
PS > Get-AWSPowerShellVersion

Tools for PowerShell
Version 4.1.11.0
Copyright 2012-2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.

Amazon Web Services SDK for .NET
Core Runtime Version 3.7.0.12
Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.

Release notes: https://github.com/aws/aws-tools-for-powershell/blob/master/CHANGELOG.md

This software includes third party software subject to the following copyrights:
- Logging from log4net, Apache License
  [http://logging.apache.org/log4net/license.html]

You can also add the `-ListServiceVersionInfo` parameter to a `Get-AWSPowerShellVersion` command to see a list of the AWS services that are supported in the current version of the tools. If you use the modularized `AWS.Tools.*` option, only the modules that you currently have imported are displayed.

```powershell
PS > Get-AWSPowerShellVersion -ListServiceVersionInfo

<table>
<thead>
<tr>
<th>Service</th>
<th>Noun Prefix Module Name</th>
<th>SDK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---
<table>
<thead>
<tr>
<th>Version</th>
<th>-------</th>
<th>-----------------</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alexa For Business</td>
<td>ALXB</td>
<td>AWS.Tools.AlexaForBusiness</td>
</tr>
<tr>
<td>3.7.0.11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amplify Backend</td>
<td>AMPB</td>
<td>AWS.Tools.AmplifyBackend</td>
</tr>
<tr>
<td>3.7.0.11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amazon API Gateway</td>
<td>AG</td>
<td>AWS.Tools.APIGateway</td>
</tr>
<tr>
<td>3.7.0.11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amazon API Gateway Management API</td>
<td>AGM</td>
<td>AWS.Tools.ApiGatewayManagementApi</td>
</tr>
<tr>
<td>3.7.0.11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amazon API Gateway V2</td>
<td>AG2</td>
<td>AWS.Tools.ApiGatewayV2</td>
</tr>
<tr>
<td>3.7.0.11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amazon Appflow</td>
<td>AF</td>
<td>AWS.Tools.Appflow</td>
</tr>
<tr>
<td>3.7.1.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amazon Route 53</td>
<td>R53</td>
<td>AWS.Tools.Route53</td>
</tr>
<tr>
<td>3.7.0.12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amazon Route 53 Domains</td>
<td>R53D</td>
<td>AWS.Tools.Route53Domains</td>
</tr>
<tr>
<td>3.7.0.11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amazon Route 53 Resolver</td>
<td>R53R</td>
<td>AWS.Tools.Route53Resolver</td>
</tr>
<tr>
<td>3.7.1.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amazon Simple Storage Service (S3) S3</td>
<td>S3</td>
<td>AWS.Tools.S3</td>
</tr>
<tr>
<td>3.7.0.13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

To determine the version of PowerShell that you are running, enter $PSVersionTable to view the contents of the $PSVersionTable automatic variable.

**PS > $PSVersionTable**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSVersion</td>
<td>6.2.2</td>
</tr>
<tr>
<td>PSEdition</td>
<td>Core</td>
</tr>
<tr>
<td>GitCommitId</td>
<td>6.2.2</td>
</tr>
<tr>
<td>OS</td>
<td>Darwin 18.7.0 Darwin Kernel Version 18.7.0: Tue Aug 20 16:57:14 PDT 2019; root:xnu-4903.271.2~2/RELEASE_X86_64</td>
</tr>
<tr>
<td>Platform</td>
<td>Unix</td>
</tr>
<tr>
<td>PSCompatibleVersions</td>
<td>{1.0, 2.0, 3.0, 4.0…}</td>
</tr>
<tr>
<td>PSRemotingProtocolVersion</td>
<td>2.3</td>
</tr>
<tr>
<td>SerializationVersion</td>
<td>1.1.0.1</td>
</tr>
<tr>
<td>WSManStackVersion</td>
<td>3.0</td>
</tr>
</tbody>
</table>
Updating the AWS Tools for PowerShell on Windows

Periodically, as updated versions of the AWS Tools for PowerShell are released, you should update the version that you are running locally.

Update the modularized AWS.Tools modules

To update your AWS.Tools modules to the latest version, run the following command:

```
PS > Update-AWSToolsModule -CleanUp
```

This command updates all of the currently installed AWS.Tools modules and, after a successful update, removes other installed versions.

**Note**

The Update-AWSToolsModule cmdlet downloads all modules from the PSRepository named PSGallery (https://www.powershellgallery.com/) and considers it a trusted source. Use the command: Get-PSRepository -Name PSGallery for more information on this PSRepository.

Update the Tools for PowerShell Core

Run the Get-AWSPowerShellVersion cmdlet to determine the version that you are running, and compare that with the version of Tools for Windows PowerShell that is available on the PowerShell Gallery website. We suggest you check every two to three weeks. Support for new commands and AWS services is available only after you update to a version with that support.

Before you install a newer release of AWSPowerShell.NetCore, uninstall the existing module. Close any open PowerShell sessions before you uninstall the existing package. Run the following command to uninstall the package.

```
PS > Uninstall-Module -Name AWSPowerShell.NetCore -AllVersions
```

After the package is uninstalled, install the updated module by running the following command.

```
PS > Install-Module -Name AWSPowerShell.NetCore
```
After installation, run the command `Import-Module AWSPowerShell.NetCore` to load the updated cmdlets into your PowerShell session.

### Update the Tools for Windows PowerShell

Run the `Get-AWSPowerShellVersion` cmdlet to determine the version that you are running, and compare that with the version of Tools for Windows PowerShell that is available on the [PowerShell Gallery](https://www.powerhellgallery.com) website. We suggest you check every two to three weeks. Support for new commands and AWS services is available only after you update to a version with that support.

- If you installed by using the `Install-Module` cmdlet, run the following commands.

```
PS > Uninstall-Module -Name AWSPowerShell -AllVersions
PS > Install-Module -Name AWSPowerShell
```

- If you installed by using a downloaded ZIP file:
  1. Download the most recent version from the [Tools for PowerShell](https://aws.amazon.com) web site. Compare the package version number in the downloaded file name with the version number you get when you run the `Get-AWSPowerShellVersion` cmdlet.
  2. If the download version is a higher number than the version you have installed, close all Tools for Windows PowerShell consoles.
  3. Install the newer version of the Tools for Windows PowerShell.

After installation, run `Import-Module AWSPowerShell` to load the updated cmdlets into your PowerShell session. Or run the custom AWS Tools for PowerShell console from your **Start** menu.

### Installing AWS Tools for PowerShell on Linux or macOS

This topic provides instructions on how to install the AWS Tools for PowerShell on Linux or macOS.

#### Overview of Setup

To install AWS Tools for PowerShell on a Linux or macOS computer, you can choose from two package options:

- **AWS.Tools** – The modularized version of AWS Tools for PowerShell. Each AWS service is supported by its own individual, small module, with shared support modules `AWS.Tools.Common`. 
• **AWSPowerShell.NetCore** – The single, large-module version of AWS Tools for PowerShell. All AWS services are supported by this single, large module.

| Note | Be aware that the single module might be too large to use with AWS Lambda functions. Instead, use the modularized version shown above. |

Setting either of these up on a computer running Linux or macOS involves the following tasks, described in detail later in this topic:

1. Install PowerShell Core 6.0 or later on a supported system.
2. After installing PowerShell Core, start PowerShell by running `pwsh` in your system shell.
3. Install either `AWS.Tools` or `AWSPowerShell.NetCore`.
4. Run the appropriate `Import-Module` cmdlet to import the module into your PowerShell session.
5. Run the `Initialize-AWSDefaultConfiguration` cmdlet to provide your AWS credentials.

**Prerequisites**

To run the AWS Tools for PowerShell Core, your computer must be running PowerShell Core 6.0 or later.

- For a list of supported Linux platform releases and for information about how to install the latest version of PowerShell on a Linux-based computer, see [Installing PowerShell on Linux](https://docs.microsoft.com/en-us/powershell/scripting/install/installing-powershell-on-linux) on Microsoft's website. Some Linux-based operating systems, such as Arch, Kali, and Raspbian, are not officially supported, but have varying levels of community support.
- For information about supported macOS versions and about how to install the latest version of PowerShell on macOS, see [Installing PowerShell on macOS](https://docs.microsoft.com/en-us/powershell/scripting/install/installing-powershell-on-macos) on Microsoft's website.

**Install AWS.Tools on Linux or macOS**

You can install the modularized version of AWS Tools for PowerShell on computers that are running PowerShell Core 6.0 or later. For information about how to install PowerShell Core, see [Installing various versions of PowerShell](https://docs.microsoft.com/en-us/powershell/scripting/install/installing-various-versions-of-powershell) on the Microsoft PowerShell website.
You can install AWS.Tools in one of three ways:

- Using the cmdlets in the AWS.Tools.Installer module. This module simplifies the installation and update of other AWS.Tools modules. AWS.Tools.Installer requires PowerShellGet, and automatically downloads and installs an updated version of it. AWS.Tools.Installer automatically keeps your module versions in sync. When you install or update to a newer version of one module, the cmdlets in AWS.Tools.Installer automatically update all of your other AWS.Tools modules to the same version.

  This method is described in the procedure that follows.

- Downloading the modules from AWS.Tools.zip and extracting them in one of the module directories. You can discover your module directories by printing the value of the $Env:PSModulePath variable.

- Installing each service module from the PowerShell Gallery using the Install-Module cmdlet.

**To install AWS.Tools on Linux or macOS using the AWS.Tools.Installer module**

1. Start a PowerShell Core session by running the following command.

   $ pwsh

   ![Note](https://via.placeholder.com/150)
   We recommend that you *don't* run PowerShell as an administrator with elevated permissions except when required by the task at hand. This is because of the potential security risk and is inconsistent with the principle of least privilege.

2. To install the modularized AWS.Tools package using the AWS.Tools.Installer module, run the following command.

   PS > Install-Module -Name AWS.Tools.Installer

   ![Untrusted repository](https://via.placeholder.com/150)
   You are installing the modules from an untrusted repository. If you trust this repository, change its InstallationPolicy value by running the Set-PSRepository cmdlet. Are you sure you want to install the modules from 'PSGallery'?
If you are notified that the repository is "untrusted", you're asked if you want to install anyway. Enter `y` to allow PowerShell to install the module. To avoid the prompt and install the module without trusting the repository, you can run the following command.

```
PS > Install-Module -Name AWS.Tools.Installer -Force
```

3. You can now install the module for each service that you want to use. For example, the following command installs the Amazon EC2 and Amazon S3 modules. This command also installs any dependent modules that are required for the specified module to work. For example, when you install your first `AWS.Tools` service module, it also installs `AWS.Tools.Common`. This is a shared module required by all AWS service modules. It also removes older versions of the modules, and updates other modules to the same newer version.

```
PS > Install-AWSToolsModule AWS.Tools.EC2,AWS.Tools.S3 -CleanUp
```

### Note

The `Install-AWSToolsModule` cmdlet downloads all requested modules from the PSRepository named PSGallery (https://www.powershellgallery.com/) and...
considers the repository as a trusted source. Use the command `Get-PSRepository -Name PSGallery` for more information about this PSRepository.

The previous command installs modules into the default directories on your system. The actual directories depend on your operating system distribution and version and on the version of PowerShell you installed. For example, if you installed PowerShell 7 on a RHEL-like system, the default modules are most likely located in `/opt/microsoft/powershell/7/Modules` (or `$PSHOME/Modules`) and user modules are most likely located in `~/.local/share/powershell/Modules`. For more information, see [Install PowerShell on Linux](https://docs.microsoft.com/en-us/powershell/scripting/install/install-powershell-on-linux) on the Microsoft PowerShell website. To see where modules are installed, run the following command:

```
PS > Get-Module -ListAvailable
```

To install other modules, run similar commands with the appropriate module names, as found in the [PowerShell Gallery](https://www.powershellgallery.com).

### Install AWSPowerShell.NetCore on Linux or macOS

To upgrade to a newer release of AWSPowerShell.NetCore, follow the instructions in [Updating the AWS Tools for PowerShell on Linux or macOS](https://docs.aws.amazon.com/powershell/latest/userguide/installation.html). Uninstall earlier versions of AWSPowerShell.NetCore first.

You can install AWSPowerShell.NetCore in one of two ways:

- Downloading the module from [AWSPowerShell.NetCore.zip](https://github.com/aws/aws-powershell/releases) and extracting it in one of the module directories. You can discover your module directories by printing the value of the `$Env:PSModulePath` variable.
- Installing from the PowerShell Gallery using the `Install-Module` cmdlet as described in the following procedure.

#### To install AWSPowerShell.NetCore on Linux or macOS using the Install-Module cmdlet

Start a PowerShell Core session by running the following command.

```
$ pwsh
```
Note

We recommend that you *don't* start PowerShell by running `sudo pwsh` to run PowerShell with elevated, administrator rights. This is because of the potential security risk and is inconsistent with the principle of least privilege.

To install the AWSPowerShell.NetCore single-module package from the PowerShell Gallery, run the following command.

```powershell
PS > Install-Module -Name AWSPowerShell.NetCore
```

Untrusted repository

You are installing the modules from an untrusted repository. If you trust this repository, change its `InstallationPolicy` value by running the `Set-PSRepository` cmdlet. Are you sure you want to install the modules from 'PSGallery'?

[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "N"):

```
y
```

If you are notified that the repository is "untrusted", you're asked if you want to install anyway. Enter `y` to allow PowerShell to install the module. To avoid the prompt without trusting the repository, you can run the following command.

```powershell
PS > Install-Module -Name AWSPowerShell.NetCore -Force
```

You don't have to run this command as root, unless you want to install the AWS Tools for PowerShell for all users of a computer. To do this, run the following command in a PowerShell session that you have started with `sudo pwsh`.

```powershell
PS > Install-Module -Scope AllUsers -Name AWSPowerShell.NetCore -Force
```

Script Execution

The `Set-ExecutionPolicy` command isn't available on non-Windows systems. You can run `Get-ExecutionPolicy`, which shows that the default execution policy setting in PowerShell Core running on non-Windows systems is `Unrestricted`. For more information, see [About Execution Policies](https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_execution_policies?view=powershell-5.1) on the Microsoft TechNet website.
Because the PSModulePath includes the location of the AWS module's directory, the Get-Module -ListAvailable cmdlet shows the module that you installed.

**AWS.Tools**

```
PS > Get-Module -ListAvailable

    ModuleType Version Name                                PSEdition ExportedCommands
    ----------- ------- ----                                --------- ----------------
    Binary     3.3.563.1 AWS.Tools.Common                    Desk      {Clear-
AWSHistory, Set-AWSHistoryConfiguration, Initialize-AWSDefaultConfiguration, Clear-
AWSDefaultConfigurat...
```

**AWSPowerShell.NetCore**

```
PS > Get-Module -ListAvailable

    ModuleType Version Name                  ExportedCommands
    ----------- ------- ----------------
    Binary     3.3.563.1 AWSPowerShell.NetCore
```

**Configure a PowerShell Console to Use the AWS Tools for PowerShell Core (AWSPowerShell.NetCore Only)**

PowerShell Core typically loads modules automatically whenever you run a cmdlet in the module. But this doesn't work for AWSPowerShell.NetCore because of its large size. To start running AWSPowerShell.NetCore cmdlets, you must first run the Import-Module AWSPowerShell.NetCore command. This isn't required for cmdlets in AWS.Tools modules.

**Initialize Your PowerShell Session**

When you start PowerShell on a Linux-based or macOS-based system after you have installed the AWS Tools for PowerShell, you must run Initialize-AWSDefaultConfiguration to specify which AWS access key to use. For more information about Initialize-AWSDefaultConfiguration, see Using AWS Credentials.
Note

In earlier (before 3.3.96.0) releases of the AWS Tools for PowerShell, this cmdlet was named Initialize-AWSDefaults.

Versioning

AWS releases new versions of the AWS Tools for PowerShell periodically to support new AWS services and features. To determine the version of the AWS Tools for PowerShell that you have installed, run the Get-AWSPowerShellVersion cmdlet.

PS > Get-AWSPowerShellVersion

Tools for PowerShell
Version 4.0.123.0
Copyright 2012-2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.

Amazon Web Services SDK for .NET
Core Runtime Version 3.3.103.22
Copyright 2009-2015 Amazon.com, Inc. or its affiliates. All Rights Reserved.

Release notes: https://github.com/aws/aws-tools-for-powershell/blob/master/CHANGELOG.md

This software includes third party software subject to the following copyrights:
- Logging from log4net, Apache License
  [http://logging.apache.org/log4net/license.html]

To see a list of the supported AWS services in the current version of the tools, add the -ListServiceVersionInfo parameter to a Get-AWSPowerShellVersion cmdlet.

To determine the version of PowerShell that you are running, enter $PSVersionTable to view the contents of the $PSVersionTable automatic variable.

PS > $PSVersionTable

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>----</td>
<td>-----</td>
</tr>
<tr>
<td>PSVersion</td>
<td>6.2.2</td>
</tr>
<tr>
<td>PSEdition</td>
<td>Core</td>
</tr>
<tr>
<td>GitCommitId</td>
<td>6.2.2</td>
</tr>
</tbody>
</table>
Updating the AWS Tools for PowerShell on Linux or macOS

Periodically, as updated versions of the AWS Tools for PowerShell are released, you should update the version that you're running locally.

Update the modularized AWS.Tools modules

To update your AWS.Tools modules to the latest version, run the following command:

```
PS > Update-AWSToolsModule -CleanUp
```

This command updates all of the currently installed AWS.Tools modules and, for those modules that were successfully updated, removes the earlier versions.

Note

The Update-AWSToolsModule cmdlet downloads all modules from the PSRepository named PSGallery (https://www.powershellgallery.com/) and considers it a trusted source. Use the command Get-PSRepository -Name PSGallery for more information about this PSRepository.

Update the Tools for PowerShell Core

Run the Get-AWSPowerShellVersion cmdlet to determine the version that you are running, and compare that with the version of Tools for Windows PowerShell that is available on the PowerShell Gallery website. We suggest you check every two to three weeks. Support for new commands and AWS services is available only after you update to a version with that support.

Before you install a newer release of AWSPowerShell.NetCore, uninstall the existing module. Close any open PowerShell sessions before you uninstall the existing package. Run the following command to uninstall the package.
PS > Uninstall-Module -Name AWSPowerShell.NetCore -AllVersions

After the package is uninstalled, install the updated module by running the following command.

PS > Install-Module -Name AWSPowerShell.NetCore

After installation, run the command `Import-Module AWSPowerShell.NetCore` to load the updated cmdlets into your PowerShell session.

**Related Information**

- [Get started with the AWS Tools for Windows PowerShell](#)
- [Work with AWS services in the AWS Tools for PowerShell](#)

### Migrating from AWS Tools for PowerShell Version 3.3 to Version 4

AWS Tools for PowerShell version 4 is a backward-compatible update to AWS Tools for PowerShell version 3.3. It adds significant improvements while maintaining existing cmdlet behavior.

Your existing scripts should continue to work after upgrading to the new version, but we do recommend that you test them thoroughly before upgrading your production environments.

This section describes the changes and explains how they might impact your scripts.

**New Fully Modularized AWS .Tools Version**

The AWSPowerShell.NetCore and AWSPowerShell packages were "monolithic". This meant that all of the AWS services were supported in the same module, making it very large, and growing larger as each new AWS service and feature was added. The new AWS .Tools package is broken up into smaller modules that give you the flexibility to download and install only those that you require for the AWS services that you use. The package includes a shared AWS .Tools .Common module that is required by all of the other modules, and an AWS .Tools .Installer module that simplifies installing, updating, and removing modules as needed.

This also enables auto-importing of cmdlets on first call, without having to first call `Import-Module`. However, to interact with the associated .NET objects before calling a cmdlet, you must still call `Import-Module` to let PowerShell know about the relevant .NET types.

**Related Information**
For example, the following command has a reference to Amazon.EC2.Model.Filter. This type of reference can't trigger auto-importing, so you must call Import-Module first or the command fails.

```powershell
PS > $filter = [Amazon.EC2.Model.Filter]@{Name="vpc-id";Values="vpc-1234abcd"}
InvalidOperation: Unable to find type [Amazon.EC2.Model.Filter].
```

```
PS > Import-Module AWS.Tools.EC2
PS > $filter = [Amazon.EC2.Model.Filter]@{Name="vpc-id";Values="vpc-1234abcd"}
PS > Get-EC2Instance -Filter $filter -Select Reservations.Instances.InstanceId
i-0123456789abcdefg
i-0123456789hijklmn
```

**New Get-AWSService cmdlet**

To help you discover the names of the modules for each AWS service in the AWS.Tools collection of modules, you can use the Get-AWSService cmdlet.

```
PS > Get-AWSService
Service : ACMPCA
  CmdletNounPrefix : PCA
  ModuleName : AWS.Tools.ACMPCA
  SDKAssemblyVersion : 3.3.101.56
  ServiceName : Certificate Manager Private Certificate Authority

Service : AlexaForBusiness
  CmdletNounPrefix : ALXB
  ModuleName : AWS.Tools.AlexaForBusiness
  SDKAssemblyVersion : 3.3.106.26
  ServiceName : Alexa For Business
```

**New -Select Parameter to Control the Object Returned by a Cmdlet**

Most cmdlets in version 4 support a new -Select parameter. Each cmdlet calls the AWS service APIs for you using the AWS SDK for .NET. Then the AWS Tools for PowerShell client converts the response into an object that you can use in your PowerShell scripts and pipe to other commands. Sometimes the final PowerShell object has more fields or properties in the original response than you need, and other times you might want the object to include fields or properties of the response.
that are not there by default. The -Select parameter enables you to specify what is included in the .NET object returned by the cmdlet.

For example, the **Get-S3Object** cmdlet invokes the Amazon S3 SDK operation **ListObjects**. That operation returns a **ListObjectsResponse** object. However, by default, the Get-S3Object cmdlet returns only the S3Objects element of the SDK response to the PowerShell user. In the following example, that object is an array with two elements.

```powershell
PS > Get-S3Object -BucketName mybucket

ETag : "01234567890123456789012345678901111"
BucketName : mybucket
Key : file1.txt
LastModified : 9/30/2019 1:31:40 PM
Owner : Amazon.S3.Model.Owner
Size : 568
StorageClass : STANDARD

ETag : "01234567890123456789012345678902222"
BucketName : mybucket
Key : file2.txt
LastModified : 7/15/2019 9:36:54 AM
Owner : Amazon.S3.Model.Owner
Size : 392
StorageClass : STANDARD
```

In AWS Tools for PowerShell version 4, you can specify `-Select *` to return the complete .NET response object returned by the SDK API call.

```powershell
PS > Get-S3Object -BucketName mybucket -Select *

IsTruncated    : False
NextMarker     :
S3Objects      : {file1.txt, file2.txt}
Name           : mybucket
Prefix         :
MaxKeys        : 1000
CommonPrefixes : {}
Delimiter      :
```

You can also specify the path to the specific nested property you want. The following example returns only the Key property of each element in the S3Objects array.

```powershell
PS > Get-S3Object -BucketName mybucket -Select .Key

file1.txt
file2.txt
```
In certain situations it can be useful to return a cmdlet parameter. You can do this with `-Select ParameterName`. This feature supplants the `-PassThru` parameter, which is still available but deprecated.

```powershell
PS > Get-S3Object -BucketName mybucket -Select S3Objects.Key
file1.txt
file2.txt
```

The reference topic for each cmdlet identifies whether it supports the `-Select` parameter.

### More Consistent Limiting of the Number of Items in the Output

Earlier versions of AWS Tools for PowerShell enabled you to use the `-MaxItems` parameter to specify the maximum number of objects returned in the final output.

This behavior is removed from AWS.Tools.

This behavior is deprecated in AWSPowerShell.NetCore and AWSPowerShell, and will be removed from those versions in a future release.

If the underlying service API supports a `MaxItems` parameter, it’s still available and functions as the API specifies. But it no longer has the added behavior of limiting the number of items returned in the output of the cmdlet.

To limit the number of items returned in the final output, pipe the output to the `Select-Object` cmdlet and specify the `-First n` parameter, where `n` is the maximum number of items to include in the final output.

```powershell
PS > Get-S3ObjectV2 -BucketName BUCKET_NAME -Select S3Objects.Key | select -first 2
file1.txt
file2.txt
```
Not all AWS services supported -MaxItems in the same way, so this removes that inconsistency and the unexpected results that sometimes occurred. Also, -MaxItems combined with the new -Select parameter could sometimes result in confusing results.

**Easier to Use Stream Parameters**

Parameters of type Stream or byte[] can now accept string, string[], or FileInfo values.

For example, you can use any of the following examples.

```powershell
PS > Invoke-LMFunction -FunctionName MyTestFunction -PayloadStream '{
    >> "some": "json"
    >> }

PS > Invoke-LMFunction -FunctionName MyTestFunction -PayloadStream (ls .\some.json)

PS > Invoke-LMFunction -FunctionName MyTestFunction -PayloadStream @('{', '"some": "json"}', ', '})
```

AWS Tools for PowerShell converts all strings to byte[] using UTF-8 encoding.

**Extending the Pipe by Property Name**

To make the user experience more consistent, you can now pass pipeline input by specifying the property name for *any* parameter.

In the following example, we create a custom object with properties that have names that match the parameter names of the target cmdlet. When the cmdlet runs, it automatically consumes those properties as its parameters.

```powershell
PS > [pscustomobject] @{ BucketName='myBucket'; Key='file1.txt'; PartNumber=1 } | Get-S3ObjectMetadata
```

**Note**

Some properties supported this in earlier versions of AWS Tools for PowerShell. Version 4 makes this more consistent by enabling it for *all* parameters.
Static Common Parameters

To improve consistency in version 4.0 of AWS Tools for PowerShell, all parameters are static.

In earlier versions of AWS Tools for PowerShell, some common parameters such as AccessKey, SecretKey, ProfileName, or Region, were dynamic, while all other parameters were static. This could create problems because PowerShell binds static parameters before dynamic ones. For example, let's say you ran the following command.

```
PS > Get-EC2Region -Region us-west-2
```

Earlier versions of PowerShell bound the value `us-west-2` to the `-RegionName` static parameter instead of the `-Region` dynamic parameter. Likely, this could confuse users.

**AWS.Tools Declares and Enforces Mandatory Parameters**

The AWS.Tools.* modules now declare and enforce mandatory cmdlet parameters. When an AWS Service declares that a parameter of an API is required, PowerShell prompts you for the corresponding cmdlet parameter if you didn't specify it. This applies only to AWS.Tools. To ensure backward compatibility, this does not apply to AWSPowerShell.NetCore or AWSPowerShell.

**All Parameters Are Nullable**

You can now assign `$null` to value type parameters (numbers and dates). This change should not affect existing scripts. This enables you to bypass the prompt for a mandatory parameter. Mandatory parameters are enforced in AWS.Tools only.

If you run the following example using version 4, it effectively bypasses client-side validation because you provide a "value" for each mandatory parameter. However, the Amazon EC2 API service call fails because the AWS service still requires that information.

```
PS > Get-EC2InstanceAttribute -InstanceId $null -Attribute $null
WARNING: You are passing $null as a value for parameter InstanceId which is marked as required.
In case you believe this parameter was incorrectly marked as required, report this by opening an issue at https://github.com/aws/aws-tools-for-powershell/issues.
WARNING: You are passing $null as a value for parameter Attribute which is marked as required.
```
In case you believe this parameter was incorrectly marked as required, report this by opening an issue at https://github.com/aws/aws-tools-for-powershell/issues.

Get-EC2InstanceAttribute : The request must contain the parameter instanceId

Removing Previously Deprecated Features

The following features were deprecated in previous releases of AWS Tools for PowerShell and are removed in version 4:

- Removed the -Terminate parameter from the Stop-EC2Instance cmdlet. Use Remove-EC2Instance instead.
- Removed the -ProfileName parameter from the Clear-AWSCredential cmdlet. Use Remove-AWSCredentialProfile instead.
- Removed cmdlets Import-EC2Instance and Import-EC2Volume.
Get started with the AWS Tools for Windows PowerShell

Some of the topics in this section describe the fundamentals of using the Tools for Windows PowerShell after you have installed the tools. For example, they explain how to specify which credentials and AWS Region the Tools for Windows PowerShell should use when interacting with AWS.

Other topics in this section provide information about advanced ways that you can configure the tools, your environment, and your projects.

Topics

- Configure tool authentication with AWS
- Specify AWS Regions
- Configure federated identity with the AWS Tools for PowerShell
- Cmdlet discovery and aliases
- Pipelining and $AWSHistory
- Credential and profile resolution
- Additional information about users and roles
- Using legacy credentials

Configure tool authentication with AWS

You must establish how your code authenticates with AWS when developing with AWS services. There are different ways in which you can configure programmatic access to AWS resources, depending on the environment and the AWS access available to you.

To see various methods of authentication for the Tools for PowerShell, see Authentication and access in the AWS SDKs and Tools Reference Guide.

This topic assumes that a new user is developing locally, has not been given a method of authentication by their employer, and will be using AWS IAM Identity Center to obtain temporary credentials. If your environment doesn't fall under these assumptions, some of the information in this topic might not apply to you, or some of the information might have already been given to you.
Configuring this environment requires several steps, which are summarized as follows:

1. **Enable and configure IAM Identity Center**
2. **Configure the Tools for PowerShell to use IAM Identity Center.**
3. **Start an AWS access portal session**

**Enable and configure IAM Identity Center**

To use AWS IAM Identity Center, it must first be enabled and configured. To see details about how to do this for PowerShell, look at Step 1 in the topic for IAM Identity Center authentication in the AWS SDKs and Tools Reference Guide. Specifically, follow any necessary instructions under I do not have established access through IAM Identity Center.

**Configure the Tools for PowerShell to use IAM Identity Center.**

Starting with version 4.1.538 of the Tools for PowerShell, the recommended method to configure SSO credentials and start an AWS access portal session is to use the Initialize-AWSSSOConfiguration and Invoke-AWSSSOLogin cmdlets, as described in this topic. If you don't have access to that version of the Tools for PowerShell (or later) or can't use those cmdlets, you can still perform these tasks by using the AWS CLI. To find out how, see Use the AWS CLI for portal login.

The following procedure updates the shared AWS config file with SSO information that the Tools for PowerShell uses to obtain temporary credentials. As a consequence of this procedure, an AWS access portal session is also started. If the shared config file already has SSO information and you just want to know how to start an access portal session using the Tools for PowerShell, see the next section in this topic, Start an AWS access portal session.

1. If you haven't already done so, open PowerShell and install the AWS Tools for PowerShell as appropriate for your operating system and environment, including the common cmdlets. For information about how to do this, see Installing the AWS Tools for PowerShell.

   For example, if installing the modularized version of the Tools for PowerShell on Windows, you would most likely run commands similar to the following:
2. Run the following command. Replace the example property values with values from your IAM Identity Center configuration. For information about these properties and how to find them, see IAM Identity Center credential provider settings in the AWS SDKs and Tools Reference Guide.

```powershell
$params = @{
    ProfileName = 'my-sso-profile'
    AccountId = '111122223333'
    RoleName = 'SamplePermissionSet'
    SessionName = 'my-sso-session'
    StartUrl = 'https://provided-domain.awsapps.com/start'
    SSORegion = 'us-west-2'
    RegistrationScopes = 'sso:account:access'
};
Initialize-AWSSSOConfiguration @params
```

Alternatively, you can simply use the cmdlet by itself, Initialize-AWSSSOConfiguration, and the Tools for PowerShell prompts you for the property values.

Considerations for certain property values:

- If you simply followed the instructions to enable and configure IAM Identity Center, the value for -RoleName might be PowerUserAccess. But if you created an IAM Identity Center permission set specifically for PowerShell work, use that instead.
- Be sure to use the AWS Region where you have configured IAM Identity Center.

3. At this point, the shared AWS config file contains a profile called my-sso-profile with a set of configuration values that can be referenced from the Tools for PowerShell. To find the location of this file, see Location of the shared files in the AWS SDKs and Tools Reference Guide.

The Tools for PowerShell uses the profile's SSO token provider to acquire credentials before sending requests to AWS. The sso_role_name value, which is an IAM role connected to an IAM Identity Center permission set, should allow access to the AWS services used in your application.

The following sample shows the profile that was created by using the command shown above. Some of the property values and their order might be different in your actual profile.
The profile's `sso-session` property refers to the section named `my-sso-session`, which contains settings to initiate an AWS access portal session.

```powershell
[profile my-sso-profile]
  sso_account_id=111122223333
  sso_role_name=SamplePermissionSet
  sso_session=my-sso-session

[sso-session my-sso-session]
  sso_region=us-west-2
  sso_registration_scopes=sso:account:access
  sso_start_url=https://provided-domain.awsapps.com/start/
```

4. If you already have an active AWS access portal session, the Tools for PowerShell informs you that you are already logged in.

If that's not the case, the Tools for PowerShell attempts to automatically open the SSO authorization page in your default web browser. Follow the prompts in your browser, which might include an SSO authorization code, username and password, and permission to access AWS IAM Identity Center accounts and permission sets.

The Tools for PowerShell informs you that SSO login was successful.

**Start an AWS access portal session**

Before running commands that accesses AWS services, you need an active AWS access portal session so that the Tools for PowerShell can use IAM Identity Center authentication to resolve credentials. To sign in to the AWS access portal, run the following command in PowerShell, where `-ProfileName my-sso-profile` is the name of the profile that was created in the shared config file when you followed the procedure in the previous section of this topic.

```powershell
Invoke-AWSSSOLogin -ProfileName my-sso-profile
```

If you already have an active AWS access portal session, the Tools for PowerShell informs you that you are already logged in.

If that's not the case, the Tools for PowerShell attempts to automatically open the SSO authorization page in your default web browser. Follow the prompts in your browser, which might
include an SSO authorization code, username and password, and permission to access AWS IAM Identity Center accounts and permission sets.

The Tools for PowerShell informs you that SSO login was successful.

To test if you already have an active session, run the following command after installing or importing the `AWS.Tools.SecurityToken` module as needed.

```
Get-STSCallerIdentity -ProfileName my-sso-profile
```

The response to the `Get-STSCallerIdentity` cmdlet reports the IAM Identity Center account and permission set configured in the shared config file.

**Example**

The following is an example of how to use IAM Identity Center with the Tools for PowerShell. It assumes the following:

- You have enabled IAM Identity Center and configured it as described previously in this topic. The SSO properties are in the `my-sso-profile` profile, which was configured earlier in this topic.
- When you log in through the `Initialize-AWSSSOConfiguration` or `Invoke-AWSSSOLogin` cmdlets, the user has at least read-only permissions for Amazon S3.
- Some S3 buckets are available for that user to view.

Install or import the `AWS.Tools.S3` module as needed and then use the following PowerShell command to display a list of the S3 buckets.

```
Get-S3Bucket -ProfileName my-sso-profile
```

**Additional information**

- For more options on authentication for the Tools for PowerShell, such as the use of profiles and environment variables, see the [configuration] chapter in the AWS SDKs and Tools Reference Guide.

- Some commands require an AWS Region to be specified. There are a number of ways to do so, including the `-Region` cmdlet option, the `[default]` profile, and the `AWS_REGION` environment variable. For more information, see [Specify AWS Regions] in this guide and [AWS Region] in the AWS SDKs and Tools Reference Guide.
• To learn more about best practices, see Security best practices in IAM in the IAM User Guide.
• To create short-term AWS credentials, see Temporary Security Credentials in the IAM User Guide.
• To learn about other credential providers, see Standardized credential providers in the AWS SDKs and Tools Reference Guide.

Topics
• Use the AWS CLI for portal login

Use the AWS CLI for portal login

Starting with version 4.1.538 of the Tools for PowerShell, the recommended method to configure SSO credentials and start an AWS access portal session is to use the Initialize-AWSSS0Configuration and Invoke-AWSSS0Login cmdlets, as described in Configure tool authentication with AWS. If you don't have access to that version of the Tools for PowerShell (or later) or can't use those cmdlets, you can still perform these tasks by using the AWS CLI.

Configure the Tools for PowerShell to use IAM Identity Center through the AWS CLI.

If you haven't already done so, be sure to Enable and configure IAM Identity Center before you proceed.

Information about how to configure the Tools for PowerShell to use IAM Identity Center through the AWS CLI is in Step 2 in the topic for IAM Identity Center authentication in the AWS SDKs and Tools Reference Guide. After you complete this configuration, your system should contain the following elements:

• The AWS CLI, which you use to start an AWS access portal session before you run your application.
• The shared AWS config file that contains a [default] profile with a set of configuration values that can be referenced from the Tools for PowerShell. To find the location of this file, see Location of the shared files in the AWS SDKs and Tools Reference Guide. The Tools for PowerShell uses the profile's SSO token provider to acquire credentials before sending requests to AWS. The sso_role_name value, which is an IAM role connected to an IAM Identity Center permission set, should allow access to the AWS services used in your application.
The following sample config file shows a [default] profile set up with an SSO token provider. The profile's sso_session setting refers to the named sso-session section. The sso-session section contains settings to initiate an AWS access portal session.

```
[default]
    sso_session = my-sso
    sso_account_id = 111122223333
    sso_role_name = SampleRole
    region = us-east-1
    output = json

[sso-session my-sso]
    sso_region = us-east-1
    sso_start_url = https://provided-domain.awsapps.com/start
    sso_registration_scopes = sso:account:access
```

⚠️ Important

Your PowerShell session must have the following modules installed and imported so that SSO resolution can work:

- AWS.Tools.SSO
- AWS.Tools.SSOOIDC

If you're using an older version of the Tools for PowerShell and you don't have these modules, you will get an error similar to the following: "Assembly AWSSDK.SSOOIDC could not be found...".

**Start an AWS access portal session**

Before running commands that accesses AWS services, you need an active AWS access portal session so that the Tools for Windows PowerShell can use IAM Identity Center authentication to resolve credentials. Depending on your configured session lengths, your access will eventually expire and the Tools for Windows PowerShell will encounter an authentication error. To sign in to the AWS access portal, run the following command in the AWS CLI.
aws sso login

Since you are using the [default] profile, you do not need to call the command with the --profile option. If your SSO token provider configuration is using a named profile, the command is `aws sso login --profile named-profile` instead. For more information about named profiles, see the Profiles section in the AWS SDKs and Tools Reference Guide.

To test if you already have an active session, run the following AWS CLI command (with the same consideration for named profile):

```
aws sts get-caller-identity
```

The response to this command should report the IAM Identity Center account and permission set configured in the shared config file.

**Note**

If you already have an active AWS access portal session and run `aws sso login`, you will not be required to provide credentials. The sign-in process might prompt you to allow the AWS CLI access to your data. Because the AWS CLI is built on top of the SDK for Python, permission messages may contain variations of the botocore name.

**Example**

The following is an example of how to use IAM Identity Center with the Tools for PowerShell. It assumes the following:

- You have enabled IAM Identity Center and configured it as described previously in this topic. The SSO properties are in the [default] profile.
- When you log in through the AWS CLI by using `aws sso login`, that user has at least read-only permissions for Amazon S3.
- Some S3 buckets are available for that user to view.

Use the following PowerShell commands to display a list of the S3 buckets:
Install-Module AWS.Tools.Installer
Install-AWSToolsModule S3
# And if using an older version of the AWS Tools for PowerShell:
Install-AWSToolsModule SSO, SSOOIDC

# In older versions of the AWS Tools for PowerShell, we're not invoking a cmdlet from these modules directly,
# so we must import them explicitly:
Import-Module AWS.Tools.SSO
Import-Module AWS.Tools.SSOOIDC

# Older versions of the AWS Tools for PowerShell don't support the SSO login flow, so login with the CLI
aws sso login

# Now we can invoke cmdlets using the SSO profile
Get-S3Bucket

As mentioned above, since you are using the [default] profile, you do not need to call the Get-S3Bucket cmdlet with the -ProfileName option. If your SSO token provider configuration is using a named profile, the command is Get-S3Bucket -ProfileName named-profile. For more information about named profiles, see the Profiles section in the AWS SDKs and Tools Reference Guide.

Additional information

- For more options on authentication for the Tools for PowerShell, such as the use of profiles and environment variables, see the configuration chapter in the AWS SDKs and Tools Reference Guide.
- Some commands require an AWS Region to be specified. There are a number of ways to do so, including the -Region cmdlet option, the [default] profile, and the AWS_REGION environment variable. For more information, see Specify AWS Regions in this guide and AWS Region in the AWS SDKs and Tools Reference Guide.
- To learn more about best practices, see Security best practices in IAM in the IAM User Guide.
- To create short-term AWS credentials, see Temporary Security Credentials in the IAM User Guide.
- To learn about other credential providers, see Standardized credential providers in the AWS SDKs and Tools Reference Guide.
Specify AWS Regions

There are two ways to specify the AWS Region to use when running AWS Tools for PowerShell commands:

- Use the \-Region common parameter on individual commands.
- Use the Set-DefaultAWSRegion command to set a default Region for all commands.

Many AWS cmdlets fail if the Tools for Windows PowerShell can't figure out what Region to use. Exceptions include cmdlets for Amazon S3, Amazon SES, and AWS Identity and Access Management, which automatically default to a global endpoint.

To specify the region for a single AWS command

Add the \-Region parameter to your command, such as the following.

```powershell
PS > Get-EC2Image \-Region us-west-2
```

To set a default region for all AWS CLI commands in the current session

From the PowerShell command prompt, type the following command.

```powershell
PS > Set-DefaultAWSRegion \-Region us-west-2
```

**Note**

This setting persists only for the current session. To apply the setting to all of your PowerShell sessions, add this command to your PowerShell profile as you did for the Import-Module command.

To view the current default region for all AWS CLI commands

From the PowerShell command prompt, type the following command.

```powershell
PS > Get-DefaultAWSRegion
```

<table>
<thead>
<tr>
<th>Region</th>
<th>Name</th>
<th>IsShellDefault</th>
</tr>
</thead>
<tbody>
<tr>
<td>------</td>
<td>--------</td>
<td>---------------</td>
</tr>
<tr>
<td></td>
<td>--------</td>
<td>---------------</td>
</tr>
</tbody>
</table>
To clear the current default Region for all AWS CLI commands

From the PowerShell command prompt, type the following command.

```
PS > Clear-DefaultAWSRegion
```

To view a list of all available AWS Regions

From the PowerShell command prompt, type the following command. The third column in the sample output identifies which Region is the default for your current session.

```
PS > Get-AWSRegion
```

<table>
<thead>
<tr>
<th>Region</th>
<th>Name</th>
<th>IsShellDefault</th>
</tr>
</thead>
<tbody>
<tr>
<td>ap-east-1</td>
<td>Asia Pacific (Hong Kong)</td>
<td>False</td>
</tr>
<tr>
<td>ap-northeast-1</td>
<td>Asia Pacific (Tokyo)</td>
<td>False</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>us-east-2</td>
<td>US East (Ohio)</td>
<td>False</td>
</tr>
<tr>
<td>us-west-1</td>
<td>US West (N. California)</td>
<td>False</td>
</tr>
<tr>
<td>us-west-2</td>
<td>US West (Oregon)</td>
<td>True</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

>Note

Some Regions might be supported but not included in the output of the Get-AWSRegion cmdlet. For example, this is sometimes true of Regions that are not yet global. If you're not able to specify a Region by adding the -Region parameter to a command, try specifying the Region in a custom endpoint instead, as shown in the following section.

Specifying a Custom or Nonstandard Endpoint

Specify a custom endpoint as a URL by adding the -EndpointUrl common parameter to your Tools for Windows PowerShell command, in the following sample format.

```
PS > Some-AWS-PowerShellCmdlet -EndpointUrl "custom endpoint URL" -Other -Parameters
```
The following is an example using the Get-EC2Instance cmdlet. The custom endpoint is in the us-west-2, or US West (Oregon) Region in this example, but you can use any other supported AWS Region, including regions that are not enumerated by Get-AWSRegion.

```
PS > Get-EC2Instance -EndpointUrl "https://service-custom-url.us-west-2.amazonaws.com" -InstanceID "i-0555a30a2000000e1"
```

**Additional information**

For additional information about AWS Regions, see [AWS Region](#) in the *AWS SDKs and Tools Reference Guide*.

**Configure federated identity with the AWS Tools for PowerShell**

To let users in your organization access AWS resources, you must configure a standard and repeatable authentication method for purposes of security, auditability, compliance, and the capability to support role and account separation. Although it's common to provide users with the ability to access AWS APIs, without federated API access, you would also have to create AWS Identity and Access Management (IAM) users, which defeats the purpose of using federation. This topic describes SAML (Security Assertion Markup Language) support in the AWS Tools for PowerShell that eases your federated access solution.

SAML support in the AWS Tools for PowerShell lets you provide your users federated access to AWS services. SAML is an XML-based, open-standard format for transmitting user authentication and authorization data between services; in particular, between an identity provider (such as Active Directory Federation Services), and a service provider (such as AWS). For more information about SAML and how it works, see [SAML](#) on Wikipedia, or [SAML Technical Specifications](#) at the Organization for the Advancement of Structured Information Standards (OASIS) website. SAML support in the AWS Tools for PowerShell is compatible with SAML 2.0.

**Prerequisites**

You must have the following in place before you try to use SAML support for the first time.

- A federated identity solution that is correctly integrated with your AWS account for console access by using only your organizational credentials. For more information about how to do this specifically for Active Directory Federation Services, see [About SAML 2.0 Federation](#) in the IAM
User Guide, and the blog post, Enabling Federation to AWS Using Windows Active Directory, AD FS, and SAML 2.0. Although the blog post covers AD FS 2.0, the steps are similar if you are running AD FS 3.0.

- Version 3.1.31.0 or newer of the AWS Tools for PowerShell installed on your local workstation.

How an Identity-Federated User Gets Federated Access to AWS Service APIs

The following process describes, at a high level, how an Active Directory (AD) user is federated by AD FS to gain access to AWS resources.

1. The client on federated user's computer authenticates against AD FS.
2. If authentication succeeds, AD FS sends the user a SAML assertion.
3. AWS Security Token Service
4. Service APIs
5. AWS

1. The client on federated user's computer authenticates against AD FS.
2. If authentication succeeds, AD FS sends the user a SAML assertion.
3. The user's client sends the SAML assertion to the AWS Security Token Service (STS) as part of a SAML federation request.

4. STS returns a SAML response that contains AWS temporary credentials for a role the user can assume.

5. The user accesses AWS service APIs by including those temporary credentials in request made by AWS Tools for PowerShell.

How SAML Support Works in the AWS Tools for PowerShell

This section describes how AWS Tools for PowerShell cmdlets enable configuration of SAML-based identity federation for users.

1. AWS Tools for PowerShell authenticates against AD FS by using the Windows user's current credentials, or interactively, when the user tries to run a cmdlet that requires credentials to call into AWS.

2. AD FS authenticates the user.

3. AD FS generates a SAML 2.0 authentication response that includes an assertion; the purpose of the assertion is to identify and provide information about the user. AWS Tools for PowerShell extracts the list of the user's authorized roles from the SAML assertion.
4. AWS Tools for PowerShell forwards the SAML request, including the requested role's Amazon Resource Names (ARN), to STS by making the `AssumeRoleWithSAMLRequest` API call.

5. If the SAML request is valid, STS returns a response that contains the AWS AccessKeyId, SecretAccessKey, and SessionToken. These credentials last for 3,600 seconds (1 hour).

6. The user now has valid credentials to work with any AWS service APIs that the user's role is authorized to access. AWS Tools for PowerShell automatically applies these credentials for any subsequent AWS API calls, and renews them automatically when they expire.

**Note**

When the credentials expire, and new credentials are required, AWS Tools for PowerShell automatically re authenticates with AD FS, and obtains new credentials for a subsequent hour. For users of domain-joined accounts, this process occurs silently. For accounts that are not domain-joined, AWS Tools for PowerShell prompts users to enter their credentials before they can reauthenticate.

---

# How to Use the PowerShell SAML Configuration Cmdlets

AWS Tools for PowerShell includes two new cmdlets that provide SAML support.

- `Set-AWSSamlEndpoint` configures your AD FS endpoint, assigns a friendly name to the endpoint, and optionally describes the authentication type of the endpoint.

- `Set-AWSSamlRoleProfile` creates or edits a user account profile that you want to associate with an AD FS endpoint, identified by specifying the friendly name you provided to the `Set-AWSSamlEndpoint` cmdlet. Each role profile maps to a single role that a user is authorized to perform.

  Just as with AWS credential profiles, you assign a friendly name to the role profile. You can use the same friendly name with the `Set-AWSCredential` cmdlet, or as the value of the `-ProfileName` parameter for any cmdlet that invokes AWS service APIs.

Open a new AWS Tools for PowerShell session. If you are running PowerShell 3.0 or newer, the AWS Tools for PowerShell module is automatically imported when you run any of its cmdlets. If you are running PowerShell 2.0, you must import the module manually by running the ``` Import-Module``` cmdlet, as shown in the following example.
How to Run the Set-AWSSamlEndpoint and Set-AWSSamlRoleProfile Cmdlets

1. First, configure the endpoint settings for the AD FS system. The simplest way to do this is to store the endpoint in a variable, as shown in this step. Be sure to replace the placeholder account IDs and AD FS host name with your own account IDs and AD FS host name. Specify the AD FS host name in the Endpoint parameter.

   ```powershell
   ```

2. To create the endpoint settings, run the Set-AWSSamlEndpoint cmdlet, specifying the correct value for the AuthenticationType parameter. Valid values include Basic, Digest, Kerberos, Negotiate, and NTLM. If you do not specify this parameter, the default value is Kerberos.

   ```powershell
   PS > $epName = Set-AWSSamlEndpoint -Endpoint $endpoint -StoreAs ADFS-Demo -AuthenticationType NTLM
   ```

   The cmdlet returns the friendly name you assigned by using the -StoreAs parameter, so you can use it when you run Set-AWSSamlRoleProfile in the next line.

3. Now, run the Set-AWSSamlRoleProfile cmdlet to authenticate with the AD FS identity provider and get the set of roles (in the SAML assertion) that the user is authorized to perform.

   The Set-AWSSamlRoleProfile cmdlet uses the returned set of roles to either prompt the user to select a role to associate with the specified profile, or validate that role data provided in parameters is present (if not, the user is prompted to choose). If the user is authorized for only one role, the cmdlet associates the role with the profile automatically, without prompting the user. There is no need to provide a credential to set up a profile for domain-joined usage.

   ```powershell
   PS > Set-AWSSamlRoleProfile -StoreAs SAMLDemoProfile -EndpointName $epName
   ```

   Alternatively, for non-domain-joined accounts, you can provide Active Directory credentials, and then select an AWS role to which the user has access, as shown in the following line. This
is useful if you have different Active Directory user accounts to differentiate roles within your organization (for example, administration functions).

```powershell
PS > $credential = Get-Credential -Message "Enter the domain credentials for the endpoint"
PS > Set-AWSSamlRoleProfile -EndpointName $epName -NetworkCredential $credential -StoreAs SAMLDemoProfile
```

4. In either case, the Set-AWSSamlRoleProfile cmdlet prompts you to choose which role should be stored in the profile. The following example shows two available roles: ADFS-Dev, and ADFS-Production. The IAM roles are associated with your AD login credentials by the AD FS administrator.

```
Select Role
Select the role to be assumed when this profile is active
```

Alternatively, you can specify a role without the prompt, by entering the RoleARN, PrincipalARN, and optional NetworkCredential parameters. If the specified role is not listed in the assertion returned by authentication, the user is prompted to choose from available roles.

```powershell
PS > $params = @{ "NetworkCredential"=$credential, "PrincipalARN"="{arn:aws:iam::012345678912:saml-provider/ADFS}", "RoleARN"="{arn:aws:iam::012345678912:role/ADFS-Dev}" }
PS > $epName | Set-AWSSamlRoleProfile @params -StoreAs SAMLDemoProfile1 -Verbose
```

5. You can create profiles for all roles in a single command by adding the StoreAllRoles parameter, as shown in the following code. Note that the role name is used as the profile name.

```powershell
PS > Set-AWSSamlRoleProfile -EndpointName $epName -StoreAllRoles ADFS-Dev ADFS-Production
```

**How to Use Role Profiles to Run Cmdlets that Require AWS Credentials**

To run cmdlets that require AWS credentials, you can use role profiles defined in the AWS shared credential file. Provide the name of a role profile to Set-AWSCredential (or as the value for...
any ProfileName parameter in the AWS Tools for PowerShell) to get temporary AWS credentials automatically for the role that is described in the profile.

Although you use only one role profile at a time, you can switch between profiles within a shell session. The Set-AWSCredential cmdlet does not authenticate and get credentials when you run it by itself; the cmdlet records that you want to use a specified role profile. Until you run a cmdlet that requires AWS credentials, no authentication or request for credentials occurs.

You can now use the temporary AWS credentials that you obtained with the SAMLDemoProfile profile to work with AWS service APIs. The following sections show examples of how to use role profiles.

**Example 1: Set a Default Role with Set-AWSCredential**

This example sets a default role for a AWS Tools for PowerShell session by using Set-AWSCredential. Then, you can run cmdlets that require credentials, and are authorized by the specified role. This example lists all Amazon Elastic Compute Cloud instances in the US West (Oregon) Region that are associated with the profile you specified with the Set-AWSCredential cmdlet.

```powershell
PS > Set-AWSCredential -ProfileName SAMLDemoProfile
PS > Get-EC2Instance -Region us-west-2 | Format-Table -Property Instances,GroupNames

<table>
<thead>
<tr>
<th>Instances</th>
<th>GroupNames</th>
</tr>
</thead>
<tbody>
<tr>
<td>{TestInstance1}</td>
<td>{default}</td>
</tr>
<tr>
<td>{TestInstance2}</td>
<td>{}</td>
</tr>
<tr>
<td>{TestInstance3}</td>
<td>{launch-wizard-6}</td>
</tr>
<tr>
<td>{TestInstance4}</td>
<td>{default}</td>
</tr>
<tr>
<td>{TestInstance5}</td>
<td>{}</td>
</tr>
<tr>
<td>{TestInstance6}</td>
<td>{AWS-OpsWorks-Default-Server}</td>
</tr>
</tbody>
</table>
```

**Example 2: Change Role Profiles During a PowerShell Session**

This example lists all available Amazon S3 buckets in the AWS account of the role associated with the SAMLDemoProfile profile. The example shows that although you might have been using another profile earlier in your AWS Tools for PowerShell session, you can change profiles by specifying a different value for the -ProfileName parameter with cmdlets that support it. This is a common task for administrators who manage Amazon S3 from the PowerShell command line.
Note that the Get-S3Bucket cmdlet specifies the name of the profile created by running the Set-AWSSamlRoleProfile cmdlet. This command could be useful if you had set a role profile earlier in your session (for example, by running the Set-AWSCredential cmdlet) and wanted to use a different role profile for the Get-S3Bucket cmdlet. The profile manager makes temporary credentials available to the Get-S3Bucket cmdlet.

Although the credentials expire after 1 hour (a limit enforced by STS), AWS Tools for PowerShell automatically refreshes the credentials by requesting a new SAML assertion when the tool detects that the current credentials have expired.

For domain-joined users, this process occurs without interruption, because the current user's Windows identity is used during authentication. For non-domain-joined user accounts, AWS Tools for PowerShell shows a PowerShell credential prompt requesting the user password. The user provides credentials that are used to reauthenticate the user and get a new assertion.

**Example 3: Get Instances in a Region**

The following example lists all Amazon EC2 instances in the Asia Pacific (Sydney) Region that are associated with the account used by the ADFS-Production profile. This is a useful command for returning all Amazon EC2 instances in a region.

Note that the Get-S3Bucket cmdlet specifies the name of the profile created by running the Set-AWSSamlRoleProfile cmdlet. This command could be useful if you had set a role profile earlier in your session (for example, by running the Set-AWSCredential cmdlet) and wanted to use a different role profile for the Get-S3Bucket cmdlet. The profile manager makes temporary credentials available to the Get-S3Bucket cmdlet.

Although the credentials expire after 1 hour (a limit enforced by STS), AWS Tools for PowerShell automatically refreshes the credentials by requesting a new SAML assertion when the tool detects that the current credentials have expired.

For domain-joined users, this process occurs without interruption, because the current user's Windows identity is used during authentication. For non-domain-joined user accounts, AWS Tools for PowerShell shows a PowerShell credential prompt requesting the user password. The user provides credentials that are used to reauthenticate the user and get a new assertion.

**Example 3: Get Instances in a Region**

The following example lists all Amazon EC2 instances in the Asia Pacific (Sydney) Region that are associated with the account used by the ADFS-Production profile. This is a useful command for returning all Amazon EC2 instances in a region.
Additional Reading

For general information about how to implement federated API access, see How to Implement a General Solution for Federated API/CLI Access Using SAML 2.0.

For support questions or comments, visit the AWS Developer Forums for PowerShell Scripting or .NET Development.

Cmdlet discovery and aliases

This section shows you how to list services that are supported by the AWS Tools for PowerShell, how to show the set of cmdlets provided by the AWS Tools for PowerShell in support of those services, and how to find alternative cmdlet names (also called aliases) to access those services.

Cmdlet Discovery

All AWS service operations (or APIs) are documented in the API Reference Guide for each service. For example, see the IAM API Reference. There is, in most cases, a one-to-one correspondence between an AWS service API and an AWS PowerShell cmdlet. To get the cmdlet name that corresponds to an AWS service API name, run the AWS Get-AWSCmdletName cmdlet with the -ApiOperation parameter and the AWS service API name. For example, to get all possible cmdlet names that are based on any available DescribeInstances AWS service API, run the following command:

```
PS > Get-AWSCmdletName -ApiOperation DescribeInstances
```

<table>
<thead>
<tr>
<th>CmdletName</th>
<th>ServiceOperation</th>
<th>ServiceName</th>
<th>CmdletNounPrefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get-EC2Instance</td>
<td>DescribeInstances</td>
<td>Amazon Elastic Compute Cloud</td>
<td>EC2</td>
</tr>
<tr>
<td>Get-GMLInstance</td>
<td>DescribeInstances</td>
<td>Amazon GameLift Service</td>
<td>GML</td>
</tr>
</tbody>
</table>

The -ApiOperation parameter is the default parameter, so you can omit the parameter name. The following example is equivalent to the previous one:

```
PS > Get-AWSCmdletName DescribeInstances
```
If you know the names of both the API and the service, you can include the -Service parameter along with either the cmdlet noun prefix or part of the AWS service name. For example, the cmdlet noun prefix for Amazon EC2 is EC2. To get the cmdlet name that corresponds to the DescribeInstances API in the Amazon EC2 service, run one of the following commands. They are all result in the same output:

```
PS > Get-AWSCmdletName -ApiOperation DescribeInstances -Service EC2
PS > Get-AWSCmdletName -ApiOperation DescribeInstances -Service Compute
PS > Get-AWSCmdletName -ApiOperation DescribeInstances -Service "Compute Cloud"
```

<table>
<thead>
<tr>
<th>CmdletName</th>
<th>ServiceOperation</th>
<th>ServiceName</th>
<th>CmdletNounPrefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get-EC2Instance</td>
<td>DescribeInstances</td>
<td>Amazon Elastic Compute Cloud</td>
<td>EC2</td>
</tr>
</tbody>
</table>

Parameter values in these commands are case-insensitive.

If you do not know the name of either the desired AWS service API or the AWS service, you can use the -ApiOperation parameter, along with the pattern to match, and the -MatchWithRegex parameter. For example, to get all available cmdlet names that contain SecurityGroup, run the following command:

```
PS > Get-AWSCmdletName -ApiOperation SecurityGroup -MatchWithRegex
```

<table>
<thead>
<tr>
<th>CmdletName</th>
<th>ServiceOperation</th>
<th>ServiceName</th>
<th>CmdletNounPrefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approve-ECCacheSecurityGroupIngress</td>
<td>AuthorizeCacheSecurityGroupIngress</td>
<td>Amazon ElastiCache</td>
<td>EC</td>
</tr>
<tr>
<td>Get-ECCacheSecurityGroup</td>
<td>DescribeCacheSecurityGroups</td>
<td>Amazon ElastiCache</td>
<td>EC</td>
</tr>
<tr>
<td>New-ECCacheSecurityGroup</td>
<td>CreateCacheSecurityGroup</td>
<td>Amazon ElastiCache</td>
<td>EC</td>
</tr>
<tr>
<td>Remove-ECCacheSecurityGroup</td>
<td>DeleteCacheSecurityGroup</td>
<td>Amazon ElastiCache</td>
<td>EC</td>
</tr>
<tr>
<td>Revoke-ECCacheSecurityGroupIngress</td>
<td>RevokeCacheSecurityGroupIngress</td>
<td>Amazon ElastiCache</td>
<td>EC</td>
</tr>
<tr>
<td>Add-EC2SecurityGroupToClientVpnTargetNetwork</td>
<td>ApplySecurityGroupsToClientVpnTargetNetwork</td>
<td>Amazon Elastic Compute Cloud</td>
<td>EC2</td>
</tr>
<tr>
<td>Get-EC2SecurityGroup</td>
<td>DescribeSecurityGroups</td>
<td>Amazon Elastic Compute Cloud</td>
<td>EC2</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
<td>Service</td>
<td>Module</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>Get-EC2SecurityGroupReference</td>
<td>DescribeSecurityGroupReferences</td>
<td>Amazon Elastic Compute Cloud</td>
<td>EC2</td>
</tr>
<tr>
<td>Get-EC2StaleSecurityGroup</td>
<td>DescribeStaleSecurityGroups</td>
<td>Amazon Elastic Compute Cloud</td>
<td>EC2</td>
</tr>
<tr>
<td>Grant-EC2SecurityGroupEgress</td>
<td>AuthorizeSecurityGroupEgress</td>
<td>Amazon Elastic Compute Cloud</td>
<td>EC2</td>
</tr>
<tr>
<td>Grant-EC2SecurityGroupIngress</td>
<td>AuthorizeSecurityGroupIngress</td>
<td>Amazon Elastic Compute Cloud</td>
<td>EC2</td>
</tr>
<tr>
<td>New-EC2SecurityGroup</td>
<td>CreateSecurityGroup</td>
<td>Amazon Elastic Compute Cloud</td>
<td>EC2</td>
</tr>
<tr>
<td>Remove-EC2SecurityGroup</td>
<td>DeleteSecurityGroup</td>
<td>Amazon Elastic Compute Cloud</td>
<td>EC2</td>
</tr>
<tr>
<td>Revoke-EC2SecurityGroupEgress</td>
<td>RevokeSecurityGroupEgress</td>
<td>Amazon Elastic Compute Cloud</td>
<td>EC2</td>
</tr>
<tr>
<td>Revoke-EC2SecurityGroupIngress</td>
<td>RevokeSecurityGroupIngress</td>
<td>Amazon Elastic Compute Cloud</td>
<td>EC2</td>
</tr>
<tr>
<td>Update-EC2SecurityGroupRuleEgressDescription</td>
<td>UpdateSecurityGroupRuleDescriptionsEgress</td>
<td>Amazon Elastic Compute Cloud</td>
<td>EC2</td>
</tr>
<tr>
<td>Update-EC2SecurityGroupRuleIngressDescription</td>
<td>UpdateSecurityGroupRuleDescriptionsIngress</td>
<td>Amazon Elastic Compute Cloud</td>
<td>EC2</td>
</tr>
<tr>
<td>Edit-EFSMountTargetSecurityGroup</td>
<td>ModifyMountTargetSecurityGroups</td>
<td>Amazon Elastic File System</td>
<td>EFS</td>
</tr>
<tr>
<td>Get-EFSMountTargetSecurityGroup</td>
<td>DescribeMountTargetSecurityGroups</td>
<td>Amazon Elastic File System</td>
<td>EFS</td>
</tr>
<tr>
<td>Join-ELBSecurityGroupToLoadBalancer</td>
<td>ApplySecurityGroupsToLoadBalancer</td>
<td>Elastic Load Balancing</td>
<td>ELB</td>
</tr>
<tr>
<td>Set-ELB2SecurityGroup</td>
<td>SetSecurityGroups</td>
<td>Elastic Load Balancing V2</td>
<td>ELB2</td>
</tr>
<tr>
<td>Enable-RDSDBSecurityGroupIngress</td>
<td>AuthorizeDBSecurityGroupIngress</td>
<td>Amazon Relational Database Service</td>
<td>RDS</td>
</tr>
<tr>
<td>Get-RDSDBSecurityGroup</td>
<td>DescribeDBSecurityGroups</td>
<td>Amazon Relational Database Service</td>
<td>RDS</td>
</tr>
<tr>
<td>New-RDSDBSecurityGroup</td>
<td>CreateDBSecurityGroup</td>
<td>Amazon Relational Database Service</td>
<td>RDS</td>
</tr>
<tr>
<td>Remove-RDSDBSecurityGroup</td>
<td>DeleteDBSecurityGroup</td>
<td>Amazon Relational Database Service</td>
<td>RDS</td>
</tr>
<tr>
<td>Revoke-RDSDBSecurityGroupIngress</td>
<td>RevokeDBSecurityGroupIngress</td>
<td>Amazon Relational Database Service</td>
<td>RDS</td>
</tr>
<tr>
<td>Approve-RSClusterSecurityGroupIngress</td>
<td>AuthorizeClusterSecurityGroupIngress</td>
<td>Amazon Redshift</td>
<td>RS</td>
</tr>
<tr>
<td>Get-RSClusterSecurityGroup</td>
<td>DescribeClusterSecurityGroups</td>
<td>Amazon Redshift</td>
<td>RS</td>
</tr>
<tr>
<td>New-RSClusterSecurityGroup</td>
<td>CreateClusterSecurityGroup</td>
<td>Amazon Redshift</td>
<td>RS</td>
</tr>
</tbody>
</table>
If you know the name of the AWS service but not the AWS service API, include both the \-MatchWithRegex parameter and the \-Service parameter to scope the search down to a single service. For example, to get all cmdlet names that contain SecurityGroup in only the Amazon EC2 service, run the following command

```
PS > Get-AWSCmdletName -ApiOperation SecurityGroup -MatchWithRegex -Service EC2
```

<table>
<thead>
<tr>
<th>CmdletName</th>
<th>ServiceOperation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add-EC2SecurityGroupToClientVpnTargetNetwork</td>
<td>Amazon Elastic Compute Cloud EC2</td>
</tr>
<tr>
<td>ApplySecurityGroupsToClientVpnTargetNetwork</td>
<td>Amazon Elastic Compute Cloud EC2</td>
</tr>
<tr>
<td>Get-EC2SecurityGroup</td>
<td>DescribeSecurityGroups</td>
</tr>
<tr>
<td>Get-EC2StaleSecurityGroup</td>
<td>DescribeStaleSecurityGroups</td>
</tr>
<tr>
<td>New-EC2SecurityGroup</td>
<td>CreateSecurityGroup</td>
</tr>
<tr>
<td>Remove-EC2SecurityGroup</td>
<td>DeleteSecurityGroup</td>
</tr>
<tr>
<td>Revoke-EC2SecurityGroupEgress</td>
<td>RevokeSecurityGroupEgress</td>
</tr>
<tr>
<td>Revoke-EC2SecurityGroupIngress</td>
<td>RevokeSecurityGroupIngress</td>
</tr>
<tr>
<td>Update-EC2SecurityGroupRuleEgressDescription</td>
<td>UpdateSecurityGroupRuleDescriptionsEgress</td>
</tr>
<tr>
<td>Update-EC2SecurityGroupRuleIngressDescription</td>
<td>UpdateSecurityGroupRuleDescriptionsIngress</td>
</tr>
<tr>
<td>UpdateSecurityGroupRuleDescriptionsIngress</td>
<td>Amazon Elastic Compute Cloud EC2</td>
</tr>
</tbody>
</table>
If you know the name of the AWS Command Line Interface (AWS CLI) command, you can use the -AwsCliCommand parameter and the desired AWS CLI command name to get the name of the cmdlet that's based on the same API. For example, to get the cmdlet name that corresponds to the authorize-security-group-ingress AWS CLI command call in the Amazon EC2 service, run the following command:

```
PS > Get-AWSCmdletName -AwsCliCommand "aws ec2 authorize-security-group-ingress"
```

<table>
<thead>
<tr>
<th>CmdletName</th>
<th>ServiceOperation</th>
<th>ServiceName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grant-EC2SecurityGroupIngress</td>
<td>AuthorizeSecurityGroupIngress</td>
<td>Amazon Elastic Compute Cloud EC2</td>
</tr>
</tbody>
</table>

The Get-AWSCmdletName cmdlet needs only enough of the AWS CLI command name to identify the service and the AWS API.

To get a list of all of the cmdlets in the Tools for PowerShell Core, run the PowerShell Get-Command cmdlet, as shown in the following example.

```
PS > Get-Command -Module AWSPowerShell.NetCore
```

You can run the same command with -Module AWSPowerShell to see the cmdlets in the AWS Tools for Windows PowerShell.

The Get-Command cmdlet generates the list of cmdlets in alphabetical order. Note that by default the list is sorted by PowerShell verb, rather than PowerShell noun.

To sort results by service instead, run the following command:

```
PS > Get-Command -Module AWSPowerShell.NetCore | Sort-Object Noun,Verb
```

To filter the cmdlets that are returned by the Get-Command cmdlet, pipe the output to the PowerShell Select-String cmdlet. For example, to view the set of cmdlets that work with AWS regions, run the following command:

```
PS > Get-Command -Module AWSPowerShell.NetCore | Select-String region
```

Clear-DefaultAWSRegion
Copy-HSM2BackupToRegion
Get-AWSRegion
Get-DefaultAWSRegion
Get-EC2Region
Get-LSRegionList
Get-RDSSourceRegion
Set-DefaultAWSRegion

You can also find cmdlets for a specific service by filtering for the service prefix of cmdlet
nouns. To see the list of available service prefixes, run Get-AWSPowerShellVersion -
ListServiceVersionInfo. The following example returns cmdlets that support the Amazon
CloudWatch Events service.

```
PS > Get-Command -Module AWSPowerShell -Noun CWE*
```

<table>
<thead>
<tr>
<th>CommandType</th>
<th>Name</th>
<th>Version</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cmdlet</td>
<td>Add-CWEResourceTag</td>
<td>3.3.563.1</td>
<td></td>
</tr>
<tr>
<td>AWSPowerShell.NetCore</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Disable-CWEEventSource</td>
<td>3.3.563.1</td>
<td></td>
</tr>
<tr>
<td>AWSPowerShell.NetCore</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Disable-CWERule</td>
<td>3.3.563.1</td>
<td></td>
</tr>
<tr>
<td>AWSPowerShell.NetCore</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Enable-CWEEventSource</td>
<td>3.3.563.1</td>
<td></td>
</tr>
<tr>
<td>AWSPowerShell.NetCore</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Enable-CWERule</td>
<td>3.3.563.1</td>
<td></td>
</tr>
<tr>
<td>AWSPowerShell.NetCore</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Get-CWEEventBus</td>
<td>3.3.563.1</td>
<td></td>
</tr>
<tr>
<td>AWSPowerShell.NetCore</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Get-CWEEventBusList</td>
<td>3.3.563.1</td>
<td></td>
</tr>
<tr>
<td>AWSPowerShell.NetCore</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Get-CWEEventSource</td>
<td>3.3.563.1</td>
<td></td>
</tr>
<tr>
<td>AWSPowerShell.NetCore</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Get-CWEEventSourceList</td>
<td>3.3.563.1</td>
<td></td>
</tr>
<tr>
<td>AWSPowerShell.NetCore</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Get-CWEPartnerEventSource</td>
<td>3.3.563.1</td>
<td></td>
</tr>
<tr>
<td>AWSPowerShell.NetCore</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Get-CWEPartnerEventSourceAccountList</td>
<td>3.3.563.1</td>
<td></td>
</tr>
<tr>
<td>AWSPowerShell.NetCore</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Get-CWEPartnerEventSourceList</td>
<td>3.3.563.1</td>
<td></td>
</tr>
<tr>
<td>AWSPowerShell.NetCore</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Get-CWEResourceTag</td>
<td>3.3.563.1</td>
<td></td>
</tr>
<tr>
<td>AWSPowerShell.NetCore</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cmdlet Naming and Aliases

The cmdlets in the AWS Tools for PowerShell for each service are based on the methods provided by the AWS SDK for the service. However, because of PowerShell's mandatory naming conventions, the name of a cmdlet might be different from the name of the API call or method.
on which it is based. For example, the Get-EC2Instance cmdlet is based on the Amazon EC2DescribeInstances method.

In some cases, the cmdlet name may be similar to a method name, but it may actually perform a different function. For example, the Amazon S3 GetObject method retrieves an Amazon S3 object. However, the Get-S3Object cmdlet returns information about an Amazon S3 object rather than the object itself.

```powershell
PS > Get-S3Object -BucketName text-content -Key aws-tech-docs

ETag          : "df0000002a0fe0000f3c000004EXAMPLE"
BucketName    : aws-tech-docs
Key           : javascript/frameset.js
LastModified  : 6/13/2011 1:24:18 PM
Owner         : Amazon.S3.Model.Owner
Size          : 512
StorageClass  : STANDARD
```

To get an S3 object with the AWS Tools for PowerShell, run the Read-S3Object cmdlet:

```powershell
PS > Read-S3Object -BucketName text-content -Key text-object.txt -file c:\tmp\text-object-download.text

Mode          LastWriteTime            Length Name
----          -------------            ------ ----
-a---         11/5/2012   7:29 PM      20622 text-object-download.text
```

⚠️ **Note**

The cmdlet help for an AWS cmdlet provides the name of the AWS SDK API on which the cmdlet is based.

For more information about standard PowerShell verbs and their meanings, see [Approved Verbs for PowerShell Commands](#).

All AWS cmdlets that use the Remove verb – and the Stop-EC2Instance cmdlet when you add the -Terminate parameter – prompt for confirmation before proceeding. To bypass confirmation, add the -Force parameter to your command.
Important
AWS cmdlets do not support the -WhatIf switch.

Aliases

Setup of the AWS Tools for PowerShell installs an aliases file that contains aliases for many of the AWS cmdlets. You might find these aliases to be more intuitive than the cmdlet names. For example, service names and AWS SDK method names replace PowerShell verbs and nouns in some aliases. An example is the EC2-DescribeInstances alias.

Other aliases use verbs that, though they do not follow standard PowerShell conventions, can be more descriptive of the actual operation. For example, the alias file maps the alias Get-S3Content to the cmdlet Read-S3Object.

```
PS > Set-Alias -Name Get-S3Content -Value Read-S3Object
```

The aliases file is located in the AWS Tools for PowerShell installation directory. To load the aliases into your environment, `dot-source` the file. The following is a Windows-based example.

```
PS > . "C:\Program Files (x86)\AWS Tools\PowerShell\AWSPowershell\AWSAliases.ps1"
```

For a Linux or macOS shell, it might look like this:

```
./.local/share/powershell/Modules/AWSPowerShell.NetCore/3.3.563.1/AWSAliases.ps1
```

To show all AWS Tools for PowerShell aliases, run the following command. This command uses the ? alias for the PowerShell Where-Object cmdlet and the Source property to filter for only aliases that come from the AWSPowerShell.NetCore module.

```
PS > Get-Alias | ? Source -like "AWSPowerShell.NetCore"
```

<table>
<thead>
<tr>
<th>CommandType</th>
<th>Name</th>
<th>Version</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias</td>
<td>Add-ASInstances</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>AWSPowerShell</td>
<td>Add-CTTag</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>Alias</td>
<td>Description</td>
<td>Version</td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------------------------------------</td>
<td>----------</td>
<td></td>
</tr>
<tr>
<td>Add-DPTags</td>
<td>Add DPTags</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>Add-DSTags</td>
<td>Add DSTags</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>Add-ELBTags</td>
<td>Add ELB Tags</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>Add-EMRTags</td>
<td>Add EMR Tags</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>Add-ESTags</td>
<td>Add ESTags</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>Add-MLTag</td>
<td>Add MLTag</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>Clear-AWSCredentials</td>
<td>Clear AWS Credentials</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>Clear-AWSDefaults</td>
<td>Clear AWS Defaults</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>Dismount-ASInstances</td>
<td>Dismount AS Instances</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>Edit-EC2Hosts</td>
<td>Edit EC2 Hosts</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>Edit-RSClusterIamRoles</td>
<td>Edit RS Cluster Iam Roles</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>Enable-ORGAllFeatures</td>
<td>Enable ORGAll Features</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>Find-CTEvents</td>
<td>Find CTEvents</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>Get-ASACases</td>
<td>Get ASACases</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>Get-ASAccountLimits</td>
<td>Get AS Account Limits</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>Get-ASACommunications</td>
<td>Get ASA Communications</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>Get-ASAServices</td>
<td>Get AS Services</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>Get-ASASeverityLevels</td>
<td>Get ASA Severity Levels</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>Get-ASATrustedAdvisorCheckRefreshStatuses</td>
<td>Get AS Trusted Advisor Check Refresh Statuses</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>Get-ASATrustedAdvisorChecks</td>
<td>Get AS Trusted Advisor Checks</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>Get-ASATrustedAdvisorCheckSummaries</td>
<td>Get AS Trusted Advisor Check Summaries</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>Get-ASLifecycleHooks</td>
<td>Get AS Lifecycle Hooks</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>Alias</td>
<td>Get-ASLifecycleHookTypes</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------------------------------------</td>
<td>------------</td>
<td></td>
</tr>
<tr>
<td>AWSPowerShell</td>
<td>Get-AWSCredentials</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>AWSPowerShell</td>
<td>Get-CDAApplications</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>AWSPowerShell</td>
<td>Get-CDDeployments</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>AWSPowerShell</td>
<td>Get-CFCloudFrontOriginAccessIdentities</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>AWSPowerShell</td>
<td>Get-CFDistributions</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>AWSPowerShell</td>
<td>Get-CFGConfigRules</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>AWSPowerShell</td>
<td>Get-CFGConfigurationRecorders</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>AWSPowerShell</td>
<td>Get-CFGDeliveryChannels</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>AWSPowerShell</td>
<td>Get-CFInvalidations</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>AWSPowerShell</td>
<td>Get-CFNAccountLimits</td>
<td>3.3.343.0</td>
<td></td>
</tr>
<tr>
<td>AWSPowerShell</td>
<td>Get-CFNStackEvents</td>
<td>3.3.343.0</td>
<td></td>
</tr>
</tbody>
</table>

To add your own aliases to this file, you might need to raise the value of PowerShell's $MaximumAliasCount preference variable to a value greater than 5500. The default value is 4096; you can raise it to a maximum of 32768. To do this, run the following.

```
PS > $MaximumAliasCount = 32768
```

To verify that your change was successful, enter the variable name to show its current value.

```
PS > $MaximumAliasCount
32768
```
Pipelining and $AWSHistory

For AWS service calls that return collections, the objects within the collection are enumerated to the pipeline. Result objects that contain additional fields beyond the collection and which are not paging control fields have these fields added as Note properties for the calls. These Note properties are logged in the new $AWSHistory session variable, should you need to access this data. The $AWSHistory variable is described in the next section.

ℹ️ Note

In versions of the Tools for Windows PowerShell prior to v1.1, the collection object itself was emitted, which required the use of foreach {$_.getenumerator()} to continue pipelining.

Examples

The following example returns a list of AWS Regions and your Amazon EC2 machine images (AMIs) in each Region.

```powershell
PS > Get-AWSRegion | % { Echo $_.Name; Get-EC2Image -Owner self -Region $_ }
```

The following example stops all Amazon EC2 instances in the current default region.

```powershell
PS > Get-EC2Instance | Stop-EC2Instance
```

Because collections enumerate to the pipeline, the output from a given cmdlet might be $null, a single object, or a collection. If it is a collection, you can use the .Count property to determine the size of the collection. However, the .Count property is not present when only a single object is emitted. If your script needs to determine, in a consistent way, how many objects were emitted, you can check the EmittedObjectsCount property of the last command value in $AWSHistory.

$AWSHistory

To better support pipelining, output from AWS cmdlets is not reshaped to include the service response and result instances as Note properties on the emitted collection object. Instead, for those calls that emit a single collection as output, the collection is now enumerated to the
PowerShell pipeline. This means that the AWS SDK response and result data cannot exist in the pipe, because there is no containing collection object to which it can be attached.

Although most users probably won't need this data, it can be useful for diagnostic purposes, because you can see exactly what was sent to and received from the underlying AWS service calls made by the cmdlet.

Starting with version 1.1, this data and more is now available in a new shell variable named $AWSHistory. This variable maintains a record of AWS cmdlet invocations and the service responses that were received for each invocation. Optionally, this history can be configured to also record the service requests that each cmdlet made. Additional useful data, such as the overall execution time of the cmdlet, can also be obtained from each entry. For security reasons, requests and responses that contain sensitive data aren't recorded by default. However, the history can be configured to override this behavior if needed. For more information, see the Set-AWSHistoryConfiguration cmdlet shown below.

Each entry in the $AWSHistory.Commands list is of type AWSCmdletHistory. This type has the following useful members:

**CmdletName**

Name of the cmdlet.

**CmdletStart**

DateTime that the cmdlet was run.

**CmdletEnd**

DateTime that the cmdlet finished all processing.

**Requests**

If request recording is enabled, list of last service requests.

**Responses**

List of last service responses received.

**LastServiceResponse**

Helper to return the most recent service response.

**LastServiceRequest**

Helper to return the most recent service request, if available.
Note that the $AWSHistory variable is not created until an AWS cmdlet making a service call is used. It evaluates to $null until that time.

**Note**

Earlier versions of the Tools for Windows PowerShell emitted data related to service responses as Notes properties on the returned object. These are now found on the response entries that are recorded for each invocation in the list.

### Set-AWSHistoryConfiguration

A cmdlet invocation can hold zero or more service request and response entries. To limit memory impact, the $AWSHistory list keeps a record of only the last five cmdlet executions by default; and for each, the last five service responses (and if enabled, last five service requests). You can change these default limits by running the Set-AWSHistoryConfiguration cmdlet. It allows you to both control the size of the list, and whether service requests are also logged:

```
PS > Set-AWSHistoryConfiguration -MaxCmdletHistory <value> -MaxServiceCallHistory <value> -RecordServiceRequests -IncludeSensitiveData
```

All parameters are optional.

The MaxCmdletHistory parameter sets the maximum number of cmdlets that can be tracked at any time. A value of 0 turns off recording of AWS cmdlet activity. The MaxServiceCallHistory parameter sets the maximum number of service responses (and/or requests) that are tracked for each cmdlet. The RecordServiceRequests parameter, if specified, turns on tracking of service requests for each cmdlet. The IncludeSensitiveData parameter, if specified, turns on tracking of service responses and requests (if tracked) that contain sensitive data for each cmdlet.

If run with no parameters, Set-AWSHistoryConfiguration simply turns off any prior request recording, leaving the current list sizes unchanged.

To clear all entries in the current history list, run the Clear-AWSHistory cmdlet.

### $AWSHistory Examples

Enumerate the details of the AWS cmdlets that are being held in the list to the pipeline.
Access the details of the last AWS cmdlet that was run:

```powershell
PS > $AWSHistory.Commands
```

Access the details of the last service response received by the last AWS cmdlet that was run. If an AWS cmdlet is paging output, it may make multiple service calls to obtain either all data or the maximum amount of data (determined by parameters on the cmdlet):

```powershell
PS > $AWSHistory.LastServiceResponse
```

Access the details of the last request made (again, a cmdlet may make more than one request if it is paging on the user's behalf). Yields $null unless service request tracing is enabled:

```powershell
PS > $AWSHistory.LastServiceRequest
```

**Automatic Page-to-Completion for Operations that Return Multiple Pages**

For service APIs that impose a default maximum object return count for a given call or that support pageable result sets, all cmdlets "page-to-completion" by default. Each cmdlet makes as many calls as necessary on your behalf to return the complete data set to the pipeline.

In the following example, which uses `Get-S3Object`, the `$c` variable contains `S3Object` instances for every key in the bucket `test`, potentially a very large data set:

```powershell
PS > $c = Get-S3Object -BucketName test
```

If you want to retain control of the amount of data returned, you can use parameters on the individual cmdlets (for example, `MaxKey` on `Get-S3Object`) or you can explicitly handle paging yourself by using a combination of paging parameters on the cmdlets, and data placed in the `$AWSHistory` variable to get the service's next token data. The following example uses the `MaxKeys` parameter to limit the number of `S3Object` instances returned to no more than the first 500 found in the bucket:

```powershell
PS > $c = Get-S3Object -BucketName test -MaxKey 500
```
To know if more data was available but not returned, use the $AWSHistory session variable entry that recorded the service calls made by the cmdlet.

If the following expression evaluates to $true, you can find the next marker for the next set of results using $AWSHistory.LastServiceResponse.NextMarker:

```powershell
$AWSHistory.LastServiceResponse -ne $null && $AWSHistory.LastServiceResponse.IsTruncated
```

To manually control paging with Get-S3Object, use a combination of the MaxKey and Marker parameters for the cmdlet and the IsTruncated/NextMarker notes on the last recorded response. In the following example, the variable $c contains up to a maximum of 500 S3Object instances for the next 500 objects that are found in the bucket after the start of the specified key prefix marker.

```
PS > $c = Get-S3Object -BucketName test -MaxKey 500 -Marker $AWSHistory.LastServiceResponse.NextMarker
```

## Credential and profile resolution

### Credentials Search Order

When you run a command, AWS Tools for PowerShell searches for credentials in the following order. It stops when it finds usable credentials.

1. Literal credentials that are embedded as parameters in the command line.

   We strongly recommend using profiles instead of putting literal credentials in your command lines.

2. A specified profile name or profile location.
   - If you specify only a profile name, the command looks for the specified profile in the AWS SDK store and, if that does not exist, the specified profile from the AWS shared credentials file in the default location.
   - If you specify only a profile location, the command looks for the default profile from that credentials file.
   - If you specify both a name and a location, the command looks for the specified profile in that credentials file.
If the specified profile or location is not found, the command throws an exception. Search proceeds to the following steps only if you did not specify a profile or location.

3. Credentials specified by the `-Credential` parameter.

4. The session profile, if one exists.

5. The default profile, in the following order:
   a. The default profile in the AWS SDK store.
   b. The default profile in the AWS shared credentials file.
   c. The AWS PS Default profile in the AWS SDK store.

6. If the command is running on an Amazon EC2 instance that is configured to use an IAM role, the EC2 instance's temporary credentials accessed from the instance profile.

   For more information about using IAM roles for Amazon EC2 instances, see the [AWS SDK for .NET](https://docs.aws.amazon.com/sdk-for-net/latest/developer-guide/welcome.html).

If this search fails to locate the specified credentials, the command throws an exception.

**Additional information about users and roles**

In order to run Tools for PowerShell commands on AWS, you need to have some combination of users, permission sets, and service roles that are appropriate for your tasks.

The specific users, permission sets, and service roles that you create, and the way in which you use them, will depend on your requirements. The following is some additional information about why they might be used and how to create them.

**Users and permission sets**

Although it's possible to use an IAM user account with long-term credentials to access AWS services, this is no longer a best practice and should be avoided. Even during development, it is a best practice to create users and permission sets in AWS IAM Identity Center and use temporary credentials provided by an identity source.

For development, you can use the user that you created or were given in [Configure tool authentication](https://docs.aws.amazon.com/sdk-for-net/latest/developer-guide/welcome.html). If you have appropriate AWS Management Console permissions, you can also create different permission sets with least privilege for that user or create new users specifically
for development projects, providing permission sets with least privilege. The course of action you choose, if any, depends on your circumstances.

For more information about these users and permissions sets and how to create them, see Authentication and access in the AWS SDKs and Tools Reference Guide and Getting started in the AWS IAM Identity Center User Guide.

Service roles

You can set up an AWS service role to access AWS services on behalf of users. This type of access is appropriate if multiple people will be running your application remotely; for example, on an Amazon EC2 instance that you have created for this purpose.

The process for creating a service role varies depending on the situation, but is essentially the following.

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose Roles, and then choose Create role.
3. Choose AWS service, find and select EC2 (for example), and then choose the EC2 use case (for example).
4. Choose Next and select the appropriate policies for the AWS services that your application will use.

   Warning
   Do NOT choose the AdministratorAccess policy because that policy enables read and write permissions to almost everything in your account.

5. Choose Next. Enter a Role name, Description, and any tags you want.

   You can find information about tags in Controlling access using AWS resource tags in the IAM User Guide.

6. Choose Create role.

You can find high-level information about IAM roles in IAM Identities (users, user groups, and roles) in the IAM User Guide. Find detailed information about roles in the IAM roles topic.
Using legacy credentials

The topics in this section provide information about using long-term or short-term credentials without using AWS IAM Identity Center.

⚠️ Warning
To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as AWS IAM Identity Center.

ℹ️ Note
The information in these topics is for circumstances where you need to obtain and manage short-term or long-term credentials manually. For additional information about short-term and long-term credentials, see Other ways to authenticate in the AWS SDKs and Tools Reference Guide.
For best security practices, use AWS IAM Identity Center, as described in Configure tool authentication.

Important warnings and guidance for credentials

Warnings for credentials

- **Do NOT** use your account's root credentials to access AWS resources. These credentials provide unrestricted account access and are difficult to revoke.
- **Do NOT** put literal access keys or credential information in your commands or scripts. If you do, you create a risk of accidentally exposing your credentials.
- Be aware that any credentials stored in the shared AWS credentials file, are stored in plaintext.
Additional guidance for securely managing credentials

For a general discussion of how to securely manage AWS credentials, see AWS security credentials in the AWS General Reference and Security best practices and use cases in the IAM User Guide. In addition to those discussions, consider the following:

- Create additional users, such as users in IAM Identity Center, and use their credentials instead of using your AWS root user credentials. Credentials for other users can be revoked if necessary or are temporary by nature. In addition, you can apply a policy to each user for access to only certain resources and actions and thereby take a stance of least-privilege permissions.

- Use IAM roles for tasks for Amazon Elastic Container Service (Amazon ECS) tasks.

- Use IAM roles for applications that are running on Amazon EC2 instances.

Topics

- Using AWS Credentials
- Shared Credentials in AWS Tools for PowerShell

Using AWS Credentials

Each AWS Tools for PowerShell command must include a set of AWS credentials, which are used to cryptographically sign the corresponding web service request. You can specify credentials per command, per session, or for all sessions.

⚠️ Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as AWS IAM Identity Center.

ℹ️ Note

The information in this topic is for circumstances where you need to obtain and manage short-term or long-term credentials manually. For additional information about short-
term and long-term credentials, see Other ways to authenticate in the AWS SDKs and Tools Reference Guide.

For best security practices, use AWS IAM Identity Center, as described in Configure tool authentication.

As a best practice, to avoid exposing your credentials, do not put literal credentials in a command. Instead, create a profile for each set of credentials that you want to use, and store the profile in either of two credential stores. Specify the correct profile by name in your command, and the AWS Tools for PowerShell retrieves the associated credentials. For a general discussion of how to safely manage AWS credentials, see Best Practices for Managing AWS Access Keys in the Amazon Web Services General Reference.

Note

You need an AWS account to get credentials and use the AWS Tools for PowerShell. To create an AWS account, see Getting started: Are you a first-time AWS user? in the AWS Account Management Reference Guide.

Topics

- Credentials Store Locations
- Managing Profiles
- Specifying Credentials
- Credentials Search Order
- Credential Handling in AWS Tools for PowerShell Core

Credentials Store Locations

The AWS Tools for PowerShell can use either of two credentials stores:

- The AWS SDK store, which encrypts your credentials and stores them in your home folder. In Windows, this store is located at: C:\Users\username\AppData\Local\AWSToolkit\RegisteredAccounts.json.

  The AWS SDK for .NET and Toolkit for Visual Studio can also use the AWS SDK store.
• The shared credentials file, which is also located in your home folder, but stores credentials as plain text.

By default, the credentials file is stored here:

• On Windows: `C:\Users\username\.aws\credentials`

• On Mac/Linux: `~/.aws/credentials`

The AWS SDKs and the AWS Command Line Interface can also use the credentials file. If you're running a script outside of your AWS user context, be sure that the file that contains your credentials is copied to a location where all user accounts (local system and user) can access your credentials.

**Managing Profiles**

Profiles enable you to reference different sets of credentials with AWS Tools for PowerShell. You can use AWS Tools for PowerShell cmdlets to manage your profiles in the AWS SDK store. You can also manage profiles in the AWS SDK store by using the Toolkit for Visual Studio or programmatically by using the AWS SDK for .NET. For directions about how to manage profiles in the credentials file, see Best Practices for Managing AWS Access Keys.

**Add a New profile**

To add a new profile to the AWS SDK store, run the command `Set-AWSCredential`. It stores your access key and secret key in your default credentials file under the profile name you specify.

```powershell
PS > Set-AWSCredential
    -AccessKey AKIA0123456787EXAMPLE
    -SecretKey wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
    -StoreAs MyNewProfile
```

- **-AccessKey**– The access key ID.
- **-SecretKey**– The secret key.
- **-StoreAs**– The profile name, which must be unique. To specify the default profile, use the name `default`.
Update a Profile

The AWS SDK store must be maintained manually. If you later change credentials on the service—for example, by using the IAM console—running a command with the locally stored credentials fails with the following error message:

```
The Access Key Id you provided does not exist in our records.
```

You can update a profile by repeating the `Set-AWSCredential` command for the profile, and passing it the new access and secret keys.

List Profiles

You can check the current list of names with the following command. In this example, a user named Shirley has access to three profiles that are all stored in the shared credentials file (~/.aws/credentials).

```
PS > Get-AWSCredential -ListProfileDetail
```

<table>
<thead>
<tr>
<th>ProfileName</th>
<th>StoreTypeName</th>
<th>ProfileLocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>SharedCredentialsFile</td>
<td>/Users/shirley/.aws/credentials</td>
</tr>
<tr>
<td>production</td>
<td>SharedCredentialsFile</td>
<td>/Users/shirley/.aws/credentials</td>
</tr>
<tr>
<td>test</td>
<td>SharedCredentialsFile</td>
<td>/Users/shirley/.aws/credentials</td>
</tr>
</tbody>
</table>

Remove a Profile

To remove a profile that you no longer require, use the following command.

```
PS > Remove-AWSCredentialProfile -ProfileName an-old-profile-I-do-not-need
```

The `-ProfileName` parameter specifies the profile that you want to delete.

The deprecated command `Clear-AWSCredential` is still available for backward compatibility, but `Remove-AWSCredentialProfile` is preferred.

Specifying Credentials

There are several ways to specify credentials. The preferred way is to identify a profile instead of incorporating literal credentials into your command line. AWS Tools for PowerShell locates the profile using a search order that is described in [Credentials Search Order](#).
On Windows, AWS credentials stored in the AWS SDK store are encrypted with the logged-in Windows user identity. They cannot be decrypted by using another account, or used on a device that's different from the one on which they were originally created. To perform tasks that require the credentials of another user, such as a user account under which a scheduled task will run, set up a credential profile, as described in the preceding section, that you can use when you log in to the computer as that user. Log in as the task-performing user to complete the credential setup steps, and create a profile that works for that user. Then log out and log in again with your own credentials to set up the scheduled task.

**Note**

Use the `-ProfileName` common parameter to specify a profile. This parameter is equivalent to the `-StoredCredentials` parameter in earlier AWS Tools for PowerShell releases. For backward compatibility, `-StoredCredentials` is still supported.

**Default Profile (Recommended)**

All AWS SDKs and management tools can find your credentials automatically on your local computer if the credentials are stored in a profile named `default`. For example, if you have a profile named `default` on the local computer, you don't have to run either the `Initialize-AWSDefaultConfiguration` cmdlet or the `Set-AWSCredential` cmdlet. The tools automatically use the access and secret key data stored in that profile. To use an AWS Region other than your default Region (the results of `Get-DefaultAWSRegion`), you can run `Set-DefaultAWSRegion` and specify a Region.

If your profile is not named `default`, but you want to use it as the default profile for the current session, run `Set-AWSCredential` to set it as the default profile.

Although running `Initialize-AWSDefaultConfiguration` lets you specify a default profile for every PowerShell session, the cmdlet loads credentials from your custom-named profile, but overwrites the default profile with the named profile.

We recommend that you do not run `Initialize-AWSDefaultConfiguration` unless you are running a PowerShell session on an Amazon EC2 instance that was not launched with an instance profile, and you want to set up the credential profile manually. Note that the credential profile in this scenario would not contain credentials. The credential profile that results from running `Initialize-AWSDefaultConfiguration` on an EC2 instance doesn't directly store credentials,
but instead points to instance metadata (that provides temporary credentials that automatically rotate). However, it does store the instance's Region. Another scenario that might require running `Initialize-AWSDefaultConfiguration` occurs if you want to run a call against a Region other than the Region in which the instance is running. Running that command permanently overrides the Region stored in the instance metadata.

```
PS > Initialize-AWSDefaultConfiguration -ProfileName MyProfileName -Region us-west-2
```

**Note**
The default credentials are included in the AWS SDK store under the default profile name. The command overwrites any existing profile with that name.

If your EC2 instance was launched with an instance profile, PowerShell automatically gets the AWS credentials and Region information from the instance profile. You don't need to run `Initialize-AWSDefaultConfiguration`. Running the `Initialize-AWSDefaultConfiguration` cmdlet on an EC2 instance launched with an instance profile isn't necessary, because it uses the same instance profile data that PowerShell already uses by default.

**Session Profile**

Use `Set-AWSCredential` to specify a default profile for a particular session. This profile overrides any default profile for the duration of the session. We recommend this if you want to use a custom-named profile in your session instead of the current default profile.

```
PS > Set-AWSCredential -ProfileName MyProfileName
```

**Note**
In versions of the Tools for Windows PowerShell that are earlier than 1.1, the `Set-AWSCredential` cmdlet did not work correctly, and would overwrite the profile specified by "MyProfileName". We recommend using a more recent version of the Tools for Windows PowerShell.
Command Profile

On individual commands, you can add the -ProfileName parameter to specify a profile that applies to only that one command. This profile overrides any default or session profiles, as shown in the following example.

```
PS > Get-EC2Instance -ProfileName MyProfileName
```

**Note**

When you specify a default or session profile, you can also add a -Region parameter to override a default or session Region. For more information, see [Specify AWS Regions](#). The following example specifies a default profile and Region.

```
PS > Initialize-AWSDefaultConfiguration -ProfileName MyProfileName -Region us-west-2
```

By default, the AWS shared credentials file is assumed to be in the user's home folder (C:\Users\username\.aws on Windows, or ~/.aws on Linux). To specify a credentials file in a different location, include the -ProfileLocation parameter and specify the credentials file path. The following example specifies a non-default credentials file for a specific command.

```
PS > Get-EC2Instance -ProfileName MyProfileName -ProfileLocation C:\aws_service_credentials\credentials
```

**Note**

If you are running a PowerShell script during a time that you are not normally signed in to AWS—for example, you are running a PowerShell script as a scheduled task outside of your normal work hours—add the -ProfileLocation parameter when you specify the profile that you want to use, and set the value to the path of the file that stores your credentials. To be certain that your AWS Tools for PowerShell script runs with the correct account credentials, you should add the -ProfileLocation parameter whenever your script runs in a context or process that does not use an AWS account. You can also copy your credentials file to a location that is accessible to the local system or other account that your scripts use to perform tasks.
Credentials Search Order

When you run a command, AWS Tools for PowerShell searches for credentials in the following order. It stops when it finds usable credentials.

1. Literal credentials that are embedded as parameters in the command line.

   We strongly recommend using profiles instead of putting literal credentials in your command lines.

2. A specified profile name or profile location.
   - If you specify only a profile name, the command looks for the specified profile in the AWS SDK store and, if that does not exist, the specified profile from the AWS shared credentials file in the default location.
   - If you specify only a profile location, the command looks for the default profile from that credentials file.
   - If you specify both a name and a location, the command looks for the specified profile in that credentials file.

   If the specified profile or location is not found, the command throws an exception. Search proceeds to the following steps only if you did not specify a profile or location.

3. Credentials specified by the -Credential parameter.

4. The session profile, if one exists.

5. The default profile, in the following order:
   - The default profile in the AWS SDK store.
   - The default profile in the AWS shared credentials file.
   - The AWS PS Default profile in the AWS SDK store.

6. If the command is running on an Amazon EC2 instance that is configured to use an IAM role, the EC2 instance's temporary credentials accessed from the instance profile.

   For more information about using IAM roles for Amazon EC2 instances, see the AWS SDK for .NET.

If this search fails to locate the specified credentials, the command throws an exception.
Credential Handling in AWS Tools for PowerShell Core

Cmdlets in AWS Tools for PowerShell Core accept AWS access and secret keys or the names of credential profiles when they run, similarly to the AWS Tools for Windows PowerShell. When they run on Windows, both modules have access to the AWS SDK for .NET credential store file (stored in the per-user AppData\Local\AWSToolkit\RegisteredAccounts.json file).

This file stores your keys in encrypted format, and cannot be used on a different computer. It is the first file that the AWS Tools for PowerShell searches for a credential profile, and is also the file where the AWS Tools for PowerShell stores credential profiles. For more information about the AWS SDK for .NET credential store file, see Configuring AWS Credentials. The Tools for Windows PowerShell module does not currently support writing credentials to other files or locations.

Both modules can read profiles from the AWS shared credentials file that is used by other AWS SDKs and the AWS CLI. On Windows, the default location for this file is C:\Users\<userid>\.aws\credentials. On non-Windows platforms, this file is stored at ~/.aws/credentials. The -ProfileLocation parameter can be used to point to a non-default file name or file location.

The SDK credential store holds your credentials in encrypted form by using Windows cryptographic APIs. These APIs are not available on other platforms, so the AWS Tools for PowerShell Core module uses the AWS shared credentials file exclusively, and supports writing new credential profiles to the shared credential file.

The following example scripts that use the Set-AWSCredential cmdlet show the options for handling credential profiles on Windows with either the AWSPowerShell or AWSPowerShell.NetCore modules.

# Writes a new (or updates existing) profile with name "myProfileName" # in the encrypted SDK store file

Set-AWSCredential -AccessKey akey -SecretKey skey -StoreAs myProfileName

# Checks the encrypted SDK credential store for the profile and then # falls back to the shared credentials file in the default location

Set-AWSCredential -ProfileName myProfileName

# Bypasses the encrypted SDK credential store and attempts to load the # profile from the ini-format credentials file "mycredentials" in the # folder C:\MyCustomPath
Set-AWSCredential -ProfileName myProfileName -ProfileLocation C:\MyCustomPath\mycredentials

The following examples show the behavior of the **AWSPowerShell.NetCore** module on the Linux or macOS operating systems.

- # Writes a new (or updates existing) profile with name "myProfileName" # in the default shared credentials file ~/.aws/credentials
  
  Set-AWSCredential -AccessKey akey -SecretKey skey -StoreAs myProfileName

- # Writes a new (or updates existing) profile with name "myProfileName" # into an ini-format credentials file "~/mycustompath/mycredentials"

  Set-AWSCredential -AccessKey akey -SecretKey skey -StoreAs myProfileName -ProfileLocation ~/mycustompath/mycredentials

- # Reads the default shared credential file looking for the profile "myProfileName"

  Set-AWSCredential -ProfileName myProfileName

- # Reads the specified credential file looking for the profile "myProfileName"

  Set-AWSCredential -ProfileName myProfileName -ProfileLocation ~/mycustompath/mycredentials

**Shared Credentials in AWS Tools for PowerShell**

The Tools for Windows PowerShell support the use of the AWS shared credentials file, similarly to the AWS CLI and other AWS SDKs. The Tools for Windows PowerShell now support reading and writing of basic, session, and assume role credential profiles to both the .NET credentials file and the AWS shared credential file. This functionality is enabled by a new Amazon.Runtime.CredentialManagement namespace.

⚠️ **Warning**

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as [AWS IAM Identity Center](https://aws.amazon.com/iam/).
The new profile types and access to the AWS shared credential file are supported by the following parameters that have been added to the credentials-related cmdlets, `Initialize-AWSDefaultConfiguration`, `New-AWSCredential`, and `Set-AWSCredential`. In service cmdlets, you can refer to your profiles by adding the common parameter, `-ProfileName`.

### Using an IAM Role with AWS Tools for PowerShell

The AWS shared credential file enables additional types of access. For example, you can access your AWS resources by using an IAM role instead of the long term credentials of an IAM user. To do this, you must have a standard profile that has permissions to assume the role. When you tell the AWS Tools for PowerShell to use a profile that specified a role, the AWS Tools for PowerShell looks up the profile identified by the `SourceProfile` parameter. Those credentials are used to request temporary credentials for the role specified by the `RoleArn` parameter. You can optionally require the use of an multi-factor authentication (MFA) device or an `ExternalId` code when the role is assumed by a third party.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExternalId</td>
<td>The user-defined external ID to be used when assuming a role, if required by the role. This is typically only required when you delegate access to your account to a third party. The third party must include the <code>ExternalId</code> as a parameter when assuming the assigned role. For more information, see <a href="#">How to Use an External ID When Granting Access to Your</a></td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MfaSerial</td>
<td>The MFA serial number to be used when assuming a role, if required by the role. For more information, see Using Multi-Factor Authentication (MFA) in AWS in the IAM User Guide.</td>
</tr>
<tr>
<td>RoleArn</td>
<td>The ARN of the role to assume for assume role credentials. For more information about creating and using roles, see IAM Roles in the IAM User Guide.</td>
</tr>
<tr>
<td>SourceProfile</td>
<td>The name of the source profile to be used by assume role credentials. The credentials found in this profile are used to assume the role specified by the RoleArn parameter.</td>
</tr>
</tbody>
</table>

**Setup of profiles for assuming a role**

The following is an example showing how to set up a source profile that enables directly assuming an IAM role.

The first command creates a source profile that is referenced by the role profile. The second command creates the role profile that which role to assume. The third command shows the credentials for the role profile.

```
PS > Set-AWSCredential -StoreAs my_source_profile -AccessKey access_key_id -SecretKey secret_key
PS > Set-AWSCredential -StoreAs my_role_profile -SourceProfile my_source_profile -RoleArn arn:aws:iam::123456789012:role/role-i-want-to-assume
PS > Get-AWSCredential -ProfileName my_role_profile

SourceCredentials      RoleArn
RoleSessionName        Options
------------------------------
```

**Shared Credentials**
To use this role profile with the Tools for Windows PowerShell service cmdlets, add the `-ProfileName` common parameter to the command to reference the role profile. The following example uses the role profile defined in the previous example to access the [Get-S3Bucket](#) cmdlet. AWS Tools for PowerShell looks up the credentials in `my_source_profile`, uses those credentials to call `AssumeRole` on behalf of the user, and then uses those temporary role credentials to call `Get-S3Bucket`.

```
Get-S3Bucket -ProfileName my_role_profile
```

<table>
<thead>
<tr>
<th>CreationDate</th>
<th>BucketName</th>
</tr>
</thead>
<tbody>
<tr>
<td>2/27/2017 8:57:53 AM</td>
<td>4ba3578c-f88f-4d8b-b95f-92a8858dac58-bucket1</td>
</tr>
<tr>
<td>2/27/2017 10:44:37 AM</td>
<td>2091a504-66a9-4d69-8981-aaef812a02c3-bucket2</td>
</tr>
</tbody>
</table>

### Using the Credential Profile Types

To set a credential profile type, understand which parameters provide the information required by the profile type.

<table>
<thead>
<tr>
<th>Credentials Type</th>
<th>Parameters you must use</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic</strong></td>
<td>-AccessKey</td>
</tr>
<tr>
<td><strong>Session</strong></td>
<td>-AccessKey, -SecretKey</td>
</tr>
<tr>
<td><strong>Role</strong></td>
<td>-SourceProfile, -RoleArn</td>
</tr>
</tbody>
</table>
The ProfilesLocation Common Parameter

You can use -ProfileLocation to write to the shared credential file as well as instruct a cmdlet to read from the credential file. Adding the -ProfileLocation parameter controls whether Tools for Windows PowerShell uses the shared credential file or the .NET credential file. The following table describes how the parameter works in Tools for Windows PowerShell.

<table>
<thead>
<tr>
<th>Profile Location Value</th>
<th>Profile Resolution Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>null (not set) or empty</td>
<td>First, search the .NET credential file for a profile with the specified name. If the profile isn't found, search the AWS shared credentials file at (user's home directory) \aws \credentials.</td>
</tr>
<tr>
<td>The path to a file in the AWS shared credential file format</td>
<td>Search only the specified file for a profile with the given name.</td>
</tr>
</tbody>
</table>

Save Credentials to a Credentials File

To write and save credentials to one of the two credential files, run the Set-AWSCredential cmdlet. The following example shows how to do this. The first command uses Set-AWSCredential with -ProfileLocation to add access and secret keys to a profile specified by the -ProfileName parameter. In the second line, run the Get-Content cmdlet to display the contents of the credentials file.

```
PS > Set-AWSCredential -ProfileLocation C:\Users\auser\aws\credentials -ProfileName basic_profile -AccessKey access_key2 -SecretKey secret_key2
PS > Get-Content C:\Users\auser\aws\credentials
aws_access_key_id=access_key2
```
Displaying Your Credential Profiles

Run the `Get-AWSCredential` cmdlet and add the `-ListProfileDetail` parameter to return credential file types and locations, and a list of profile names.

```
PS > Get-AWSCredential -ListProfileDetail

ProfileName         StoreTypeName         ProfileLocation
-----------         -------------         ---------------
source_profile      NetSDKCredentialsFile
assume_role_profile NetSDKCredentialsFile
basic_profile       SharedCredentialsFile C:\Users\auser\.aws\credentials
```

Removing Credential Profiles

To remove credential profiles, run the new `Remove-AWSCredentialProfile` cmdlet. `Clear-AWSCredential` is deprecated, but still available for backward compatibility.

Important Notes

Only `Initialize-AWSDefaultConfiguration`, `New-AWSCredential`, and `Set-AWSCredential` support the parameters for role profiles. You cannot specify the role parameters directly on a command such as `Get-S3Bucket -SourceProfile source_profile_name -RoleArn arn:aws:iam::999999999999:role/role_name`. That does not work because service cmdlets do not directly support the `SourceProfile` or `RoleArn` parameters. Instead, you must store those parameters in a profile, then call the command with the `-ProfileName` parameter.
Work with AWS services in the AWS Tools for PowerShell

This section provides examples of using the AWS Tools for PowerShell to access AWS services. These examples help demonstrate how to use the cmdlets to perform actual AWS tasks. These examples rely on cmdlets that the Tools for PowerShell provides. To see what cmdlets are available, see the AWS Tools for PowerShell Cmdlet Reference.

PowerShell File Concatenation Encoding

Some cmdlets in the AWS Tools for PowerShell edit existing files or records that you have in AWS. An example is `Edit-R53ResourceRecordSet`, which calls the `ChangeResourceRecordSets` API for Amazon Route 53.

When you edit or concatenate files in PowerShell 5.1 or older releases, PowerShell encodes the output in UTF-16, not UTF-8. This can add unwanted characters and create results that are not valid. A hexadecimal editor can reveal the unwanted characters.

To avoid converting file output to UTF-16, you can pipe your command into PowerShell's `Out-File` cmdlet and specify UTF-8 encoding, as shown in the following example:

```
PS > *some file concatenation command* | Out-File filename.txt -Encoding utf8
```

If you are running AWS CLI commands from within the PowerShell console, the same behavior applies. You can pipe the output of an AWS CLI command into `Out-File` in the PowerShell console. Other cmdlets, such as `Export-Csv` or `Export-Clixml`, also have an `Encoding` parameter. For a complete list of cmdlets that have an `Encoding` parameter, and that allow you to correct the encoding of the output of a concatenated file, run the following command:

```
PS > Get-Command -ParameterName "Encoding"
```

Note

PowerShell 6.0 and newer, including PowerShell Core, automatically retains UTF-8 encoding for concatenated file output.
Returned Objects for the PowerShell Tools

To make AWS Tools for PowerShell more useful in a native PowerShell environment, the object returned by a AWS Tools for PowerShell cmdlet is a .NET object, not the JSON text object that is typically returned from the corresponding API in the AWS SDK. For example, Get-S3Bucket emits a Buckets collection, not an Amazon S3 JSON response object. The Buckets collection can be placed in the PowerShell pipeline and interacted with in appropriate ways. Similarly, Get-EC2Instance emits a Reservation .NET object collection, not a DescribeEC2Instances JSON result object. This behavior is by design and enables the AWS Tools for PowerShell experience to be more consistent with idiomatic PowerShell.

The actual service responses are available for you if you need them. They are stored as note properties on the returned objects. For API actions that support paging by using NextToken fields, these are also attached as note properties.

Amazon EC2

This section walks through the steps required to launch an Amazon EC2 instance including how to:

- Retrieve a list of Amazon Machine Images (AMIs).
- Create a key pair for SSH authentication.
- Create and configure an Amazon EC2 security group.
- Launch the instance and retrieve information about it.

Amazon S3

The section walks through the steps required to create a static website hosted in Amazon S3. It demonstrates how to:

- Create and delete Amazon S3 buckets.
- Upload files to an Amazon S3 bucket as objects.
- Delete objects from an Amazon S3 bucket.
- Designate an Amazon S3 bucket as a website.
AWS Lambda and AWS Tools for PowerShell

This section provides a brief overview of the AWS Lambda Tools for PowerShell module and describes the required steps for setting up the module.

Amazon SNS and Amazon SQS

This section walks through the steps required to subscribe an Amazon SQS queue to an Amazon SNS topic. It demonstrates how to:

- Create an Amazon SNS topic.
- Create an Amazon SQS queue.
- Subscribe the queue to the topic.
- Send a message to the topic.
- Receive the message from the queue.

CloudWatch

This section provides an example of how to publish custom data to CloudWatch.

- Publish a Custom Metric to Your CloudWatch Dashboard.

See Also

- Get started with the AWS Tools for Windows PowerShell

Topics

- Amazon S3 and Tools for Windows PowerShell
- Amazon EC2 and Tools for Windows PowerShell
- AWS Lambda and AWS Tools for PowerShell
- Amazon SQS, Amazon SNS and Tools for Windows PowerShell
- CloudWatch from the AWS Tools for Windows PowerShell
Using the ClientConfig parameter in cmdlets

Amazon S3 and Tools for Windows PowerShell

In this section, we create a static website using the AWS Tools for Windows PowerShell using Amazon S3 and CloudFront. In the process, we demonstrate a number of common tasks with these services. This walkthrough is modeled after the Getting Started Guide for Host a Static Website, which describes a similar process using the AWS Management Console.

The commands shown here assume that you have set default credentials and a default region for your PowerShell session. Therefore, credentials and regions are not included in the invocation of the cmdlets.

Note

There is currently no Amazon S3 API for renaming a bucket or object, and therefore, no single Tools for Windows PowerShell cmdlet for performing this task. To rename an object in S3, we recommend that you copy the object to one with a new name, by running the Copy-S3Object cmdlet, and then delete the original object by running the Remove-S3Object cmdlet.

See also

- Work with AWS services in the AWS Tools for PowerShell
- Hosting a Static Website on Amazon S3
- Amazon S3 Console

Topics

- Create an Amazon S3 Bucket, Verify Its Region, and Optionally Remove It
- Configure an Amazon S3 Bucket as a Website and Enable Logging
- Upload Objects to an Amazon S3 Bucket
- Delete Amazon S3 Objects and Buckets
- Upload In-Line Text Content to Amazon S3
Create an Amazon S3 Bucket, Verify Its Region, and Optionally Remove It

Use the New-S3Bucket cmdlet to create a new Amazon S3 bucket. The following examples creates a bucket named website-example. The name of the bucket must be unique across all regions. The example creates the bucket in the us-west-1 region.

```
PS > New-S3Bucket -BucketName website-example -Region us-west-1

<table>
<thead>
<tr>
<th>CreationDate</th>
<th>BucketName</th>
</tr>
</thead>
<tbody>
<tr>
<td>8/16/19 8:45:38 PM</td>
<td>website-example</td>
</tr>
</tbody>
</table>
```

You can verify the region in which the bucket is located using the Get-S3BucketLocation cmdlet.

```
PS > Get-S3BucketLocation -BucketName website-example

<table>
<thead>
<tr>
<th>Value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>us-west-2</td>
<td></td>
</tr>
</tbody>
</table>
```

When you're done with this tutorial, you can use the following line to remove this bucket. We suggest that you leave this bucket in place as we use it in subsequent examples.

```
PS > Remove-S3Bucket -BucketName website-example
```

Note that the bucket removal process can take some time to finish. If you try to re-create a same-named bucket immediately, the New-S3Bucket cmdlet can fail until the old one is completely gone.

See Also

- Work with AWS services in the AWS Tools for PowerShell
- Put Bucket (Amazon S3 Service Reference)
- AWS PowerShell Regions for Amazon S3
Configure an Amazon S3 Bucket as a Website and Enable Logging

Use the Write-S3BucketWebsite cmdlet to configure an Amazon S3 bucket as a static website. The following example specifies a name of index.html for the default content web page and a name of error.html for the default error web page. Note that this cmdlet does not create those pages. They need to be uploaded as Amazon S3 objects.

```
PS > Write-S3BucketWebsite -BucketName website-example -
    WebsiteConfiguration_IndexDocumentSuffix index.html -WebsiteConfiguration_ErrorDocument error.html
RequestId      : A1813E27995FFDDDD
AmazonId2      : T7hlDOeLqA5Q2XfTe8j2q3SLoP3/5XwhUU3RyJBGHU/LnC+CIWLeGgP0MY24xAlI
ResponseStream :
Headers        : {x-amz-id-2, x-amz-request-id, Content-Length, Date...}
Metadata       : {}
ResponseXml    :
```

See Also

- Work with AWS services in the AWS Tools for PowerShell
- Put Bucket Website (Amazon S3 API Reference)
- Put Bucket ACL (Amazon S3 API Reference)

Upload Objects to an Amazon S3 Bucket

Use the Write-S3Object cmdlet to upload files from your local file system to an Amazon S3 bucket as objects. The example below creates and uploads two simple HTML files to an Amazon S3 bucket, and verifies the existence of the uploaded objects. The -File parameter to Write-S3Object specifies the name of the file in the local file system. The -Key parameter specifies the name that the corresponding object will have in Amazon S3.

Amazon infers the content-type of the objects from the file extensions, in this case, ".html".

```
PS > # Create the two files using here-strings and the Set-Content cmdlet
PS > $index_html = @">> <html>  
  >> <body>  
  >> <p>
```

Configure an Amazon S3 Bucket as a Website and Enable Logging
Hello, World!

```powershell
PS > $index_html | Set-Content index.html
PS > $error_html = @""""html
"""<body
""""p
""""This is an error page.
""""/p
""""/body
""""/html
""""@"

PS > $error_html | Set-Content error.html

# Upload the files to Amazon S3 using a foreach loop
foreach ($f in "index.html", "error.html") {
    Write-S3Object -BucketName website-example -File $f -Key $f -CannedACLName public-read
}

# Verify that the files were uploaded
Get-S3BucketWebsite -BucketName website-example

IndexDocumentSuffix          ErrorDocument
-----------------------------  ------------
index.html                    error.html
```

**Canned ACL Options**

The values for specifying canned ACLs with the Tools for Windows PowerShell are the same as those used by the AWS SDK for .NET. Note, however, that these are different from the values used by the Amazon S3PutObject action. The Tools for Windows PowerShell support the following canned ACLs:

- NoACL
- private
- public-read
- public-read-write
• aws-exec-read
• authenticated-read
• bucket-owner-read
• bucket-owner-full-control
• log-delivery-write

For more information about these canned ACL settings, see Access Control List Overview.

**Note Regarding Multipart Upload**

If you use the Amazon S3 API to upload a file that is larger than 5 GB in size, you need to use multipart upload. However, the Write-S3Object cmdlet provided by the Tools for Windows PowerShell can transparently handle file uploads that are greater than 5 GB.

**Test the Website**

At this point, you can test the website by navigating to it using a browser. URLs for static websites hosted in Amazon S3 follow a standard format.

```
http://<bucket-name>.s3-website-<region>.amazonaws.com
```

For example:

```
http://website-example.s3-website-us-west-1.amazonaws.com
```

**See Also**

• [Work with AWS services in the AWS Tools for PowerShell](#)
• [Put Object (Amazon S3 API Reference)](#)
• [Canned ACLs (Amazon S3 API Reference)](#)

**Delete Amazon S3 Objects and Buckets**

This section describes how to delete the website that you created in preceding sections. You can simply delete the objects for the HTML files, and then delete the Amazon S3 bucket for the site.

First, run the Remove-S3Object cmdlet to delete the objects for the HTML files from the Amazon S3 bucket.
### PS >
```powershell
foreach ( $obj in "index.html", "error.html" ) {
    > Remove-S3Object -BucketName website-example -Key $obj
}
```

- **IsDeleteMarker**
  - `False`

The `False` response is an expected artifact of the way that Amazon S3 processes the request. In this context, it does not indicate an issue.

Now you can run the `Remove-S3Bucket` cmdlet to delete the now-empty Amazon S3 bucket for the site.

```powershell
PS > Remove-S3Bucket -BucketName website-example
```

RequestId : E480ED92A2EC703D
AmazonId2 : k6tqaqC1nMkoeYwbuJXUx1/UDa49BJd6dfLN0Ls1mWYNPHjbc8/Nyvm6AGbWcc2P
Headers    : {x-amz-id-2, x-amz-request-id, Date, Server}
Metadata    : {}
ResponseXml :

In 1.1 and newer versions of the AWS Tools for PowerShell, you can add the `-DeleteBucketContent` parameter to `Remove-S3Bucket`, which first deletes all objects and object versions in the specified bucket before trying to remove the bucket itself. Depending on the number of objects or object versions in the bucket, this operation can take a substantial amount of time. In versions of the Tools for Windows PowerShell older than 1.1, the bucket had to be empty before `Remove-S3Bucket` could delete it.

⚠️ **Note**

Unless you add the `-Force` parameter, AWS Tools for PowerShell prompts you for confirmation before the cmdlet runs.

### See Also

- [Work with AWS services in the AWS Tools for PowerShell](#)
- [Delete Object (Amazon S3 API Reference)](#)

**Delete Amazon S3 Objects and Buckets**
Upload In-Line Text Content to Amazon S3

The Write-S3Object cmdlet supports the ability to upload in-line text content to Amazon S3. Using the -Content parameter (alias -Text), you can specify text-based content that should be uploaded to Amazon S3 without needing to place it into a file first. The parameter accepts simple one-line strings as well as here strings that contain multiple lines.

```
PS > # Specifying content in-line, single line text:
PS > write-s3object mybucket -key myobject.txt -content "file content"

PS > # Specifying content in-line, multi-line text: (note final newline needed to end in-line here-string)
PS > write-s3object mybucket -key myobject.txt -content @"line 1
>> line 2
>> line 3
>> "@"

PS > # Specifying content from a variable: (note final newline needed to end in-line here-string)
PS > $x = @"line 1
>> line 2
>> line 3
>> "@

PS > write-s3object mybucket -key myobject.txt -content $x
```

Amazon EC2 and Tools for Windows PowerShell

You can perform common tasks related to Amazon EC2 using the AWS Tools for PowerShell.

The example commands shown here assume that you have set default credentials and a default region for your PowerShell session. Therefore, we don’t include credentials or region when we invoke the cmdlets. For more information, see Get started with the AWS Tools for Windows PowerShell.

Topics
• Creating a Key Pair
• Create a Security Group Using Windows PowerShell
• Find an Amazon Machine Image Using Windows PowerShell
• Launch an Amazon EC2 Instance Using Windows PowerShell

Creating a Key Pair

The following New-EC2KeyPair example creates a key pair and stores in the PowerShell variable $myPSKeyPair

```
PS > $myPSKeyPair = New-EC2KeyPair -KeyName myPSKeyPair
```

Pipe the key pair object into the Get-Member cmdlet to see the object's structure.

```
PS > $myPSKeyPair | Get-Member
```

TypeName: Amazon.EC2.Model.KeyPair

<table>
<thead>
<tr>
<th>Name</th>
<th>MemberType</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>----</td>
<td>----------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>Equals</td>
<td>Method</td>
<td>bool Equals(System.Object obj)</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>Method</td>
<td>int GetHashCode()</td>
</tr>
<tr>
<td>GetType</td>
<td>Method</td>
<td>type GetType()</td>
</tr>
<tr>
<td>ToString</td>
<td>Method</td>
<td>string ToString()</td>
</tr>
<tr>
<td>KeyFingerprint</td>
<td>Property</td>
<td>System.String KeyFingerprint {get;set;}</td>
</tr>
<tr>
<td>KeyMaterial</td>
<td>Property</td>
<td>System.String KeyMaterial {get;set;}</td>
</tr>
<tr>
<td>KeyName</td>
<td>Property</td>
<td>System.String KeyName {get;set;}</td>
</tr>
</tbody>
</table>

Pipe the key pair object into the Format-List cmdlet to view values of the KeyName, KeyFingerprint, and KeyMaterial members. (The output has been truncated for readability.)

```
PS > $myPSKeyPair | Format-List KeyName, KeyFingerprint, KeyMaterial
```

KeyName : myPSKeyPair
KeyMaterial : ----BEGIN RSA PRIVATE KEY----
MIIEogIBAAKCAQEAkK+ANYUS9c7niNjYfaCn6KYj/D0I6djnFoQE...  
Mz6btoxPcE7EMeH1wySU8nouA59xb1917+VkD74bN9KmNcpa/Mu...  
Zyn4Ve0Q5i1/MpkrRogHq08rigeTeV5Yc31vO0RFFPu0Kz4kcm...  
w3Jg8dKsWn0p10pX7V3sRC02KgJBejQUvBFGi50QK9bm4tXBeC...
The `KeyMaterial` member stores the private key for the key pair. The public key is stored in AWS. You can't retrieve the public key from AWS, but you can verify the public key by comparing the `KeyFingerprint` for the private key to that returned from AWS for the public key.

### Viewing the Fingerprint of Your Key Pair

You can use the `Get-EC2KeyPair` cmdlet to view the fingerprint for your key pair.

```powershell
PS > Get-EC2KeyPair -KeyName myPSKeyPair | format-list KeyName, KeyFingerprint
```

```
KeyName        : myPSKeyPair
```

### Storing Your Private Key

To store the private key to a file, pipe the `KeyFingerprint` member to the `Out-File` cmdlet.

```powershell
PS > $myPSKeyPair.KeyMaterial | Out-File -Encoding ascii myPSKeyPair.pem
```

You must specify `-Encoding ascii` when writing the private key to a file. Otherwise, tools such as `openssl` might not be able to read the file correctly. You can verify that the format of the resulting file is correct by using a command such as the following:

```powershell
daxKIAQmDuMdBhrR1/YMv8itFe5DiLbq7Ga+FDCs85NstBa3h...iuskGkcvgWkcFQKLMrRHoDpPb+OdfSzrjHZDpMVFMa9tT8EdbkEF...3SrNeqZPsxJIXoood3CULJpg75JU5kyWnb0+sDNVHoJiZCULCr0...G6lEfegB95Kj6IK7eVq7K6s+DHclrDeMZWa7K5FRszCuX7jssC...x098axbMzox9TNUp1ZYRJEQ0oJr0w+kcz/BSWb8NIwWLtwmJeEy...1BX9X8WFX/A8VLMxtTlelrKmLkNECyEAwltkV1p0JAFhzh9p7ZFeV...vvVpF0E99bk9phx269PB50x2K0kwCagDMMAYyasWObUlmNu/1...lmwRx7KTeQ7W1J3OlgxA1QNMMkip9c4Tb3q9Vc3t/fPf8vwfJ8C...63g6N6rk2FZH2X162BgeuwUd3eZOS05lp4VudtGcuc8/qae+5C...KXgty9n164pMv+VafXKvzhdLAdy0Khc9TGB9++VMSG5TRD15JYId...gYALEJ7m1jJKpHWAes0hiwm5VmyZpzyGstSj5sFtERlAjiETDH...YAtn143dReP9I7BOVO3n3WfIjk85g1l/00c+j8S5giLaFndWGR...9R9WIKm5BMUcSRRCdOyuwKBgeB0nGG6D0ah4HkvrUkepIbUDTD...AnEM1vXiU5UT78bKinpUihZi59qhgdK/hkOSmWhLZgVikJ5VizBf...dKrBr/yTkvRti31VF77KIV1x3xC5E/BZ+YdZEpWoCzAoGAC/d...TTld5N6opg0XAcQ3wzqoGa9ZMwc5Q9f4bfRc67emkw0ZAAwSsvWR...x302duuy7/smTwWWskDWRK51rUxoMv/VVVaqdzcOajwieNrd1c7c...-----END RSA PRIVATE KEY-----
```
Removing Your Key Pair

You need your key pair to launch and connect to an instance. When you are done using a key pair, you can remove it. To remove the public key from AWS, use the `Remove-EC2KeyPair` cmdlet. When prompted, press `Enter` to remove the key pair.

```
PS > Remove-EC2KeyPair -KeyName myPSKeyPair
```

The variable, `$myPSKeyPair`, still exists in the current PowerShell session and still contains the key pair information. The `myPSKeyPair.pem` file also exists. However, the private key is no longer valid because the public key for the key pair is no longer stored in AWS.

Create a Security Group Using Windows PowerShell

You can use the AWS Tools for PowerShell to create and configure a security group. When you create a security group, you specify whether it is for EC2-Classic or EC2-VPC. The response is the ID of the security group.

If you need to connect to your instance, you must configure the security group to allow SSH traffic (Linux) or RDP traffic (Windows).

Topics

- **Prerequisites**
- **Creating a Security Group for EC2-Classic**
- **Creating a Security Group for EC2-VPC**

Prerequisites

You need the public IP address of your computer, in CIDR notation. You can get the public IP address of your local computer using a service. For example, Amazon provides the following service: [http://checkip.amazonaws.com/](http://checkip.amazonaws.com/) or [https://checkip.amazonaws.com/](https://checkip.amazonaws.amazonaws.com/). To locate another...
service that provides your IP address, use the search phrase "what is my IP address". If you are connecting through an ISP or from behind your firewall without a static IP address, you need to find the range of IP addresses that can be used by your client computers.

⚠️ **Warning**

If you specify `0.0.0.0/0`, you are enabling traffic from any IP addresses in the world. For the SSH and RDP protocols, you might consider this acceptable for a short time in a test environment, but it's unsafe for production environments. In production, be sure to authorize access only from the appropriate individual IP address or range of addresses.

### Creating a Security Group for EC2-Classic

⚠️ **Warning**

We are retiring EC2-Classic on August 15, 2022. We recommend that you migrate from EC2-Classic to a VPC. For more information, see [Migrate from EC2-Classic to a VPC](#) in the Amazon EC2 User Guide or the Amazon EC2 User Guide. Also see the blog post [EC2-Classic Networking is Retiring – Here's How to Prepare](#).

The following example uses the `New-EC2SecurityGroup` cmdlet to create a security group for EC2-Classic.

```
PS > New-EC2SecurityGroup -GroupName myPSSecurityGroup -GroupDescription "EC2-Classic from PowerShell"
sg-0a346530123456789
```

To view the initial configuration of the security group, use the `Get-EC2SecurityGroup` cmdlet.

```
PS > Get-EC2SecurityGroup -GroupNames myPSSecurityGroup
Description         : EC2-Classic from PowerShell
GroupId             : sg-0a346530123456789
GroupName           : myPSSecurityGroup
IpPermissions       : {}
IpPermissionsEgress : {Amazon.EC2.Model.IpPermission}
```
To configure the security group to allow inbound traffic on TCP port 22 (SSH) and TCP port 3389, use the `Grant-EC2SecurityGroupIngress` cmdlet. For example, the following example script shows how you could enable SSH traffic from a single IP address, 203.0.113.25/32.

```powershell
$cidrBlocks = New-Object 'collections.generic.list[string]'
$cidrBlocks.add("203.0.113.25/32")
$ipPermissions = New-Object Amazon.EC2.Model.IpPermission
$ipPermissions.IpProtocol = "tcp"
$ipPermissions.FromPort = 22
$ipPermissions.ToPort = 22
ipPermissions.IpRanges = $cidrBlocks
Grant-EC2SecurityGroupIngress -GroupName myPSSecurityGroup -IpPermissions $ipPermissions
```

To verify the security group was updated, run the `Get-EC2SecurityGroup` cmdlet again. Note that you can't specify an outbound rule for EC2-Classic.

```powershell
PS > Get-EC2SecurityGroup -GroupNames myPSSecurityGroup
```

```powershell
OwnerId             : 123456789012
GroupName           : myPSSecurityGroup
GroupId             : sg-0a346530123456789
Description         : EC2-Classic from PowerShell
IpPermissions       : {Amazon.EC2.Model.IpPermission}
IpPermissionsEgress : {}
VpcId               : 
Tags                : {}
```

To view the security group rule, use the `IpPermissions` property.

```powershell
PS > (Get-EC2SecurityGroup -GroupNames myPSSecurityGroup).IpPermissions
```

```powershell
IpProtocol       : tcp
FromPort         : 22
ToPort           : 22
UserIdGroupPairs : {}
IpRanges         : {203.0.113.25/32}
```
Creating a Security Group for EC2-VPC

The following New-EC2SecurityGroup example adds the -VpcId parameter to create a security group for the specified VPC.

```powershell
PS > $groupid = New-EC2SecurityGroup -VpcId "vpc-da0013b3" -GroupName "myPSSecurityGroup" -GroupDescription "EC2-VPC from PowerShell"
```

To view the initial configuration of the security group, use the Get-EC2SecurityGroup cmdlet. By default, the security group for a VPC contains a rule that allows all outbound traffic. Notice that you can't reference a security group for EC2-VPC by name.

```powershell
PS > Get-EC2SecurityGroup -GroupId sg-5d293231
```

```
OwnerId             : 123456789012
GroupName           : myPSSecurityGroup
GroupId             : sg-5d293231
Description         : EC2-VPC from PowerShell
IpPermissions       : {}
IpPermissionsEgress : {Amazon.EC2.Model.IpPermission}
VpcId               : vpc-da0013b3
Tags                : {}
```

To define the permissions for inbound traffic on TCP port 22 (SSH) and TCP port 3389, use the New-Object cmdlet. The following example script defines permissions for TCP ports 22 and 3389 from a single IP address, 203.0.113.25/32.

```powershell
$ip1 = new-object Amazon.EC2.Model.IpPermission
$ip1.IpProtocol = "tcp"
$ip1.FromPort = 22
$ip1.ToPort = 22
$ip1.IpRanges.Add("203.0.113.25/32")
$ip2 = new-object Amazon.EC2.Model.IpPermission
$ip2.IpProtocol = "tcp"
$ip2.FromPort = 3389
$ip2.ToPort = 3389
$ip2.IpRanges.Add("203.0.113.25/32")
Grant-EC2SecurityGroupIngress -GroupId $groupid -IpPermissions @($ip1, $ip2)
```
To verify the security group has been updated, use the `Get-EC2SecurityGroup` cmdlet again.

```powershell
PS > Get-EC2SecurityGroup -GroupIds sg-5d293231

OwnerId             : 123456789012
GroupName           : myPSSecurityGroup
GroupId             : sg-5d293231
Description         : EC2-VPC from PowerShell
IpPermissions       : {Amazon.EC2.Model.IpPermission}
IpPermissionsEgress : {Amazon.EC2.Model.IpPermission}
VpcId               : vpc-da0013b3
Tags                : {}
```

To view the inbound rules, you can retrieve the `IpPermissions` property from the collection object returned by the previous command.

```powershell
PS > (Get-EC2SecurityGroup -GroupIds sg-5d293231).IpPermissions

IpProtocol       : tcp
FromPort         : 22
ToPort           : 22
UserIdGroupPairs : {}
IpRanges         : {203.0.113.25/32}

IpProtocol       : tcp
FromPort         : 3389
ToPort           : 3389
UserIdGroupPairs : {}
IpRanges         : {203.0.113.25/32}
```

**Find an Amazon Machine Image Using Windows PowerShell**

When you launch an Amazon EC2 instance, you specify an Amazon Machine Image (AMI) to serve as a template for the instance. However, the IDs for the AWS Windows AMIs change frequently because AWS provides new AMIs with the latest updates and security enhancements. You can use the `Get-EC2Image` and `Get-EC2ImageByName` cmdlets to find the current Windows AMIs and get their IDs.

**Topics**

- [Get-EC2Image](#)
- [Get-EC2ImageByName](#)
Get-EC2Image

The Get-EC2Image cmdlet retrieves a list of AMIs that you can use.

Use the -Owner parameter with the array value `amazon, self` so that Get-EC2Image retrieves only AMIs that belong to Amazon or to you. In this context, `you` refers to the user whose credentials you used to invoke the cmdlet.

```
PS > Get-EC2Image -Owner amazon, self
```

You can scope the results using the -Filter parameter. To specify the filter, create an object of type `Amazon.EC2.Model.Filter`. For example, use the following filter to display only Windows AMIs.

```
$platform_values = New-Object 'collections.generic.list[string]' $platform_values.add("windows")
$filter_platform = New-Object Amazon.EC2.Model.Filter -Property @{Name = "platform"; Values = $platform_values}
Get-EC2Image -Owner amazon, self -Filter $filter_platform
```

The following is an example of one of the AMIs returned by the cmdlet; the actual output of the previous command provides information for many AMIs.

```
Architecture : x86_64
BlockDeviceMappings : {/dev/sda1, xvdca, xvdcb, xvdcc...}
CreationDate : 2019-06-12T10:41:31.000Z
Description : Microsoft Windows Server 2019 Full Locale English with SQL Web 2017 AMI provided by Amazon
EnaSupport : True
Hypervisor : xen
ImageId : ami-000226b77608d973b
ImageOwnerAlias : amazon
ImageType : machine
KernelId : 
OwnerId : 801119661308
Platform : Windows
ProductCodes : {}
Public : True
RamdiskId : 
RootDeviceName : /dev/sda1
```
Get-EC2ImageByName

The Get-EC2ImageByName cmdlet enables you to filter the list of AWS Windows AMIs based on the type of server configuration you are interested in.

When run with no parameters, as follows, the cmdlet emits the complete set of current filter names:

```
PS > Get-EC2ImageByName

WINDOWS_2016_BASE
WINDOWS_2016_NANO
WINDOWS_2016_CORE
WINDOWS_2016_CONTAINER
WINDOWS_2016_SQL_SERVER_ENTERPRISE_2016
WINDOWS_2016_SQL_SERVER_STANDARD_2016
WINDOWS_2016_SQL_SERVER_WEB_2016
WINDOWS_2016_SQL_SERVEREXPRESS_2016
WINDOWS_2012R2_BASE
WINDOWS_2012R2_CORE
WINDOWS_2012R2_SQL_SERVER_EXPRESS_2016
WINDOWS_2012R2_SQL_SERVER_STANDARD_2016
WINDOWS_2012R2_SQL_SERVER_WEB_2016
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
```
To narrow the set of images returned, specify one or more filter names using the Names parameter.

PS > Get-EC2ImageByName -Names WINDOWS_2016_CORE

Architecture : x86_64
BlockDeviceMappings : {/dev/sda1, xvdca, xvdcb, xvdcc...}
CreationDate : 2019-08-16T09:36:09.000Z
Description : Microsoft Windows Server 2016 Core Locale English AMI provided by Amazon
EnaSupport : True
Hypervisor : xen
ImageId : ami-06f2a2afca06f15fc
ImageLocation : amazon/Windows_Server-2016-English-Core-Base-2019.08.16
ImageOwnerAlias : amazon
ImageType : machine
KernelId :
Name : Windows_Server-2016-English-Core-Base-2019.08.16
OwnerId : 801119661308
Platform : Windows
ProductCodes : {}
Public : True
RamdiskId :
RootDeviceName : /dev/sda1
RootDeviceType : ebs
SriovNetSupport : simple
State : available
StateReason :
Tags : {}
VirtualizationType : hvm
Launch an Amazon EC2 Instance Using Windows PowerShell

To launch an Amazon EC2 instance, you need the key pair and security group that you created in the previous sections. You also need the ID of an Amazon Machine Image (AMI). For more information, see the following documentation:

- Creating a Key Pair
- Create a Security Group Using Windows PowerShell
- Find an Amazon Machine Image Using Windows PowerShell

⚠️ Important
If you launch an instance that is not within the Free Tier, you are billed after you launch the instance and charged for the time that the instance is running even if it remains idle.

Topics
- Launching an Instance in EC2-Classic
- Launching an Instance in a VPC
- Launching a Spot Instance in a VPC

Launching an Instance in EC2-Classic

⚠️ Warning
We are retiring EC2-Classic on August 15, 2022. We recommend that you migrate from EC2-Classic to a VPC. For more information, see Migrate from EC2-Classic to a VPC in the Amazon EC2 User Guide or the Amazon EC2 User Guide. Also see the blog post EC2-Classic Networking is Retiring – Here's How to Prepare.

The following command creates and launches a single t1.micro instance.

```
PS > New-EC2Instance -ImageId ami-c49c0dac -MinCount 1
```
Your instance is in the pending state initially, but is in the running state after a few minutes. To view information about your instance, use the Get-EC2Instance cmdlet. If you have more than one instance, you can filter the results on the reservation ID using the Filter parameter. First, create an object of type Amazon.EC2.Model.Filter. Next, call Get-EC2Instance that uses the filter, and then displays the Instances property.

```powershell
PS > $reservation = New-Object 'collections.generic.list[string]'  
PS > $reservation.add("r-5caa4371")  
PS > $filter_reservation = New-Object Amazon.EC2.Model.Filter -Property @{Name = "reservation-id"; Values = $reservation}  
PS > (Get-EC2Instance -Filter $filter_reservation).Instances
```

```powershell
AmiLaunchIndex : 0  
Architecture     : x86_64  
BlockDeviceMappings : {/dev/sda1}  
ClientToken      :  
EbsOptimized     : False  
Hypervisor       : xen  
IamInstanceProfile:  
ImageId          : ami-c49c0dac  
InstanceId       : i-5203422c  
InstanceLifecycle:  
InstanceType     : t1.micro  
KernelId         :  
KeyName          : myPSKeyPair  
LaunchTime       : 12/2/2018 3:38:52 PM  
Monitoring       : Amazon.EC2.Model.Monitoring  
NetworkInterfaces: {}  
Placement        : Amazon.EC2.Model.Placement  
Platform         : Windows  
PrivateDnsName   :  
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrivateIpAddress</td>
<td>10.25.1.11</td>
</tr>
<tr>
<td>ProductCodes</td>
<td>{}</td>
</tr>
<tr>
<td>PublicDnsName</td>
<td></td>
</tr>
<tr>
<td>PublicIpAddress</td>
<td>198.51.100.245</td>
</tr>
<tr>
<td>RamdiskId</td>
<td></td>
</tr>
<tr>
<td>RootDeviceName</td>
<td>/dev/sda1</td>
</tr>
<tr>
<td>RootDeviceType</td>
<td>ebs</td>
</tr>
<tr>
<td>SecurityGroups</td>
<td>{myPSSecurityGroup}</td>
</tr>
<tr>
<td>SourceDestCheck</td>
<td>True</td>
</tr>
<tr>
<td>SpotInstanceRequestId</td>
<td></td>
</tr>
<tr>
<td>SriovNetSupport</td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>Amazon.EC2.Model.InstanceState</td>
</tr>
<tr>
<td>StateReason</td>
<td></td>
</tr>
<tr>
<td>StateTransitionReason</td>
<td></td>
</tr>
<tr>
<td>SubnetId</td>
<td></td>
</tr>
<tr>
<td>Tags</td>
<td>{}</td>
</tr>
<tr>
<td>VirtualizationType</td>
<td>hvm</td>
</tr>
<tr>
<td>VpcId</td>
<td></td>
</tr>
</tbody>
</table>

### Launching an Instance in a VPC

The following command creates a single m1.small instance in the specified private subnet. The security group must be valid for the specified subnet.

```powershell
PS > New-EC2Instance 
   -ImageId ami-c49c0dac 
   -MinCount 1 -MaxCount 1 
   -KeyName myPSKeyPair 
   -SecurityGroupId sg-5d293231 
   -InstanceType m1.small 
   -SubnetId subnet-d60013bf
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReservationId</td>
<td>r-b70a0ef1</td>
</tr>
<tr>
<td>OwnerId</td>
<td>123456789012</td>
</tr>
<tr>
<td>RequesterId</td>
<td></td>
</tr>
<tr>
<td>Groups</td>
<td>{}</td>
</tr>
<tr>
<td>GroupName</td>
<td>{}</td>
</tr>
<tr>
<td>Instances</td>
<td>{}</td>
</tr>
</tbody>
</table>

Your instance is in the pending state initially, but is in the running state after a few minutes. To view information about your instance, use the Get-EC2Instance cmdlet. If you have more than one instance, you can filter the results on the reservation ID using the Filter parameter. First,
create an object of type `Amazon.EC2.Model.Filter`. Next, call `Get-EC2Instance` that uses the filter, and then displays the `Instances` property.

```powershell
PS > $reservation = New-Object 'collections.generic.list[string]'
PS > $reservation.add("r-b70a0ef1")
PS > $filter_reservation = New-Object Amazon.EC2.Model.Filter -Property @{Name = "reservation-id"; Values = $reservation}
PS > (Get-EC2Instance -Filter $filter_reservation).Instances
```

```powershell
AmiLaunchIndex        : 0
Architecture          : x86_64
BlockDeviceMappings   : {/dev/sda1}
ClientToken           :
EbsOptimized          : False
Hypervisor            : xen
IamInstanceProfile    :
ImageId               : ami-c49c0dac
InstanceId            : i-5203422c
InstanceLifecycle     :
InstanceType          : m1.small
KernelId              :
KeyName               : myPSKeyPair
LaunchTime            : 12/2/2018 3:38:52 PM
Monitoring            : Amazon.EC2.Model.Monitoring
NetworkInterfaces     : {}
Placement             : Amazon.EC2.Model.Placement
Platform              : Windows
PrivateDnsName        :
PrivateIpAddress      : 10.25.1.11
ProductCodes          : {}
PublicDnsName         :
PublicIpAddress       : 198.51.100.245
RamdiskId             :
RootDeviceName        : /dev/sda1
RootDeviceType        : ebs
SecurityGroups        : {myPSSecurityGroup}
SourceDestCheck       : True
SpotInstanceRequestId :
SriovNetSupport       :
State                 : Amazon.EC2.Model.InstanceState
StateReason           :
StateTransitionReason :
SubnetId              : subnet-d60013bf
```
**Launching a Spot Instance in a VPC**

The following example script requests a Spot Instance in the specified subnet. The security group must be one you created for the VPC that contains the specified subnet.

```powershell
$interface1 = New-Object Amazon.EC2.Model.InstanceNetworkInterfaceSpecification
$interface1.DeviceIndex = 0
$interface1.SubnetId = "subnet-b61f49f0"
$interface1.PrivateIpAddress = "10.0.1.5"
$interface1.Groups.Add("sg-5d293231")
Request-EC2SpotInstance `  
  -SpotPrice 0.007 `  
  -InstanceCount 1 `  
  -Type one-time `  
  -LaunchSpecification_ImageId ami-7527031c `  
  -LaunchSpecification_InstanceType m1.small `  
  -Region us-west-2 `  
  -LaunchSpecification_NetworkInterfaces $interface1
```

**AWS Lambda and AWS Tools for PowerShell**

By using the [AWSLambdaPSCore](#) module, you can develop AWS Lambda functions in PowerShell Core 6.0 using the .NET Core 2.1 runtime. PowerShell developers can manage AWS resources and write automation scripts in the PowerShell environment by using Lambda. PowerShell support in Lambda lets you run PowerShell scripts or functions in response to any Lambda event, such as an Amazon S3 event or Amazon CloudWatch scheduled event. The AWSLambdaPSCore module is a separate AWS module for PowerShell; it is not part of the AWS Tools for PowerShell, nor does installing the AWSLambdaPSCore module install the AWS Tools for PowerShell.

After you install the AWSLambdaPSCore module, you can use any available PowerShell cmdlets—or develop your own—to author serverless functions. The AWS Lambda Tools for PowerShell module includes project templates for PowerShell-based serverless applications, and tools to publish projects to AWS.

AWSLambdaPSCore module support is available in all regions that support Lambda. For more information about supported regions, see the [AWS region table](#).
Prerequisites

The following steps are required before you can install and use the AWSLambdaPSCore module. For more detail about these steps, see Setting Up a PowerShell Development Environment in the AWS Lambda Developer Guide.

- **Install the correct release of PowerShell** – Lambda's support for PowerShell is based on the cross-platform PowerShell Core 6.0 release. You can develop PowerShell Lambda functions on Windows, Linux, or Mac. If you don't have at least this release of PowerShell installed, instructions are available on the Microsoft PowerShell documentation website.

- **Install the .NET Core 2.1 SDK** – Because PowerShell Core is based on .NET Core, the Lambda support for PowerShell uses the same .NET Core 2.1 Lambda runtime for both .NET Core and PowerShell Lambda functions. The Lambda PowerShell publishing cmdlets use the .NET Core 2.1 SDK to create the Lambda deployment package. The .NET Core 2.1 SDK is available from the Microsoft Download Center. Be sure to install the SDK, not the Runtime.

Install the AWSLambdaPSCore Module

After completing the prerequisites, you are ready to install the AWSLambdaPSCore module. Run the following command in a PowerShell Core session.

```
PS> Install-Module AWSLambdaPSCore -Scope CurrentUser
```

You are ready to start developing Lambda functions in PowerShell. For more information about how to get started, see Programming Model for Authoring Lambda Functions in PowerShell in the AWS Lambda Developer Guide.

See Also

- Announcing Lambda Support for PowerShell Core on the AWS Developer Blog
- AWSLambdaPSCore module on the PowerShell Gallery website
- Setting Up a PowerShell Development Environment
- AWS Lambda Tools for Powershell on GitHub
- AWS Lambda Console
Amazon SQS, Amazon SNS and Tools for Windows PowerShell

This section provides examples that show how to:

- Create an Amazon SQS queue and get queue ARN (Amazon Resource Name).
- Create an Amazon SNS topic.
- Give permissions to the SNS topic so that it can send messages to the queue.
- Subscribe the queue to the SNS topic
- Give IAM users or AWS accounts permissions to publish to the SNS topic and read messages from the SQS queue.
- Verify results by publishing a message to the topic and reading the message from the queue.

Create an Amazon SQS queue and get queue ARN

The following command creates an SQS queue in your default region. The output shows the URL of the new queue.

```
PS > New-SQSQueue -QueueName myQueue
https://sqs.us-west-2.amazonaws.com/123456789012/myQueue
```

The following command retrieves the ARN of the queue.

```
PS > Get-SQSQueueAttribute -QueueUrl https://sqs.us-west-2.amazonaws.com/123456789012/myQueue -AttributeName QueueArn
... QueueARN (: arn:aws:sqs:us-west-2:123456789012:myQueue
```

Create an Amazon SNS topic

The following command creates an SNS topic in your default region, and returns the ARN of the new topic.

```
PS > New-SNSTopic -Name myTopic
```
Give permissions to the SNS topic

The following example script creates both an SQS queue and an SNS topic, and grants permissions to the SNS topic so that it can send messages to the SQS queue:

```powershell
# create the queue and topic to be associated
$qurl = New-SQSQueue -QueueName "myQueue"
$topicarn = New-SNSTopic -Name "myTopic"

# get the queue ARN to inject into the policy; it will be returned
# in the output's QueueARN member but we need to put it into a variable
# so text expansion in the policy string takes effect
$qarn = (Get-SQSQueueAttribute -QueueUrl $qurl -AttributeNames "QueueArn").QueueARN

# construct the policy and inject arns
$policy = @"{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Principal": "*",
        "Action": "SQS:SendMessage",
        "Resource": "$qarn",
        "Condition": { " ArnEquals": { "aws:SourceArn": "$topicarn" } }
    }
}
""

# set the policy
Set-SQSQueueAttribute -QueueUrl $qurl -Attribute @{ Policy=$policy }
```

Subscribe the queue to the SNS topic

The following command subscribes the queue myQueue to the SNS topic myTopic, and returns the Subscription ID:

```powershell
arn:aws:sns:us-west-2:123456789012:myTopic:f8ff77c6-e719-4d70-8e5c-a54d41feb754
```
**Give permissions**

The following command grants permission to perform the `sns:Publish` action on the topic `myTopic`:

```
```

The following command grants permission to perform the `sqs:ReceiveMessage` and `sqs:DeleteMessage` actions on the queue `myQueue`:

```
PS > Add-SQSPermission ` `-QueueUrl https://sqs.us-west-2.amazonaws.com/123456789012/myQueue `-AWSAccountId "123456789012" `-Label queue-permission `-ActionName SendMessage, ReceiveMessage
```

**Verify results**

The following command tests your new queue and topic by publishing a message to the SNS topic `myTopic` and returns the `MessageId`:

```
728180b6-f62b-49d5-b4d3-3824bb2e77f4
```

The following command retrieves the message from the SQS queue `myQueue` and displays it.

```
Attributes : {}
Body : {
    "Type" : "Notification",
    "MessageId" : "491c687d-b78d-5c48-b7a0-3d8d769ee91b",
    "Message": "Have A Nice Day!"
}
```
### CloudWatch from the AWS Tools for Windows PowerShell

This section shows an example of how to use the Tools for Windows PowerShell to publish custom metric data to CloudWatch.

This example assumes that you have set default credentials and a default region for your PowerShell session.

#### Publish a Custom Metric to Your CloudWatch Dashboard

The following PowerShell code initializes an CloudWatch MetricDatum object and posts it to the service. You can see the result of this operation by navigating to the [CloudWatch console](https://console.aws.amazon.com/cloudwatch/home).
$dat = New-Object Amazon.CloudWatch.Model.MetricDatum
$dat.Timestamp = (Get-Date).ToUniversalTime()
$dat.MetricName = "New Posts"
$dat.Unit = "Count"
$dat.Value = ".50"
Write-CWMetricData -Namespace "Usage Metrics" -MetricData $dat

Note the following:

- The date-time information that you use to initialize $dat.Timestamp must be in Universal Time (UTC).
- The value that you use to initialize $dat.Value can be either a string value enclosed in quotes, or a numeric value (no quotes). The example shows a string value.

**See Also**

- Work with AWS services in the AWS Tools for PowerShell
- AmazonCloudWatchClient.PutMetricData (.NET SDK Reference)
- MetricDatum (Service API Reference)
- Amazon CloudWatch Console

**Using the ClientConfig parameter in cmdlets**

The ClientConfig parameter can be used to specify certain configuration settings when you connect to a service. Most of the possible properties of this parameter are defined in the Amazon.Runtime.ClientConfig class, which is inherited into the APIs for AWS services. For an example of simple inheritance, see the Amazon.Keyspaces.AmazonKeyspacesConfig class. In addition, some services define additional properties that are appropriate only for that service. For an example of additional properties that have been defined, see the Amazon.S3.AmazonS3Config class, specifically the ForcePathStyle property.

**Using the ClientConfig parameter**

To use the ClientConfig parameter, you can specify it on the command line as a ClientConfig object or use PowerShell splatting to pass a collection of parameter values to a command as a unit. These methods are shown in the following examples. The examples assume that the
AWS. Tools.S3 module has been installed and imported, and that you have a [default] credentials profile with appropriate permissions.

**Defining a ClientConfig object**

```powershell
$s3Config = New-Object -TypeName Amazon.S3.AmazonS3Config
$s3Config.ForcePathStyle = $true
$s3Config.Timeout = [TimeSpan]::FromMilliseconds(150000)
Get-S3Object -BucketName <BUCKET_NAME> -ClientConfig $s3Config
```

**Adding ClientConfig properties by using PowerShell splatting**

```powershell
$params=@{
    ClientConfig=@{
        ForcePathStyle=$true
        Timeout=[TimeSpan]::FromMilliseconds(150000)
    }
    BucketName="<BUCKET_NAME>"
}
Get-S3Object @params
```

**Using an undefined property**

When using PowerShell splatting, if you specify a ClientConfig property that doesn't exist, the AWS Tools for PowerShell doesn't detect the error until runtime, at which time it returns an exception. Modifying the example from above:

```powershell
$params=@{
    ClientConfig=@{
        ForcePathStyle=$true
        UndefinedProperty="Value"
        Timeout=[TimeSpan]::FromMilliseconds(150000)
    }
    BucketName="<BUCKET_NAME>"
}
Get-S3Object @params
```

This example produces an exception similar to the following:
Cannot bind parameter 'ClientConfig'. Cannot create object of type "Amazon.S3.AmazonS3Config". The UndefinedProperty property was not found for the Amazon.S3.AmazonS3Config object.

Specifying the AWS Region

You can use the ClientConfig parameter to set the AWS Region for the command. The Region is set through the RegionEndpoint property. The AWS Tools for PowerShell calculates the Region to use according to the following precedence:

1. The -Region parameter
2. The Region passed in the ClientConfig parameter
3. The PowerShell session state
4. The shared AWS config file
5. The environment variables
6. The Amazon EC2 instance metadata, if enabled.
Tools for PowerShell code examples

The code examples in this topic show you how to use the AWS Tools for PowerShell with AWS.

*Actions* are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

*Scenarios* are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

*Cross-service examples* are sample applications that work across multiple AWS services.

**Examples**

- Actions and scenarios using Tools for PowerShell

**Actions and scenarios using Tools for PowerShell**

The following code examples show how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with AWS services.

*Actions* are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

*Scenarios* are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

**Services**

- ACM examples using Tools for PowerShell
- Application Auto Scaling examples using Tools for PowerShell
- AppStream 2.0 examples using Tools for PowerShell
- Aurora examples using Tools for PowerShell
- Auto Scaling examples using Tools for PowerShell
- AWS Budgets examples using Tools for PowerShell
- AWS Cloud9 examples using Tools for PowerShell
- AWS CloudFormation examples using Tools for PowerShell
- CloudFront examples using Tools for PowerShell
- CloudTrail examples using Tools for PowerShell
- CloudWatch examples using Tools for PowerShell
- CodeCommit examples using Tools for PowerShell
- CodeDeploy examples using Tools for PowerShell
- CodePipeline examples using Tools for PowerShell
- Amazon Cognito Identity examples using Tools for PowerShell
- AWS Config examples using Tools for PowerShell
- Device Farm examples using Tools for PowerShell
- AWS Directory Service examples using Tools for PowerShell
- AWS DMS examples using Tools for PowerShell
- DynamoDB examples using Tools for PowerShell
- Amazon EC2 examples using Tools for PowerShell
- Amazon ECR examples using Tools for PowerShell
- Amazon ECS examples using Tools for PowerShell
- Amazon EFS examples using Tools for PowerShell
- Amazon EKS examples using Tools for PowerShell
- Elastic Load Balancing - Version 1 examples using Tools for PowerShell
- Elastic Load Balancing - Version 2 examples using Tools for PowerShell
- Amazon FSx examples using Tools for PowerShell
- AWS Glue examples using Tools for PowerShell
- AWS Health examples using Tools for PowerShell
- IAM examples using Tools for PowerShell
- Kinesis examples using Tools for PowerShell
- Lambda examples using Tools for PowerShell
- Amazon ML examples using Tools for PowerShell
- Macie examples using Tools for PowerShell
The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with ACM.

*Actions* are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

*Scenarios* are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.
Actions

Get-ACMCertificate

The following code example shows how to use Get-ACMCertificate.

Tools for PowerShell

Example 1: This example shows how to return a certificate and its chain using the ARN of the certificate.


- For API details, see GetCertificate in AWS Tools for PowerShell Cmdlet Reference.

Get-ACMCertificateDetail

The following code example shows how to use Get-ACMCertificateDetail.

Tools for PowerShell

Example 1: Returns details of the specified certificate.


Output:

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CreatedAt</td>
<td>1/21/2016 5:55:59 PM</td>
</tr>
<tr>
<td>DomainName</td>
<td><a href="http://www.example.com">www.example.com</a></td>
</tr>
<tr>
<td>DomainValidationOptions</td>
<td>{www.example.com}</td>
</tr>
<tr>
<td>InUseBy</td>
<td>{}</td>
</tr>
<tr>
<td>IssuedAt</td>
<td>1/1/0001 12:00:00 AM</td>
</tr>
<tr>
<td>Issuer</td>
<td></td>
</tr>
<tr>
<td>KeyAlgorithm</td>
<td>RSA-2048</td>
</tr>
<tr>
<td>NotAfter</td>
<td>1/1/0001 12:00:00 AM</td>
</tr>
<tr>
<td>NotBefore</td>
<td>1/1/0001 12:00:00 AM</td>
</tr>
<tr>
<td>RevocationReason</td>
<td></td>
</tr>
</tbody>
</table>
For API details, see **DescribeCertificate** in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-ACMCertificateList**

The following code example shows how to use `Get-ACMCertificateList`.

**Tools for PowerShell**

**Example 1:** Retrieves a list of all your certificate ARNs and the domain name for each. The cmdlet will automatically paginate to retrieve all the ARNs. To manually control pagination, use the `-MaxItem` parameter to control how many certificate ARNs are returned for each service call and the `-NextToken` parameter to indicate the starting point for each call.

```powershell
Get-ACMCertificateList
```

**Output:**

<table>
<thead>
<tr>
<th>CertificateArn</th>
<th>DomainName</th>
</tr>
</thead>
</table>

**Example 2:** Retrieves a list of all your certificate ARNs where the certificate status matches on the supplied states.

```powershell
Get-ACMCertificateList -CertificateStatus "VALIDATION_TIMED_OUT","FAILED"
```

**Example 3:** This example returns a list of all certificates in the us-east-1 region that have a key type of RSA_2048, and an extended key usage, or purpose, of CODE_SIGNING. You can find the values for these filtering parameters in the ListCertificates Filters API reference topic: https://docs.aws.amazon.com/acm/latest/APIReference/API_Filters.html.
Get-ACMCertificateList -Region us-east-1 -Includes_KeyType RSA_2048 -Includes_ExtendedKeyUsage CODE_SIGNING

Output:

<table>
<thead>
<tr>
<th>CertificateArn</th>
<th>DomainName</th>
</tr>
</thead>
<tbody>
<tr>
<td>arn:aws:acm:us-east-1:8xxxxxxxxxxxxx:certificate/xxxxxxxxx-d7c0-48c1-af8d-2133d8f30zzz</td>
<td>*.route53docs.com</td>
</tr>
<tr>
<td>arn:aws:acm:us-east-1:8xxxxxxxxxxxxx:certificate/xxxxxxxxx-98a5-443d-a734-800430c80zzz</td>
<td>nerdzism.net</td>
</tr>
<tr>
<td>arn:aws:acm:us-east-1:8xxxxxxxxxxxxx:certificate/xxxxxxxxx-2be6-4376-8fa7-bad559525zzz</td>
<td></td>
</tr>
<tr>
<td>arn:aws:acm:us-east-1:8xxxxxxxxxxxxx:certificate/xxxxxxxxx-e7ca-44c5-803e-24d9f2f36zzz</td>
<td></td>
</tr>
<tr>
<td>arn:aws:acm:us-east-1:8xxxxxxxxxxxxx:certificate/xxxxxxxxx-1241-4b71-80b1-090305a62zzz</td>
<td></td>
</tr>
<tr>
<td>arn:aws:acm:us-east-1:8xxxxxxxxxxxxx:certificate/xxxxxxxxx-8709-4568-8c64-f94617c99zzz</td>
<td></td>
</tr>
<tr>
<td>arn:aws:acm:us-east-1:8xxxxxxxxxxxxx:certificate/xxxxxxxxx-a8fa-4a61-98cf-e08ccc0e6zzz</td>
<td></td>
</tr>
</tbody>
</table>

- For API details, see [ListCertificates](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**New-ACMCertificate**

The following code example shows how to use `New-ACMCertificate`.

**Tools for PowerShell**

**Example 1: Creates a new certificate. The service returns the ARN of the new certificate.**

New-ACMCertificate -DomainName "www.example.com"

Output:

arn:aws:acm:us-east-1:123456789012:certificate/12345678-1234-1234-1234-123456789012
Example 2: Creates a new certificate. The service returns the ARN of the new certificate.

```
New-ACMCertificate -DomainName "www.example.com" -SubjectAlternativeName "example.com","www.example.net"
```

Output:

```
arn:aws:acm:us-east-1:123456789012:certificate/12345678-1234-1234-1234-123456789012
```

- For API details, see [RequestCertificate](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Remove-ACMCertificate

The following code example shows how to use Remove-ACMCertificate.

Tools for PowerShell

**Example 1:** Deletes the certificate identified by the supplied ARN and the associated private key. The cmdlet will prompt for confirmation before proceeding; add the -Force switch to suppress confirmation.

```
```

- For API details, see [DeleteCertificate](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Send-ACMValidationEmail

The following code example shows how to use Send-ACMValidationEmail.

Tools for PowerShell

**Example 1:** Requests that the email to validate domain ownership for 'www.example.com' be sent. If your shell's $ConfirmPreference is set to 'Medium' or lower, the cmdlet will prompt for confirmation before proceeding. Add the -Force switch to suppress confirmation prompts.

```
$params = @(
)
```
Domain="www.example.com"
    ValidationDomain="example.com"
}
Send-ACMValidationEmail @params

- For API details, see ResendValidationEmail in AWS Tools for PowerShell Cmdlet Reference.

Application Auto Scaling examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with Application Auto Scaling.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics
  - Actions

Actions

Add-AASScalableTarget

The following code example shows how to use Add-AASScalableTarget.

Tools for PowerShell

Example 1: This cmdlet registers or updates a scalable target. A scalable target is a resource that Application Auto Scaling can scale out and scale in.

Add-AASScalableTarget -ServiceNamespace AppStream -ResourceId fleet/MyFleet -ScalableDimension appstream:fleet:DesiredCapacity -MinCapacity 2 -MaxCapacity 10

- For API details, see RegisterScalableTarget in AWS Tools for PowerShell Cmdlet Reference.
Get-AASScalableTarget

The following code example shows how to use Get-AASScalableTarget.

Tools for PowerShell

Example 1: This example will provide information about the Application Autoscaling Scalable targets in the specified namespace.

```
Get-AASScalableTarget -ServiceNamespace "AppStream"
```

Output:

<table>
<thead>
<tr>
<th>CreationTime</th>
<th>11/7/2019 2:30:03 AM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxCapacity</td>
<td>5</td>
</tr>
<tr>
<td>MinCapacity</td>
<td>1</td>
</tr>
<tr>
<td>ResourceId</td>
<td>fleet/Test</td>
</tr>
<tr>
<td>RoleARN</td>
<td>arn:aws:iam::012345678912:role/aws-service-role/appstream.application-autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_AppStreamFleet</td>
</tr>
<tr>
<td>ScalableDimension</td>
<td>appstream:fleet:DesiredCapacity</td>
</tr>
<tr>
<td>ServiceNamespace</td>
<td>appstream</td>
</tr>
<tr>
<td>SuspendedState</td>
<td>Amazon.ApplicationAutoScaling.Model.SuspendedState</td>
</tr>
</tbody>
</table>

- For API details, see [DescribeScalableTargets](#) in AWS Tools for PowerShell Cmdlet Reference.

Get-AASScalingActivity

The following code example shows how to use Get-AASScalingActivity.

Tools for PowerShell

Example 1: Provides descriptive information about the scaling activities in the specified namespace from the previous six weeks.

```
Get-AASScalingActivity -ServiceNamespace AppStream
```

Output:

<table>
<thead>
<tr>
<th>ActivityId</th>
<th>2827409f-b639-4c6b-a957-8055d5d07434</th>
</tr>
</thead>
</table>

Application Auto Scaling
Cause: monitor alarm Appstream2-MyFleet-default-scale-in-Alarm in state ALARM triggered policy default-scale-in
Description: Setting desired capacity to 2.
Details:
EndTime: 12/14/2019 11:32:49 AM
ResourceId: fleet/MyFleet
ScalableDimension: appstream:fleet:DesiredCapacity
ServiceNamespace: appstream
StartTime: 12/14/2019 11:32:14 AM
StatusCode: Successful
StatusMessage: Successfully set desired capacity to 2. Change successfully fulfilled by appstream.

- For API details, see DescribeScalingActivities in AWS Tools for PowerShell Cmdlet Reference.

Get-AASScalingPolicy

The following code example shows how to use Get-AASScalingPolicy.

Tools for PowerShell

Example 1: This cmdlet describe the Application Auto Scaling scaling policies for the specified service namespace.

Get-AASScalingPolicy -ServiceNamespace AppStream

Output:

Alarms: {Appstream2-LabFleet-default-scale-out-Alarm}
PolicyName: default-scale-out
PolicyType: StepScaling
ResourceId: fleet/LabFleet
ScalableDimension: appstream:fleet:DesiredCapacity
ServiceNamespace: appstream
TargetTrackingScalingPolicyConfiguration:

Alarms : {Appstream2-LabFleet-default-scale-in-Alarm}
PolicyName : default-scale-in
PolicyType : StepScaling
ResourceId : fleet/LabFleet
ScalableDimension : appstream:fleet:DesiredCapacity
ServiceNamespace : appstream
StepScalingPolicyConfiguration : 
  Amazon.ApplicationAutoScaling.Model.StepScalingPolicyConfiguration
TargetTrackingScalingPolicyConfiguration :


Get-AASScheduledAction

The following code example shows how to use Get-AASScheduledAction.

**Tools for PowerShell**

**Example 1:** This cmdlet lists the actions scheduled for your Auto Scaling group that haven't run or that have not reached their end time.

```
Get-AASScheduledAction -ServiceNamespace AppStream
```

**Output:**

<table>
<thead>
<tr>
<th>CreationTime</th>
<th>12/22/2019 9:25:52 AM</th>
</tr>
</thead>
<tbody>
<tr>
<td>EndTime</td>
<td>1/1/0001 12:00:00 AM</td>
</tr>
<tr>
<td>ResourceId</td>
<td>fleet/MyFleet</td>
</tr>
<tr>
<td>ScalableDimension</td>
<td>appstream:fleet:DesiredCapacity</td>
</tr>
<tr>
<td>ScalableTargetAction</td>
<td>Amazon.ApplicationAutoScaling.Model.ScalableTargetAction</td>
</tr>
<tr>
<td>Schedule</td>
<td>cron(0 0 8 ? * MON-FRI *)</td>
</tr>
</tbody>
</table>
Remove-AASScalableTarget

The following code example shows how to use Remove-AASScalableTarget.

Tools for PowerShell

Example 1: This cmdlet deregisters an Application Auto Scaling scalable target. Deregistering a scalable target deletes the scaling policies that are associated with it.

```powershell
Remove-AASScalableTarget -ResourceId fleet/MyFleet -ScalableDimension appstream:fleet:DesiredCapacity -ServiceNamespace AppStream
```

Output:

```
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-AASScalableTarget (DeregisterScalableTarget)" on target "fleet/MyFleet".  
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): Y
```

For API details, see DescribeScheduledActions in AWS Tools for PowerShell Cmdlet Reference.

Remove-AASScalingPolicy

The following code example shows how to use Remove-AASScalingPolicy.

Tools for PowerShell

Example 1: This cmdlet deletes the specified scaling policy for an Application Auto Scaling scalable target.

```powershell
Remove-AASScalingPolicy -ServiceNamespace AppStream -PolicyName "default-scale-out" -ResourceId fleet/Test -ScalableDimension appstream:fleet:DesiredCapacity
```

For API details, see DeregisterScalableTarget in AWS Tools for PowerShell Cmdlet Reference.
Remove-AASScheduledAction

The following code example shows how to use Remove-AASScheduledAction.

Tools for PowerShell

Example 1: This cmdlet deletes the specified scheduled action for an Application Auto Scaling scalable target.

```powershell
Remove-AASScheduledAction -ServiceNamespace AppStream -ScheduledActionName WeekDaysFleetScaling -ResourceId fleet/MyFleet -ScalableDimension appstream:fleet:DesiredCapacity
```

Output:

```
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-AASScheduledAction (DeleteScheduledAction)" on target "WeekDaysFleetScaling".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): Y
```

Set-AASScalingPolicy

The following code example shows how to use Set-AASScalingPolicy.

Tools for PowerShell

Example 1: This cmdlet creates or updates a policy for an Application Auto Scaling scalable target. Each scalable target is identified by a service namespace, resource ID, and scalable dimension.

```powershell
Set-AASScalingPolicy -ServiceNamespace AppStream -PolicyName ASFleetScaleInPolicy -PolicyType StepScaling -ResourceId fleet/MyFleet -ScalableDimension appstream:fleet:DesiredCapacity -StepScalingPolicyConfiguration_AdjustmentType ChangeInCapacity -StepScalingPolicyConfiguration_Cooldown 360
```
Output:

<table>
<thead>
<tr>
<th>Alarms</th>
<th>PolicyARN</th>
</tr>
</thead>
</table>

- For API details, see [PutScalingPolicy](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Set-AASScheduledAction

The following code example shows how to use `Set-AASScheduledAction`.

**Tools for PowerShell**

**Example 1:** This cmdlet creates or updates a scheduled action for an Application Auto Scaling scalable target. Each scalable target is identified by a service namespace, resource ID, and scalable dimension.

```
Set-AASScheduledAction -ServiceNamespace AppStream -ResourceId fleet/MyFleet -Schedule "cron(0 0 8 ? * MON-FRI *)" -ScalableDimension appstream:Fleet:DesiredCapacity -ScheduledActionName WeekDaysFleetScaling -ScalableTargetAction_MinCapacity 5 -ScalableTargetAction_MaxCapacity 10
```

- For API details, see [PutScheduledAction](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### AppStream 2.0 examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with AppStream 2.0.

*Actions* are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.
Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

- Actions

Actions

Add-APSResourceTag

The following code example shows how to use Add-APSResourceTag.

Tools for PowerShell

Example 1: This sample adds a resource Tag to AppStream resource

```
SessionScriptTest -Tag @{$StackState='Test'} -Select ^Tag
```

Output:

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>StackState</td>
<td>Test</td>
</tr>
</tbody>
</table>
```

- For API details, see TagResource in AWS Tools for PowerShell Cmdlet Reference.

Copy-APSImage

The following code example shows how to use Copy-APSImage.

Tools for PowerShell

Example 1: This sample copies an image to other region

```
Copy-APSImage -DestinationImageName TestImageCopy -DestinationRegion us-west-2 -SourceImageName PowerShell
```
The following code example shows how to use `Disable-APSUser`.

**Tools for PowerShell**

**Example 1: This sample disables an user in USERPOOL**

```
Disable-APSUser -AuthenticationType USERPOOL -UserName TestUser@lab.com
```

• For API details, see [DisableUser](#) in *AWS Tools for PowerShell Cmdlet Reference*.

The following code example shows how to use `Enable-APSUser`.

**Tools for PowerShell**

**Example 1: This sample enables a disabled user in USERPOOL**

```
Enable-APSUser -AuthenticationType USERPOOL -UserName TestUser@lab.com
```

• For API details, see [EnableUser](#) in *AWS Tools for PowerShell Cmdlet Reference*.

The following code example shows how to use `Get-APSAssociatedFleetList`.

**Tools for PowerShell**

**Example 1: This sample displays fleet associated with a stack**

```
Get-APSAssociatedFleetList -StackName PowershellStack
```
Output:

PowershellFleet

- For API details, see ListAssociatedFleets in AWS Tools for PowerShell Cmdlet Reference.

Get-APSAssociatedStackList

The following code example shows how to use Get-APSAssociatedStackList.

Tools for PowerShell

Example 1: This sample displays stack associated with a fleet

Get-APSAssociatedStackList -FleetName PowershellFleet

Output:

PowershellStack

- For API details, see ListAssociatedStacks in AWS Tools for PowerShell Cmdlet Reference.

Get-APSDirectoryConfigList

The following code example shows how to use Get-APSDirectoryConfigList.

Tools for PowerShell

Example 1: This sample displays Directory Configurations created in AppStream

Get-APSDirectoryConfigList | Select DirectoryName, OrganizationalUnitDistinguishedNames, CreatedTime

Output:

DirectoryName OrganizationalUnitDistinguishedNames          CreatedTime
------------- ------------------------------------ -----------
Test.com      {OU=AppStream,DC=Test,DC=com}              9/6/2019 10:56:40 AM
contoso.com   {OU=AppStream,OU=contoso,DC=contoso,DC=com} 8/9/2019 9:08:50 AM

- For API details, see DescribeDirectoryConfigs in AWS Tools for PowerShell Cmdlet Reference.
Get-APSFleetList

The following code example shows how to use Get-APSFleetList.

Tools for PowerShell

Example 1: This Sample displays details of a fleet

Get-APSFleetList -Name Test

Output:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arn</td>
<td>arn:aws:appstream:us-east-1:1234567890:fleet/Test</td>
</tr>
<tr>
<td>ComputeCapacityStatus</td>
<td>Amazon.AppStream.Model.ComputeCapacityStatus</td>
</tr>
<tr>
<td>CreatedTime</td>
<td>9/12/2019 5:00:45 PM</td>
</tr>
<tr>
<td>Description</td>
<td>Test</td>
</tr>
<tr>
<td>DisconnectTimeoutInSeconds</td>
<td>900</td>
</tr>
<tr>
<td>DisplayName</td>
<td>Test</td>
</tr>
<tr>
<td>DomainJoinInfo</td>
<td></td>
</tr>
<tr>
<td>EnableDefaultInternetAccess</td>
<td>False</td>
</tr>
<tr>
<td>FleetErrors</td>
<td>{}</td>
</tr>
<tr>
<td>FleetType</td>
<td>ON_DEMAND</td>
</tr>
<tr>
<td>IamRoleArn</td>
<td></td>
</tr>
<tr>
<td>IdleDisconnectTimeoutInSeconds</td>
<td>900</td>
</tr>
<tr>
<td>ImageArn</td>
<td>arn:aws:appstream:us-east-1:1234567890:image/Test</td>
</tr>
<tr>
<td>ImageName</td>
<td>Test</td>
</tr>
<tr>
<td>InstanceType</td>
<td>stream.standard.medium</td>
</tr>
<tr>
<td>MaxUserDurationInSeconds</td>
<td>57600</td>
</tr>
<tr>
<td>Name</td>
<td>Test</td>
</tr>
<tr>
<td>State</td>
<td>STOPPED</td>
</tr>
<tr>
<td>VpcConfig</td>
<td>Amazon.AppStream.Model.VpcConfig</td>
</tr>
</tbody>
</table>

- For API details, see [DescribeFleets](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-APSImageBuilderList

The following code example shows how to use Get-APSImageBuilderList.

Tools for PowerShell

Example 1: This Sample displays details of an ImageBuilder
Get-APSImageBuilderList -Name TestImage

Output:

AccessEndpoints : {}
AppstreamAgentVersion : 06-19-2019
CreatedTime : 1/14/2019 4:33:05 AM
Description :
DisplayName : TestImage
DomainJoinInfo :
EnableDefaultInternetAccess : False
IamRoleArn :
ImageArn : arn:aws:appstream:us-east-1::image/Base-Image-Buildern
ImageBuilderErrors : {} 
InstanceType : stream.standard.large
Name : TestImage
Platform : WINDOWS
State : STOPPED
StateChangeReason :
VpcConfig : Amazon.AppStream.Model.VpcConfig

- For API details, see [DescribeImageBuilders](aws docs) in [AWS Tools for PowerShell Cmdlet Reference](aws docs).

**Get-APSImageList**

The following code example shows how to use Get-APSImageList.

**Tools for PowerShell**

**Example 1: This sample displays private AppStream Images**

Get-APSImageList -Type PRIVATE | select DisplayName, ImageBuilderName, Visibility, Arn

Output:

<table>
<thead>
<tr>
<th>DisplayName</th>
<th>ImageBuilderName</th>
<th>Visibility</th>
<th>Arn</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Get-APSImagePermission

The following code example shows how to use Get-APSImagePermission.

Tools for PowerShell

Example 1: This sample displays Image permissions on a shared AppStream Image

```powershell
Get-APSImagePermission -Name Powershell  | select SharedAccountId, @{n="AllowFleet";e={$_._ImagePermissions.AllowFleet}}, @{n="AllowImageBuilder";e={$_._ImagePermissions.AllowImageBuilder}}
```

Output:

```
SharedAccountId AllowFleet AllowImageBuilder
--------------------------
123456789012          True              True
```

Get-APSSessionList

The following code example shows how to use Get-APSSessionList.

Tools for PowerShell

Example 1: This sample displays list of sessions to a fleet

```powershell
Get-APSSessionList -FleetName PowershellFleet -StackName PowershellStack
```

Output:

```
AuthenticationType : API
```
AWS Tools for PowerShell

User Guide

ConnectionState: CONNECTED
FleetName: PowershellFleet
Id: d8987c70-4394-4324-a396-2d485c26f2a2
MaxExpirationTime: 12/27/2019 4:54:07 AM
StackName: PowershellStack
StartTime: 12/26/2019 12:54:12 PM
State: ACTIVE
UserId: Test

For API details, see DescribeSessions in AWS Tools for PowerShell Cmdlet Reference.

Get-APSStackList

The following code example shows how to use Get-APSStackList.

Tools for PowerShell

Example 1: This sample displays list of AppStream Stack

Get-APSStackList | Select DisplayName, Arn, CreatedTime

Output:

DisplayName                Arn                                      CreatedTime
-----------------           -----------------                    -----------

For API details, see DescribeStacks in AWS Tools for PowerShell Cmdlet Reference.

Get-APSTagsForResourceList

The following code example shows how to use Get-APSTagsForResourceList.

Tools for PowerShell

Example 1: This sample displays tags on an AppStream resource

Output:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>StackState</td>
<td>Test</td>
</tr>
</tbody>
</table>


Get-APSUsageReportSubscription

The following code example shows how to use Get-APSUsageReportSubscription.

Tools for PowerShell

Example 1: This sample displays AppStreamUsageReport configuration details

Get-APSUsageReportSubscription

Output:

<table>
<thead>
<tr>
<th>LastGeneratedReportDate</th>
<th>S3BucketName</th>
<th>Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/1/0001 12:00:00 AM</td>
<td>appstream-logs-us-east-1-123456789012-sik1hnxe</td>
<td>DAILY</td>
</tr>
</tbody>
</table>


Get-APSUser

The following code example shows how to use Get-APSUser.

Tools for PowerShell

Example 1: This Sample displays list of users with enabled status
Get-APSUser -AuthenticationType USERPOOL | Select-Object UserName, AuthenticationType, Enabled

Output:

<table>
<thead>
<tr>
<th>UserName</th>
<th>AuthenticationType</th>
<th>Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:foo1@contoso.com">foo1@contoso.com</a></td>
<td>USERPOOL</td>
<td>True</td>
</tr>
<tr>
<td><a href="mailto:foo2@contoso.com">foo2@contoso.com</a></td>
<td>USERPOOL</td>
<td>True</td>
</tr>
<tr>
<td><a href="mailto:foo3@contoso.com">foo3@contoso.com</a></td>
<td>USERPOOL</td>
<td>True</td>
</tr>
<tr>
<td><a href="mailto:foo4@contoso.com">foo4@contoso.com</a></td>
<td>USERPOOL</td>
<td>True</td>
</tr>
<tr>
<td><a href="mailto:foo5@contoso.com">foo5@contoso.com</a></td>
<td>USERPOOL</td>
<td>True</td>
</tr>
</tbody>
</table>

• For API details, see [DescribeUsers](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-APSUserStackAssociation

The following code example shows how to use Get-APSUserStackAssociation.

Tools for PowerShell

Example 1: This sample displays list of users assigned to a stack

Get-APSUserStackAssociation -StackName PowershellStack

Output:

<table>
<thead>
<tr>
<th>AuthenticationType</th>
<th>SendEmailNotification</th>
<th>StackName</th>
<th>UserName</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERPOOL</td>
<td>False</td>
<td>PowershellStack</td>
<td><a href="mailto:TestUser1@lab.com">TestUser1@lab.com</a></td>
</tr>
<tr>
<td>USERPOOL</td>
<td>False</td>
<td>PowershellStack</td>
<td><a href="mailto:TestUser2@lab.com">TestUser2@lab.com</a></td>
</tr>
</tbody>
</table>

• For API details, see [DescribeUserStackAssociations](#) in *AWS Tools for PowerShell Cmdlet Reference*.

New-APSDirectoryConfig

The following code example shows how to use New-APSDirectoryConfig.
Example 1: This sample creates a directory configuration in AppStream

```powershell
New-APSDirectoryConfig -ServiceAccountCredentials_AccountName contoso\ServiceAccount -ServiceAccountCredentials_AccountPassword MyPass -DirectoryName contoso.com -OrganizationalUnitDistinguishedName "OU=AppStream,OU=Contoso,DC=Contoso,DC=com"
```

Output:

```
CreatedAt DirectoryName OrganizationalUnitDistinguishedNames
------------- --------------- ------------------------------------
12/27/2019 11:00:30 AM contoso.com   {OU=AppStream,OU=Contoso,DC=Contoso,DC=com}
```

- For API details, see [CreateDirectoryConfig](#) in *AWS Tools for PowerShell Cmdlet Reference*.

New-APSFleet

The following code example shows how to use `New-APSFleet`.

Example 1: This sample creates a new AppStream fleet

```powershell
New-APSFleet -ComputeCapacity_DesiredInstance 1 -InstanceType stream.standard.medium -Name TestFleet -DisplayName TestFleet -FleetType ON_DEMAND -EnableDefaultInternetAccess $True -VpcConfig_SubnetIds "subnet-123ce32","subnet-a1234cfd" -VpcConfig_SecurityGroupIds sg-4d012a34 -ImageName SessionScriptTest -Region us-west-2
```

Output:

```
ComputeCapacityStatus         : Amazon.AppStream.Model.ComputeCapacityStatus
CreatedTime                    : 12/27/2019 11:24:42 AM
Description                    :
DisconnectTimeoutInSeconds     : 900
```

AppStream 2.0
DisplayName                    : TestFleet
DomainJoinInfo                 :
EnableDefaultInternetAccess    : True
FleetErrors                    : {}  
FleetType                      : ON_DEMAND
IamRoleArn                     :
IdleDisconnectTimeoutInSeconds : 0
SessionScriptTest             : SessionScriptTest
ImageName                      : SessionScriptTest
InstanceType                   : stream.standard.medium
MaxUserDurationInSeconds       : 57600
Name                           : TestFleet
State                          : STOPPED
VpcConfig                      : Amazon.AppStream.Model.VpcConfig

• For API details, see CreateFleet in AWS Tools for PowerShell Cmdlet Reference.

New-APSImageBuilder

The following code example shows how to use New-APSImageBuilder.

Tools for PowerShell

Example 1: This sample creates an Image Builder in AppStream

```
New-APSImageBuilder -InstanceType stream.standard.medium -Name TestIB -DisplayName TestIB -ImageName AppStream-WinServer2012R2-12-12-2019 -EnableDefaultInternetAccess $True -VpcConfig_SubnetId subnet-a1234cfd -VpcConfig_SecurityGroupIds sg-2d012a34 -Region us-west-2
```

Output:

```
AccessEndpoints             : {}
AppstreamAgentVersion       : 12-16-2019
CreatedTime                  : 12/27/2019 11:39:24 AM
Description                  : 
DisplayName                  : TestIB
DomainJoinInfo               : 
EnableDefaultInternetAccess  : True
```
AWS Tools for PowerShell

IamRoleArn                  : 
ImageArn                    : arn:aws:appstream:us-west-2::image/AppStream-WinServer2012R2-12-12-2019 
ImageBuilderErrors          : {} 
InstanceType                : stream.standard.medium 
Name                        : TestIB 
NetworkAccessConfiguration  : 
Platform                    : WINDOWS 
State                       : PENDING 
StateChangeReason           : 
VpcConfig                   : Amazon.AppStream.Model.VpcConfig 

• For API details, see [CreateImageBuilder](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**New-APSImageBuilderStreamingURL**

The following code example shows how to use `New-APSImageBuilderStreamingURL`.

**Tools for PowerShell**

**Example 1:** This sample creates an ImageBuilder streaming URL with validity of 2 hours

```
New-APSImageBuilderStreamingURL -Name TestIB -Validity 7200 -Region us-west-2
```

**Output:**

```
Expires               StreamingURL
-------               ------------
12/27/2019 1:49:13 PM https://appstream2.us-west-2.aws.amazon.com/authenticate?parameters=eyJ0eXBlIjoiQURNSU4iLCJleHBpcmVzIjoiMTU3NzQ1NDU1MyIsImF3c0FjY291bnRJZCI6IjM5MzQwMzgxMTQwNyIsInVzZXJJZCI6ImFkbWluIiwiY2F
```

• For API details, see [CreateImageBuilderStreamingURL](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**New-APSStack**

The following code example shows how to use `New-APSStack`.

**Tools for PowerShell**

**Example 1:** This sample creates a new AppStream Stack

```
New-APSStack
```

**Example 1:** This sample creates a new AppStream Stack

```
New-APSStack
```

AppStream 2.0
New-APSStack -Name TestStack -DisplayName TestStack -ApplicationSettings_Enabled $True -ApplicationSettings_SettingsGroup TestStack -Region us-west-2

Output:

AccessEndpoints : {}
ApplicationSettings : Amazon.AppStream.Model.ApplicationSettingsResponse
CreatedTime : 12/27/2019 12:34:19 PM
Description :
DisplayName : TestStack
EmbedHostDomains : {}
FeedbackURL :
Name : TestStack
RedirectURL :
StackErrors : {}
StorageConnectors : {}

• For API details, see CreateStack in AWS Tools for PowerShell Cmdlet Reference.

New-APSStreamingURL

The following code example shows how to use New-APSStreamingURL.

Tools for PowerShell

Example 1: This sample creates a streaming URL of Stack

New-APSStreamingURL -StackName SessionScriptTest -FleetName SessionScriptNew -UserID TestUser

Output:

Expires StreamingURL
-------- --------------
12/27/2019 12:43:37 PM https://appstream2.us-east-1.aws.amazon.com/authenticate?parameters=eyJ0eXBlIjoiRU5EX1VTRVIiLCJleHBpcmVzIjoiMTU3NzQ1MDYxNyIsImF3c0FjY291bnRJZCI6IjM5MzQwMzgxMTQwNyIsInVzZXJJZCI6IlRlc3RVc2
For API details, see CreateStreamingURL in AWS Tools for PowerShell Cmdlet Reference.

**New-APSUsageReportSubscription**

The following code example shows how to use New-APSUsageReportSubscription.

**Tools for PowerShell**

**Example 1: This sample enables AppStream Usage Reports**

```powershell
New-APSUsageReportSubscription
```

**Output:**

<table>
<thead>
<tr>
<th>S3BucketName</th>
<th>Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>appstream-logs-us-east-1-123456789012-sik2hnxed</td>
<td>DAILY</td>
</tr>
</tbody>
</table>

For API details, see CreateUsageReportSubscription in AWS Tools for PowerShell Cmdlet Reference.

**New-APSUser**

The following code example shows how to use New-APSUser.

**Tools for PowerShell**

**Example 1: This sample creates a user in USERPOOL**

```powershell
New-APSUser -UserName Test@lab.com -AuthenticationType USERPOOL -FirstName 'kt' -LastName 'aws' -Select ^UserName
```

**Output:**

Test@lab.com

For API details, see CreateUser in AWS Tools for PowerShell Cmdlet Reference.
Register-APSFleet

The following code example shows how to use Register-APSFleet.

Tools for PowerShell

**Example 1: This sample registers fleet with a stack**

```
Register-APSFleet -StackName TestStack -FleetName TestFleet -Region us-west-2
```

- For API details, see [AssociateFleet](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Register-APSUserStackBatch

The following code example shows how to use Register-APSUserStackBatch.

Tools for PowerShell

**Example 1: This sample assigns stack to a user in USERPOOL**

```
Register-APSUserStackBatch -UserStackAssociation
@{AuthenticationType="USERPOOL";SendEmailNotification=$False;StackName="PowershellStack";UserName="TestUser1@lab.com"}
```

- For API details, see [BatchAssociateUserStack](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Remove-APSDirectoryConfig

The following code example shows how to use Remove-APSDirectoryConfig.

Tools for PowerShell

**Example 1: This sample removes AppStream Directory configuration**

```
Remove-APSDirectoryConfig -DirectoryName contoso.com
```

**Output:**

```
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-APSDirectoryConfig (DeleteDirectoryConfig)" on target "contoso.com".
```
- For API details, see [DeleteDirectoryConfig](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-APSFleet**

The following code example shows how to use `Remove-APSFleet`.

**Tools for PowerShell**

**Example 1: This sample removes deletes an AppStream fleet**

```
Remove-APSFleet -Name TestFleet -Region us-west-2
```

**Output:**

```
Confirm
Are you sure you want to perform this action?
Performing the operation " Remove-APSFleet (DeleteFleet) " on target " TestFleet ".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y") : A
```

- For API details, see [DeleteFleet](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-APSImage**

The following code example shows how to use `Remove-APSImage`.

**Tools for PowerShell**

**Example 1: This sample deletes an Image**

```
Remove-APSImage -Name TestImage -Region us-west-2
```

**Output:**

```
Confirm
Are you sure you want to perform this action?
Performing the operation " Remove-APSImage (DeleteImage) " on target " TestImage ".
```
Applications: {}
AppstreamAgentVersion: LATEST
BaseImageArn:
CreatedTime: 12/27/2019 1:34:10 PM
Description:
DisplayName: TestImage
ImageBuilderName:
ImageBuilderSupported: True
ImagePermissions:
Name: TestImage
Platform: WINDOWS
PublicBaseImageReleasedDate: 6/12/2018 12:00:00 AM
State: AVAILABLE
StateChangeReason:
Visibility: PRIVATE

- For API details, see [DeleteImage](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-APSImageBuilder**

The following code example shows how to use `Remove-APSImageBuilder`.

### Tools for PowerShell

**Example 1: This sample deletes an ImageBuilder**

```powershell
Remove-APSImageBuilder -Name TestIB -Region us-west-2
```

**Output:**

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-APSImageBuilder (DeleteImageBuilder)" on target "TestIB".

[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): A

AccessEndpoints: {}
AppstreamAgentVersion       : 12-16-2019
CreatedTime                 : 12/27/2019 11:39:24 AM
Description                 :
DisplayName                 : TestIB
DomainJoinInfo              :
EnableDefaultInternetAccess : True
IamRoleArn                  :
ImageArn                    : arn:aws:appstream:us-west-2::image/AppStream-WinServer2012R2-12-12-2019
ImageBuilderErrors          : {}
InstanceType                : stream.standard.medium
Name                        : TestIB
Platform                    : WINDOWS
State                       : DELETING
StateChangeReason           :
VpcConfig                   : Amazon.AppStream.Model.VpcConfig

- For API details, see [DeleteImageBuilder](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-APSImagePermission**

The following code example shows how to use Remove-APSImagePermission.

**Tools for PowerShell**

**Example 1: This sample removes permissions of an Image**

```powershell
Remove-APSImagePermission -Name Powershell -SharedAccountId 123456789012
```

**Output:**

```
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-APSImagePermission (DeleteImagePermissions)" on target "Powershell".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): A
```

- For API details, see [DeleteImagePermissions](#) in *AWS Tools for PowerShell Cmdlet Reference*.
Remove-APSResourceTag

The following code example shows how to use Remove-APSResourceTag.

Tools for PowerShell

Example 1: This sample removes a resource tag from AppStream resource

```powershell
```

Output:

```
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-APSResourceTag (UntagResource)" on target "arn:aws:appstream:us-east-1:123456789012:stack/SessionScriptTest".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): A
```

- For API details, see UntagResource in AWS Tools for PowerShell Cmdlet Reference.

Remove-APSStack

The following code example shows how to use Remove-APSStack.

Tools for PowerShell

Example 1: This sample deletes a Stack

```powershell
Remove-APSStack -Name TestStack -Region us-west-2
```

Output:

```
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-APSStack (DeleteStack)" on target "TestStack".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): A
```

- For API details, see DeleteStack in AWS Tools for PowerShell Cmdlet Reference.
### Remove-APSUsageReportSubscription

The following code example shows how to use `Remove-APSUsageReportSubscription`.

#### Tools for PowerShell

**Example 1: This sample disables AppStream Usage Report subscription**

```powershell
Remove-APSUsageReportSubscription
```

**Output:**

```
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-APSUsageReportSubscription (DeleteUsageReportSubscription)" on target ".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): A
```

- For API details, see [DeleteUsageReportSubscription](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Remove-APSUser

The following code example shows how to use `Remove-APSUser`.

#### Tools for PowerShell

**Example 1: This sample deletes a user from USERPOOL**

```powershell
Remove-APSUser -UserName TestUser@lab.com -AuthenticationType USERPOOL
```

**Output:**

```
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-APSUser (DeleteUser)" on target "TestUser@lab.com".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): A
```

- For API details, see [DeleteUser](#) in *AWS Tools for PowerShell Cmdlet Reference*.  

---
Revoke-APSSession

The following code example shows how to use Revoke-APSSession.

Tools for PowerShell

Example 1: This sample revokes a session to AppStream fleet

```
Revoke-APSSession -SessionId 6cd2f9a3-f948-4aa1-8014-8a7dcde14877
```

- For API details, see [ExpireSession](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Start-APSFleet

The following code example shows how to use Start-APSFleet.

Tools for PowerShell

Example 1: This sample starts a fleet

```
Start-APSFleet -Name PowershellFleet
```

- For API details, see [StartFleet](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Start-APSImageBuilder

The following code example shows how to use Start-APSImageBuilder.

Tools for PowerShell

Example 1: This sample starts an ImageBuilder

```
Start-APSImageBuilder -Name TestImage
```

Output:

```
AccessEndpoints : {}
AppstreamAgentVersion : 06-19-2019
CreatedTime : 1/14/2019 4:33:05 AM
```
Description                 : TestImage
DisplayName                 : TestImage
DomainJoinInfo              :
EnableDefaultInternetAccess : False
IamRoleArn                  :
ImageArn                    : arn:aws:appstream:us-east-1::image/Base-Image-Builder-05-02-2018
ImageBuilderErrors          : {}
InstanceType                : stream.standard.large
Name                        : TestImage
Platform                    : WINDOWS
State                       : PENDING
StateChangeReason           :
VpcConfig                   : Amazon.AppStream.Model.VpcConfig

• For API details, see StartImageBuilder in AWS Tools for PowerShell Cmdlet Reference.

Stop-APSFleet

The following code example shows how to use Stop-APSFleet.

Tools for PowerShell

Example 1: This sample stops a fleet

Stop-APSFleet -Name PowershellFleet

• For API details, see StopFleet in AWS Tools for PowerShell Cmdlet Reference.

Stop-APSImageBuilder

The following code example shows how to use Stop-APSImageBuilder.

Tools for PowerShell

Example 1: This sample stops an ImageBuilder

Stop-APSImageBuilder -Name TestImage

Output:
For API details, see *StopImageBuilder* in *AWS Tools for PowerShell Cmdlet Reference*.

**Unregister-APSFleet**

The following code example shows how to use `Unregister-APSFleet`.

**Tools for PowerShell**

**Example 1: This sample unregisters a fleet from stack**

```powershell
Unregister-APSFleet -StackName TestStack -FleetName TestFleet -Region us-west-2
```

For API details, see *DisassociateFleet* in *AWS Tools for PowerShell Cmdlet Reference*.

**Unregister-APSUserStackBatch**

The following code example shows how to use `Unregister-APSUserStackBatch`.

**Tools for PowerShell**

**Example 1: This sample removes an user from an assigned Stack**

```powershell
Unregister-APSUserStackBatch -StackName TestStack -UserName TestUser -Region us-west-2
```
Unregister-APSUserStackBatch -UserStackAssociation
@{AuthenticationType="USERPOOL";SendEmailNotification=$False;StackName="PowershellStack";UserName="TestUser1@lab.com"}

- For API details, see [BatchDisassociateUserStack](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Update-APSDirectoryConfig

The following code example shows how to use `Update-APSDirectoryConfig`.

**Tools for PowerShell**

**Example 1:** This sample updates the Directory configuration created in AppStream

```powershell
Update-APSDirectoryConfig -ServiceAccountCredentials_AccountName contoso\ServiceAccount -ServiceAccountCredentials_AccountPassword MyPass@1$@#
-DirectoryName contoso.com -OrganizationalUnitDistinguishedName "OU=AppStreamNew,OU=Contoso,DC=Contoso,DC=com"
```

**Output:**

<table>
<thead>
<tr>
<th>CreatedTime</th>
<th>DirectoryName</th>
<th>OrganizationalUnitDistinguishedNames</th>
</tr>
</thead>
<tbody>
<tr>
<td>12/27/2019 3:50:02 PM</td>
<td>contoso.com</td>
<td>{OU=AppStreamNew,OU=Contoso,DC=Contoso,DC=com}</td>
</tr>
</tbody>
</table>

- For API details, see [UpdateDirectoryConfig](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Update-APSFleet

The following code example shows how to use `Update-APSFleet`.

**Tools for PowerShell**

**Example 1:** This sample updates properties of a fleet

```powershell
Update-APSFleet -Name PowershellFleet -EnableDefaultInternetAccess $True -DisconnectTimeoutInSecond 950
```
Output:

<table>
<thead>
<tr>
<th>Arn</th>
<th>arn:aws:appstream:us-east-1:123456789012:fleet/PowershellFleet</th>
</tr>
</thead>
<tbody>
<tr>
<td>ComputeCapacityStatus</td>
<td>Amazon.AppStream.Model.ComputeCapacityStatus</td>
</tr>
<tr>
<td>CreatedTime</td>
<td>4/24/2019 8:39:41 AM</td>
</tr>
<tr>
<td>Description</td>
<td>PowershellFleet</td>
</tr>
<tr>
<td>DisconnectTimeoutInSeconds</td>
<td>950</td>
</tr>
<tr>
<td>DisplayName</td>
<td>PowershellFleet</td>
</tr>
<tr>
<td>DomainJoinInfo</td>
<td></td>
</tr>
<tr>
<td>EnableDefaultInternetAccess</td>
<td>True</td>
</tr>
<tr>
<td>FleetErrors</td>
<td>{}</td>
</tr>
<tr>
<td>FleetType</td>
<td>ON_DEMAND</td>
</tr>
<tr>
<td>IamRoleArn</td>
<td></td>
</tr>
<tr>
<td>IdleDisconnectTimeoutInSeconds</td>
<td>900</td>
</tr>
<tr>
<td>ImageArn</td>
<td>arn:aws:appstream:us-east-1:123456789012:image/Powershell</td>
</tr>
<tr>
<td>ImageName</td>
<td>Powershell</td>
</tr>
<tr>
<td>InstanceType</td>
<td>stream.standard.medium</td>
</tr>
<tr>
<td>MaxUserDurationInSeconds</td>
<td>57600</td>
</tr>
<tr>
<td>Name</td>
<td>PowershellFleet</td>
</tr>
<tr>
<td>State</td>
<td>STOPPED</td>
</tr>
<tr>
<td>VpcConfig</td>
<td>Amazon.AppStream.Model.VpcConfig</td>
</tr>
</tbody>
</table>

• For API details, see [UpdateFleet](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Update-APSImagePermission**

The following code example shows how to use Update-APSImagePermission.

**Tools for PowerShell**

**Example 1: This sample shares an AppStream Image with other account**

```
Update-APSImagePermission -Name Powershell -SharedAccountId 123456789012 -ImagePermissions_AllowFleet $True -ImagePermissions_AllowImageBuilder $True
```

• For API details, see [UpdateImagePermissions](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Update-APSStack**

The following code example shows how to use Update-APSStack.
Tools for PowerShell

Example 1: This sample updates (enables) Application settings persistence and Home Folders on a Stack

Update-APSStack -Name PowershellStack -ApplicationSettings_Enabled $True -ApplicationSettings_SettingsGroup PowershellStack -StorageConnector @{ConnectorType="HOMEFOLDERS"}

Output:

AccessEndpoints : {}
ApplicationSettings : Amazon.AppStream.Model.ApplicationSettingsResponse
Arn : arn:aws:appstream:us-east-1:123456789012:stack/PowershellStack
CreatedTime : 4/24/2019 8:49:29 AM
Description : PowershellStack
DisplayName : PowershellStack
EmbedHostDomains : {}
FeedbackURL : 
Name : PowershellStack
RedirectURL : 
StackErrors : {}
StorageConnectors : {Amazon.AppStream.Model.StorageConnector,
Amazon.AppStream.Model.StorageConnector}
UserSettings : {Amazon.AppStream.Model.UserSetting,
Amazon.AppStream.Model.UserSetting}

- For API details, see UpdateStack in AWS Tools for PowerShell Cmdlet Reference.

Aurora examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with Aurora.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.
Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

**Topics**

- **Actions**

**Actions**

**Get-RDSOrderableDBInstanceOption**

The following code example shows how to use `Get-RDSOrderableDBInstanceOption`.

**Tools for PowerShell**

**Example 1:** This example lists the DB engine versions that support a specific DB instance class in an AWS Region.

```powershell
$params = @{
    Engine = 'aurora-postgresql'
    DBInstanceClass = 'db.r5.large'
    Region = 'us-east-1'
}
Get-RDSOrderableDBInstanceOption @params
```

**Example 2:** This example lists the DB instance classes that are supported for a specific DB engine version in an AWS Region.

```powershell
$params = @{
    Engine = 'aurora-postgresql'
    EngineVersion = '13.6'
    Region = 'us-east-1'
}
Get-RDSOrderableDBInstanceOption @params
```

- For API details, see [DescribeOrderableDBInstanceOptions](#) in *AWS Tools for PowerShell Cmdlet Reference*.  

---

* Auror
Auto Scaling examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with Auto Scaling.

*Actions* are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

*Scenarios* are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

**Topics**

- **Actions**

**Actions**

Add-ASLoadBalancer

The following code example shows how to use Add-ASLoadBalancer.

**Tools for PowerShell**

*Example 1: This example attaches the specified load balancer to the specified Auto Scaling group.*

```powershell
Add-ASLoadBalancer -LoadBalancerName my-lb -AutoScalingGroupName my-asg
```

- For API details, see [AttachLoadBalancers](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Complete-ASLifecycleAction

The following code example shows how to use Complete-ASLifecycleAction.

**Tools for PowerShell**

*Example 1: This example completes the specified lifecycle action.*
Complete-ASLifecycleAction -LifecycleHookName myLifecycleHook -AutoScalingGroupName my-asg -LifecycleActionResult CONTINUE -LifecycleActionToken bcd2f1b8-9a78-44d3-8a7a-4dd07d7cf635

- For API details, see CompleteLifecycleAction in AWS Tools for PowerShell Cmdlet Reference.

**Disable-ASMetricsCollection**

The following code example shows how to use Disable-ASMetricsCollection.

**Tools for PowerShell**

**Example 1:** This example disables monitoring of the specified metrics for the specified Auto Scaling group.

```
Disable-ASMetricsCollection -AutoScalingGroupName my-asg -Metric @("GroupMinSize", "GroupMaxSize")
```

**Example 2:** This example disables monitoring of all metrics for the specified Auto Scaling group.

```
Disable-ASMetricsCollection -AutoScalingGroupName my-asg
```

- For API details, see DisableMetricsCollection in AWS Tools for PowerShell Cmdlet Reference.

**Dismount-ASInstance**

The following code example shows how to use Dismount-ASInstance.

**Tools for PowerShell**

**Example 1:** This example detaches the specified instance from the specified Auto Scaling group and decreases the desired capacity so that Auto Scaling does not launch a replacement instance.

```
Dismount-ASInstance -InstanceId i-93633f9b -AutoScalingGroupName my-asg -ShouldDecrementDesiredCapacity $true
```

**Output:**
Example 2: This example detaches the specified instance from the specified Auto Scaling group without decreasing the desired capacity. Auto Scaling launches a replacement instance.

Dismount-ASInstance -InstanceId i-7bf746a2 -AutoScalingGroupName my-asg -ShouldDecrementDesiredCapacity $false

Output:

For API details, see [DetachInstances](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Dismount-ASLoadBalancer**

The following code example shows how to use Dismount-ASLoadBalancer.
Tools for PowerShell

Example 1: This example detaches the specified load balancer from the specified Auto Scaling group.

```powershell
Dismount-ASLoadBalancer -LoadBalancerName my-lb -AutoScalingGroupName my-asg
```

- For API details, see [DetachLoadBalancers](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Enable-ASMetricsCollection

The following code example shows how to use `Enable-ASMetricsCollection`.

Tools for PowerShell

Example 1: This example enables monitoring of the specified metrics for the specified Auto Scaling group.

```powershell
Enable-ASMetricsCollection -Metric @("GroupMinSize", "GroupMaxSize") -AutoScalingGroupName my-asg -Granularity 1Minute
```

Example 2: This example enables monitoring of all metrics for the specified Auto Scaling group.

```powershell
Enable-ASMetricsCollection -AutoScalingGroupName my-asg -Granularity 1Minute
```

- For API details, see [EnableMetricsCollection](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Enter-ASStandby

The following code example shows how to use `Enter-ASStandby`.

Tools for PowerShell

Example 1: This example puts the specified instance into standby mode and decreases the desired capacity so that Auto Scaling does not launch a replacement instance.

```powershell
Enter-ASStandby -InstanceId i-93633f9b -AutoScalingGroupName my-asg -ShouldDecrementDesiredCapacity $true
```
AWS Tools for PowerShell

User Guide

Output:
ActivityId
AutoScalingGroupName
Cause
standby in response

: e36a5a54-ced6-4df8-bd19-708e2a59a649
: my-asg
: At 2015-11-22T15:48:06Z instance i-95b8484f was moved to
to a user request,
shrinking the capacity from 2 to 1.
Description
: Moving EC2 instance to Standby: i-95b8484f
Details
: {"Availability Zone":"us-west-2b","Subnet
ID":"subnet-5264e837"}
EndTime
:
Progress
: 50
StartTime
: 11/22/2015 7:48:06 AM
StatusCode
: InProgress
StatusMessage
:

Example 2: This example puts the speciﬁed instance into standby mode without decreasing
the desired capacity. Auto Scaling launches a replacement instance.
Enter-ASStandby -InstanceId i-93633f9b -AutoScalingGroupName my-asg ShouldDecrementDesiredCapacity $false

Output:
ActivityId
: e36a5a54-ced6-4df8-bd19-708e2a59a649
AutoScalingGroupName : my-asg
Cause
: At 2015-11-22T15:48:06Z instance i-95b8484f was moved to
standby in response to a user request.
Description
: Moving EC2 instance to Standby: i-95b8484f
Details
: {"Availability Zone":"us-west-2b","Subnet
ID":"subnet-5264e837"}
EndTime
:
Progress
: 50
StartTime
: 11/22/2015 7:48:06 AM
StatusCode
: InProgress
StatusMessage
:

• For API details, see EnterStandby in AWS Tools for PowerShell Cmdlet Reference.

Auto Scaling

162


AWS Tools for PowerShell

User Guide

Exit-ASStandby
The following code example shows how to use Exit-ASStandby.
Tools for PowerShell
Example 1: This example moves the speciﬁed instance out of standby mode.
Exit-ASStandby -InstanceId i-93633f9b -AutoScalingGroupName my-asg

Output:
ActivityId
AutoScalingGroupName
Cause
standby in response

: 1833d3e8-e32f-454e-b731-0670ad4c6934
: my-asg
: At 2015-11-22T15:51:21Z instance i-95b8484f was moved out of
to a user
request, increasing the capacity from 1 to 2.
Description
: Moving EC2 instance out of Standby: i-95b8484f
Details
: {"Availability Zone":"us-west-2b","Subnet
ID":"subnet-5264e837"}
EndTime
:
Progress
: 30
StartTime
: 11/22/2015 7:51:21 AM
StatusCode
: PreInService
StatusMessage
:

• For API details, see ExitStandby in AWS Tools for PowerShell Cmdlet Reference.
Get-ASAccountLimit
The following code example shows how to use Get-ASAccountLimit.
Tools for PowerShell
Example 1: This example describes the Auto Scaling resource limits for your AWS account.
Get-ASAccountLimit

Output:
MaxNumberOfAutoScalingGroups
Auto Scaling

: 20
163


MaxNumberOfLaunchConfigurations : 100

- For API details, see [DescribeAccountLimits](#) in AWS Tools for PowerShell Cmdlet Reference.

### Get-ASAdjustmentType

The following code example shows how to use `Get-ASAdjustmentType`.

#### Tools for PowerShell

**Example 1:** This example describes the adjustment types that are supported by Auto Scaling.

```
Get-ASAdjustmentType
```

**Output:**

```
Type
----
ChangeInCapacity
ExactCapacity
PercentChangeInCapacity
```

- For API details, see [DescribeAdjustmentTypes](#) in AWS Tools for PowerShell Cmdlet Reference.

### Get-ASAutoScalingGroup

The following code example shows how to use `Get-ASAutoScalingGroup`.

#### Tools for PowerShell

**Example 1:** This example lists the names of your Auto Scaling groups.

```
Get-ASAutoScalingGroup | format-table -property AutoScalingGroupName
```

**Output:**

```
AutoScalingGroupName
---------------------
my-asg-1
my-asg-2
my-asg-3
```
Example 2: This example describes the specified Auto Scaling group.

Get-ASAutoScalingGroup -AutoScalingGroupName my-asg-1

Output:

AutoScalingGroupName    : my-asg-1
AvailabilityZones       : {us-west-2b, us-west-2a}
CreatedTime             : 3/1/2015 9:05:31 AM
DefaultCooldown         : 300
DesiredCapacity         : 2
EnabledMetrics          : {}
HealthCheckGracePeriod  : 300
HealthCheckType         : EC2
Instances               : {my-lc}
LaunchConfigurationName : my-lc
LoadBalancerNames       : {}
MaxSize                 : 0
MinSize                 : 0
PlacementGroup          :
Status                  :
SuspendedProcesses      : {}
Tags                    : {}
TerminationPolicies     : {Default}
VPCZoneIdentifier       : subnet-e4f33493,subnet-5264e837

Example 3: This example describes the specified two Auto Scaling groups.

Get-ASAutoScalingGroup -AutoScalingGroupName @("my-asg-1", "my-asg-2")

Example 4: This example describes the Auto Scaling instances for the specified Auto Scaling group.

(Get-ASAutoScalingGroup -AutoScalingGroupName my-asg-1).Instances
Example 5: This example describes all your Auto Scaling groups.

Get-ASAutoScalingGroup

Example 6: This example describes all your Auto Scaling groups, in batches of 10.

```powershell
$nextToken = $null
do {
    Get-ASAutoScalingGroup -NextToken $nextToken -MaxRecord 10
    $nextToken = $AWSHistory.LastServiceResponse.NextToken
} while ($nextToken -ne $null)
```

Example 7: This example describes LaunchTemplate for the specified Auto Scaling group. This example assumes that the "Instance purchase options" is set to "Adhere to launch template". In case this option is set to "Combine purchase options and instance types", LaunchTemplate could be accessed using "MixedInstancesPolicy.LaunchTemplate" property.

```
(Get-ASAutoScalingGroup -AutoScalingGroupName my-ag-1).LaunchTemplate
```

Output:

<table>
<thead>
<tr>
<th>LaunchTemplateId</th>
<th>LaunchTemplateName</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>lt-06095fd619cb40371</td>
<td>test-launch-template</td>
<td>$Default</td>
</tr>
</tbody>
</table>


Get-ASAutoScalingInstance

The following code example shows how to use Get-ASAutoScalingInstance.

Tools for PowerShell

Example 1: This example lists the IDs of your Auto Scaling instances.

```powershell
Get-ASAutoScalingInstance | format-table -property InstanceId
```

Output:
InstanceId
----------
i-12345678
i-87654321
i-abcd1234

Example 2: This example describes the specified Auto Scaling instance.

Get-ASAutoScalingInstance -InstanceId i-12345678

Output:

AutoScalingGroupName : my-asg
AvailabilityZone      : us-west-2b
HealthStatus         : HEALTHY
InstanceId           : i-12345678
LaunchConfigurationName : my-lc
LifecycleState       : InService

Example 3: This example describes the specified two Auto Scaling instances.

Get-ASAutoScalingInstance -InstanceId @("i-12345678", "i-87654321")

Example 4: This example describes the Auto Scaling instances for the specified Auto Scaling group.

(Get-ASAutoScalingGroup -AutoScalingGroupName my-asg).Instances | Get-ASAutoScalingInstance

Example 5: This example describes all your Auto Scaling instances.

Get-ASAutoScalingInstance

Example 6: This example describes all your Auto Scaling instances, in batches of 10.

$nextToken = $null
do {
    Get-ASAutoScalingInstance -NextToken $nextToken -MaxRecord 10
    $nextToken = $AWSHistory.LastServiceResponse.NextToken

Auto Scaling
• For API details, see [DescribeAutoScalingInstances](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

Get-ASAutoScalingNotificationType

The following code example shows how to use Get-ASAutoScalingNotificationType.

**Tools for PowerShell**

**Example 1:** This example lists the notification types that are supported by Auto Scaling.

```powershell
Get-ASAutoScalingNotificationType
```

**Output:**

```
autoscaling:EC2_INSTANCE_LAUNCH
autoscaling:EC2_INSTANCE_LAUNCH_ERROR
autoscaling:EC2_INSTANCE_TERMINATE
autoscaling:EC2_INSTANCE_TERMINATE_ERROR
autoscaling:TEST_NOTIFICATION
```

• For API details, see [DescribeAutoScalingNotificationTypes](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

Get-ASLaunchConfiguration

The following code example shows how to use Get-ASLaunchConfiguration.

**Tools for PowerShell**

**Example 1:** This example lists the names of your launch configurations.

```powershell
Get-ASLaunchConfiguration | format-table -property LaunchConfigurationName
```

**Output:**

```
LaunchConfigurationName
-----------------------
```


Example 2: This example describes the specified launch configuration.

Get-ASLaunchConfiguration -LaunchConfigurationName my-lc-1

Output:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssociatePublicIpAddress</td>
<td>True</td>
</tr>
<tr>
<td>BlockDeviceMappings</td>
<td>/dev/xvda</td>
</tr>
<tr>
<td>ClassicLinkVPCId</td>
<td></td>
</tr>
<tr>
<td>ClassicLinkVPCSecurityGroups</td>
<td>{}</td>
</tr>
<tr>
<td>CreatedTime</td>
<td>12/12/2014 3:22:08 PM</td>
</tr>
<tr>
<td>EbsOptimized</td>
<td>False</td>
</tr>
<tr>
<td>IamInstanceProfile</td>
<td></td>
</tr>
<tr>
<td>ImageId</td>
<td>ami-043a5034</td>
</tr>
<tr>
<td>InstanceMonitoring</td>
<td>Amazon.AutoScaling.Model.InstanceMonitoring</td>
</tr>
<tr>
<td>InstanceType</td>
<td>t2.micro</td>
</tr>
<tr>
<td>KernelId</td>
<td></td>
</tr>
<tr>
<td>KeyName</td>
<td></td>
</tr>
<tr>
<td>LaunchConfigurationName</td>
<td>my-lc-1</td>
</tr>
<tr>
<td>PlacementTenancy</td>
<td></td>
</tr>
<tr>
<td>RamdiskId</td>
<td></td>
</tr>
<tr>
<td>SecurityGroups</td>
<td>{sg-67ef0308}</td>
</tr>
<tr>
<td>SpotPrice</td>
<td></td>
</tr>
<tr>
<td>UserData</td>
<td></td>
</tr>
</tbody>
</table>

Example 3: This example describes the specified two launch configurations.

Get-ASLaunchConfiguration -LaunchConfigurationName @("my-lc-1", "my-lc-2")

Example 4: This example describes all your launch configurations.

Get-ASLaunchConfiguration
Example 5: This example describes all your launch configurations, in batches of 10.

```powershell
$nextToken = $null
do {
    Get-ASLaunchConfiguration -NextToken $nextToken -MaxRecord 10
    $nextToken = $AWSHistory.LastServiceResponse.NextToken
} while ($nextToken -ne $null)
```

- For API details, see [DescribeLaunchConfigurations](#) in AWS Tools for PowerShell Cmdlet Reference.

Get-ASLifecycleHook

The following code example shows how to use Get-ASLifecycleHook.

Tools for PowerShell

Example 1: This example describes the specified lifecycle hook.

```powershell
Get-ASLifecycleHook -AutoScalingGroupName my-asg -LifecycleHookName myLifecycleHook
```

Output:

```
AutoScalingGroupName : my-asg
DefaultResult        : ABANDON
GlobalTimeout        : 172800
HeartbeatTimeout     : 3600
LifecycleHookName    : myLifecycleHook
LifecycleTransition  : auto-scaling:EC2_INSTANCE_LAUNCHING
NotificationMetadata :
RoleARN              : arn:aws:iam::123456789012:role/my-iam-role
```

Example 2: This example describes all lifecycle hooks for the specified Auto Scaling group.

```powershell
Get-ASLifecycleHook -AutoScalingGroupName my-asg
```

Example 3: This example describes all lifecycle hooks for all your Auto Scaling groups.

```powershell
Get-ASLifecycleHook
```
Get-ASLifecycleHookType

The following code example shows how to use Get-ASLifecycleHookType.

Tools for PowerShell

Example 1: This example lists the lifecycle hook types supported by Auto Scaling.

Get-ASLifecycleHookType

Output:

autoscaling:EC2_INSTANCE_LAUNCHING  
auto-scaling:EC2_INSTANCE_TERMINATING

• For API details, see DescribeLifecycleHookTypes in AWS Tools for PowerShell Cmdlet Reference.

Get-ASLoadBalancer

The following code example shows how to use Get-ASLoadBalancer.

Tools for PowerShell

Example 1: This example describes the load balancers for the specified Auto Scaling group.

Get-ASLoadBalancer -AutoScalingGroupName my-asg

Output:

<table>
<thead>
<tr>
<th>LoadBalancerName</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>my-lb</td>
<td>Added</td>
</tr>
</tbody>
</table>

• For API details, see DescribeLoadBalancers in AWS Tools for PowerShell Cmdlet Reference.

Get-ASMetricCollectionType

The following code example shows how to use Get-ASMetricCollectionType.
Tools for PowerShell

Example 1: This example lists the metric collection types that are supported by Auto Scaling.

```powershell
(Get-ASMetricCollectionType).Metrics
```

Output:

<table>
<thead>
<tr>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupMinSize</td>
</tr>
<tr>
<td>GroupMaxSize</td>
</tr>
<tr>
<td>GroupDesiredCapacity</td>
</tr>
<tr>
<td>GroupInServiceInstances</td>
</tr>
<tr>
<td>GroupPendingInstances</td>
</tr>
<tr>
<td>GroupTerminatingInstances</td>
</tr>
<tr>
<td>GroupStandbyInstances</td>
</tr>
<tr>
<td>GroupTotalInstances</td>
</tr>
</tbody>
</table>

Example 2: This example lists the corresponding granularities.

```powershell
(Get-ASMetricCollectionType).Granularities
```

Output:

<table>
<thead>
<tr>
<th>Granularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Minute</td>
</tr>
</tbody>
</table>

- For API details, see [DescribeMetricCollectionTypes](#) in AWS Tools for PowerShell Cmdlet Reference.

Get-ASNotificationConfiguration

The following code example shows how to use Get-ASNotificationConfiguration.

Tools for PowerShell

Example 1: This example describes the notification actions associated with the specified Auto Scaling group.
Get-ASNotificationConfiguration -AutoScalingGroupName my-asg | format-list

Output:

AutoScalingGroupName : my-asg
NotificationType     : auto-scaling:EC2_INSTANCE_LAUNCH

AutoScalingGroupName : my-asg
NotificationType     : auto-scaling:EC2_INSTANCE_TERMINATE

Example 2: This example describes the notification actions associated with all your Auto Scaling groups.

Get-ASNotificationConfiguration

- For API details, see DescribeNotificationConfigurations in AWS Tools for PowerShell Cmdlet Reference.

Get-ASPolicy

The following code example shows how to use Get-ASPolicy.

Tools for PowerShell

Example 1: This example describes all policies for the specified Auto Scaling group.

Get-ASPolicy -AutoScalingGroupName my-asg

Output:

AdjustmentType : ChangeInCapacity
Alarms : {}
AutoScalingGroupName : my-asg
Cooldown : 0
EstimatedInstanceWarmup : 0
MetricAggregationType : 
MinAdjustmentMagnitude : 0
例 2：此示例描述指定策略的指定 Auto Scaling 组。

```powershell
Get-ASPolicy -AutoScalingGroupName my-asg -PolicyName @("myScaleOutPolicy", "myScaleInPolicy")
```

例 3：此示例描述所有 Auto Scaling 组的所有策略。

```powershell
Get-ASPolicy
```

- 有关 API 详细信息，请参阅 `DescribePolicies` 中的 AWS Tools for PowerShell Cmdlet Reference。

**Get-ASScalingActivity**

该示例代码示例说明了如何使用 Get-ASScalingActivity。

**Tools for PowerShell**

**例 1：此示例描述了最近六周内指定 Auto Scaling 组的缩放活动。**

```powershell
Get-ASScalingActivity -AutoScalingGroupName my-asg
```

输出：

```plaintext
ActivityId : 063308ae-aa22-4a9b-94f4-9fae4EXAMPLE
AutoScalingGroupName : my-asg
Cause : 2015-11-22T15:45:16Z a user request explicitly set group
desired capacity changing the desired
capacity from 1 to 2. At 2015-11-22T15:45:34Z an instance
was started in response to a difference
```
Example 2: This example describes the specified scaling activity.

```
Get-ASScalingActivity -ActivityId "063308ae-aa22-4a9b-94f4-9fae4EXAMPLE"
```

Example 3: This example describes the scaling activities for the last six weeks for all your Auto Scaling groups.

```
Get-ASScalingActivity
```

- For API details, see [DescribeScalingActivities](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-ASScalingProcessType**

The following code example shows how to use Get-ASScalingProcessType.
Tools for PowerShell

Example 1: This example lists the process types that are supported by Auto Scaling.

```
Get-ASScalingProcessType
```

Output:

```
ProcessName
----------
AZRebalance
AddToLoadBalancer
AlarmNotification
HealthCheck
Launch
ReplaceUnhealthy
ScheduledActions
Terminate
```

- For API details, see [DescribeScalingProcessTypes](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-ASScheduledAction

The following code example shows how to use Get-ASScheduledAction.

Tools for PowerShell

Example 1: This example describes the scheduled scaling actions for the specified Auto Scaling group.

```
Get-ASScheduledAction -AutoScalingGroupName my-asg
```

Output:

```
AutoScalingGroupName : my-asg
DesiredCapacity      : 10
EndTime              :
MaxSize              :
MinSize              :
Recurrence           :
```
ScheduledActionName  : myScheduledAction
StartTime            : 11/30/2015 8:00:00 AM
Time                 : 11/30/2015 8:00:00 AM

Example 2: This example describes the specified scheduled scaling actions.

    Get-ASScheduledAction -ScheduledActionName @("myScheduledScaleOut", "myScheduledScaleIn")

Example 3: This example describes the scheduled scaling actions that start by the specified time.

    Get-ASScheduledAction -StartTime "2015-12-01T08:00:00Z"

Example 4: This example describes the scheduled scaling actions that end by the specified time.

    Get-ASScheduledAction -EndTime "2015-12-30T08:00:00Z"

Example 5: This example describes the scheduled scaling actions for all your Auto Scaling groups.

    Get-ASScheduledAction

- For API details, see DescribeScheduledActions in AWS Tools for PowerShell Cmdlet Reference.

Get-ASTag

The following code example shows how to use Get-ASTag.

Tools for PowerShell

    Example 1: This example describes the tags with a key value of either 'myTag' or 'myTag2'. The possible values for the filter name are 'auto-scaling-group', 'key', 'value', and 'propagate-at-launch'. The syntax used by this example requires PowerShell version 3 or later.
Get-ASTag -Filter @(
   @{ Name="key"; Values=@("myTag", "myTag2") }
)

Output:

<table>
<thead>
<tr>
<th>Key</th>
<th>myTag2</th>
</tr>
</thead>
<tbody>
<tr>
<td>PropagateAtLaunch</td>
<td>True</td>
</tr>
<tr>
<td>ResourceId</td>
<td>my-asg</td>
</tr>
<tr>
<td>ResourceType</td>
<td>auto-scaling-group</td>
</tr>
<tr>
<td>Value</td>
<td>myTagValue2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>myTag</th>
</tr>
</thead>
<tbody>
<tr>
<td>PropagateAtLaunch</td>
<td>True</td>
</tr>
<tr>
<td>ResourceId</td>
<td>my-asg</td>
</tr>
<tr>
<td>ResourceType</td>
<td>auto-scaling-group</td>
</tr>
<tr>
<td>Value</td>
<td>myTagValue</td>
</tr>
</tbody>
</table>

Example 2: With PowerShell version 2, you must use New-Object to create the filter for the Filter parameter.

```powershell
$keys = New-Object string[] 2
$keys[0] = "myTag"
$keys[1] = "myTag2"
$filter = New-Object Amazon.AutoScaling.Model.Filter
$filter.Name = "key"
$filter.Values = $keys
Get-ASTag -Filter @( $filter )
```

Example 3: This example describes all tags for all your Auto Scaling groups.

```powershell
Get-ASTag
```

- For API details, see [DescribeTags](#) in AWS Tools for PowerShell Cmdlet Reference.

Get-ASTerminationPolicyType

The following code example shows how to use Get-ASTerminationPolicyType.

Tools for PowerShell

Example 1: This example lists the termination policies that are supported by Auto Scaling.
Get-ASTerminationPolicyType

Output:

ClosestToNextInstanceHour
Default
NewestInstance
OldestInstance
OldestLaunchConfiguration

- For API details, see [DescribeTerminationPolicyTypes](#) in *AWS Tools for PowerShell Cmdlet Reference*.

## Mount-ASInstance

The following code example shows how to use Mount-ASInstance.

**Tools for PowerShell**

**Example 1:** This example attaches the specified instance to the specified Auto Scaling group. Auto Scaling automatically increases the desired capacity of the Auto Scaling group.

```powershell
Mount-ASInstance -InstanceId i-93633f9b -AutoScalingGroupName my-asg
```

- For API details, see [AttachInstances](#) in *AWS Tools for PowerShell Cmdlet Reference*.

## New-ASAutoScalingGroup

The following code example shows how to use New-ASAutoScalingGroup.

**Tools for PowerShell**

**Example 1:** This example creates an Auto Scaling group with the specified name and attributes. The default desired capacity is the minimum size. Therefore, this Auto Scaling group launches two instances, one in each of the specified two Availability Zones.

```powershell
New-ASAutoScalingGroup -AutoScalingGroupName my-asg -LaunchConfigurationName my-lc -MinSize 2 -MaxSize 6 -AvailabilityZone @("us-west-2a", "us-west-2b")
```
• For API details, see CreateAutoScalingGroup in AWS Tools for PowerShell Cmdlet Reference.

New-ASLaunchConfiguration

The following code example shows how to use New-ASLaunchConfiguration.

Tools for PowerShell

Example 1: This example creates a launch configuration named 'my-lc'. The EC2 instances launched by Auto Scaling groups that use this launch configuration use specified instance type, AMI, security group, and IAM role.

New-ASLaunchConfiguration -LaunchConfigurationName my-lc -InstanceType "m3.medium" -ImageId "ami-12345678" -SecurityGroup "sg-12345678" -IamInstanceProfile "myIamRole"

• For API details, see CreateLaunchConfiguration in AWS Tools for PowerShell Cmdlet Reference.

Remove-ASAutoScalingGroup

The following code example shows how to use Remove-ASAutoScalingGroup.

Tools for PowerShell

Example 1: This example deletes the specified Auto Scaling group if it has no running instances. You are prompted for confirmation before the operation proceeds.

Remove-ASAutoScalingGroup -AutoScalingGroupName my-asg

Output:

Confirm
Are you sure you want to perform this action?
Performing operation "Remove-ASAutoScalingGroup (DeleteAutoScalingGroup)" on Target "my-asg".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):

Example 2: If you specify the Force parameter, you are not prompted for confirmation before the operation proceeds.
Remove-ASAutoScalingGroup -AutoScalingGroupName my-asg -Force

**Example 3:** This example deletes the specified Auto Scaling group and terminates any running instances that it contains.

Remove-ASAutoScalingGroup -AutoScalingGroupName my-asg -ForceDelete $true -Force

- For API details, see [DeleteAutoScalingGroup](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Remove-ASLaunchConfiguration

The following code example shows how to use Remove-ASLaunchConfiguration.

**Tools for PowerShell**

**Example 1:** This example deletes the specified launch configuration if it is not attached to an Auto Scaling group. You are prompted for confirmation before the operation proceeds.

Remove-ASLaunchConfiguration -LaunchConfigurationName my-lc

**Output:**

Confirm
Are you sure you want to perform this action?
Performing operation "Remove-ASLaunchConfiguration (DeleteLaunchConfiguration)" on Target "my-lc".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):

**Example 2:** If you specify the Force parameter, you are not prompted for confirmation before the operation proceeds.

Remove-ASLaunchConfiguration -LaunchConfigurationName my-lc -Force

- For API details, see [DeleteLaunchConfiguration](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Remove-ASLifecycleHook

The following code example shows how to use Remove-ASLifecycleHook.
Tools for PowerShell

Example 1: This example deletes the specified lifecycle hook for the specified Auto Scaling group. You are prompted for confirmation before the operation proceeds.

```
Remove-ASLifecycleHook -AutoScalingGroupName my-asg -LifecycleHookName myLifecycleHook
```

Output:

```
Confirm
Are you sure you want to perform this action?
Performing operation "Remove-ASLifecycleHook (DeleteLifecycleHook)" on Target "myLifecycleHook".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):
```

Example 2: If you specify the Force parameter, you are not prompted for confirmation before the operation proceeds.

```
Remove-ASLifecycleHook -AutoScalingGroupName my-asg -LifecycleHookName myLifecycleHook -Force
```

- For API details, see [DeleteLifecycleHook](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Remove-ASNotificationConfiguration

The following code example shows how to use Remove-ASNotificationConfiguration.

Tools for PowerShell

Example 1: This example deletes the specified notification action. You are prompted for confirmation before the operation proceeds.

```
```

Output:

```
Confirm
Are you sure you want to perform this action?
```
Performing operation "Remove-ASNotificationConfiguration (DeleteNotificationConfiguration)" on Target "arn:aws:sns:us-west-2:123456789012:my-topic".

[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):

Example 2: If you specify the Force parameter, you are not prompted for confirmation before the operation proceeds.


• For API details, see DeleteNotificationConfiguration in AWS Tools for PowerShell Cmdlet Reference.

Remove-ASPolicy

The following code example shows how to use Remove-ASPolicy.

Tools for PowerShell

Example 1: This example deletes the specified policy for the specified Auto Scaling group. You are prompted for confirmation before the operation proceeds.

Remove-ASPolicy -AutoScalingGroupName my-asg -PolicyName myScaleInPolicy

Output:

Confirm
Are you sure you want to perform this action?
Performing operation "Remove-ASPolicy (DeletePolicy)" on Target "myScaleInPolicy".

[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):

Example 2: If you specify the Force parameter, you are not prompted for confirmation before the operation proceeds.

Remove-ASPolicy -AutoScalingGroupName my-asg -PolicyName myScaleInPolicy -Force

• For API details, see DeletePolicy in AWS Tools for PowerShell Cmdlet Reference.
Remove-ASScheduledAction

The following code example shows how to use Remove-ASScheduledAction.

**Tools for PowerShell**

**Example 1:** This example deletes the specified scheduled action for the specified Auto Scaling group. You are prompted for confirmation before the operation proceeds.

```powershell
Remove-ASScheduledAction -AutoScalingGroupName my-asg -ScheduledAction "myScheduledAction"
```

**Output:**

```
Confirm
Are you sure you want to perform this action?
Performing operation "Remove-ASScheduledAction (DeleteScheduledAction)" on Target "myScheduledAction".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):
```

**Example 2:** If you specify the Force parameter, you are not prompted for confirmation before the operation proceeds.

```powershell
Remove-ASScheduledAction -AutoScalingGroupName my-asg -ScheduledAction "myScheduledAction" -Force
```

- For API details, see [DeleteScheduledAction](https://aws.amazon.com) in *AWS Tools for PowerShell Cmdlet Reference*.

Remove-ASTag

The following code example shows how to use Remove-ASTag.

**Tools for PowerShell**

**Example 1:** This example removes the specified tag from the specified Auto Scaling group. You are prompted for confirmation before the operation proceeds. The syntax used by this example requires PowerShell version 3 or later.

```powershell
Remove-ASTag -Tag @( @{ResourceType="auto-scaling-group"; ResourceId="my-asg"; Key="myTag" } )
```
Output:

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-ASTag (DeleteTags)" on target
"Amazon.AutoScaling.Model.Tag".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):

Example 2: If you specify the Force parameter, you are not prompted for confirmation
before the operation proceeds.

Remove-ASTag -Tag @( @{ResourceType="auto-scaling-group"; ResourceId="my-asg";
Key="myTag" } ) -Force

Example 3: With Powershell version 2, you must use New-Object to create the tag for the
Tag parameter.

$tag = New-Object Amazon.AutoScaling.Model.Tag
$tag.ResourceType = "auto-scaling-group"
$tag.ResourceId = "my-asg"
$tag.Key = "myTag"
Remove-ASTag -Tag $tag -Force

- For API details, see [DeleteTags](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

Resume-ASProcess

The following code example shows how to use Resume-ASProcess.

Tools for PowerShell

**Example 1:** This example resumes the specified Auto Scaling process for the specified Auto
Scaling group.

Resume-ASProcess -AutoScalingGroupName my-asg -ScalingProcess "AlarmNotification"

**Example 2:** This example resumes all suspended Auto Scaling processes for the specified
Auto Scaling group.
Resume-ASProcess -AutoScalingGroupName my-asg

- For API details, see ResumeProcesses in AWS Tools for PowerShell Cmdlet Reference.

Set-ASDesiredCapacity

The following code example shows how to use Set-ASDesiredCapacity.

Tools for PowerShell

Example 1: This example sets the size of the specified Auto Scaling group.

Set-ASDesiredCapacity -AutoScalingGroupName my-asg -DesiredCapacity 2

Example 2: This example sets the size of the specified Auto Scaling group and waits for the cooldown period to complete before scaling to the new size.

Set-ASDesiredCapacity -AutoScalingGroupName my-asg -DesiredCapacity 2 -HonorCooldown $true

- For API details, see SetDesiredCapacity in AWS Tools for PowerShell Cmdlet Reference.

Set-ASInstanceHealth

The following code example shows how to use Set-ASInstanceHealth.

Tools for PowerShell

Example 1: This example sets the status of the specified instance to 'Unhealthy', taking it out of service. Auto Scaling terminates and replaces the instance.

Set-ASInstanceHealth -HealthStatus Unhealthy -InstanceId i-93633f9b

Example 2: This example sets the status of the specified instance to 'Healthy', keeping it in service. Any health check grace period for the Auto Scaling group is not honored.

Set-ASInstanceHealth -HealthStatus Healthy -InstanceId i-93633f9b -ShouldRespectGracePeriod $false

- For API details, see SetInstanceHealth in AWS Tools for PowerShell Cmdlet Reference.
**Set-ASInstanceProtection**

The following code example shows how to use `Set-ASInstanceProtection`.

**Tools for PowerShell**

**Example 1:** This example enables instance protection for the specified instance.

```powershell
Set-ASInstanceProtection -AutoScalingGroupName my-asg -InstanceId i-12345678 - ProtectedFromScaleIn $true
```

**Example 2:** This example disables instance protection for the specified instance.

```powershell
Set-ASInstanceProtection -AutoScalingGroupName my-asg -InstanceId i-12345678 - ProtectedFromScaleIn $false
```

- For API details, see [SetInstanceProtection](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Set-ASTag**

The following code example shows how to use `Set-ASTag`.

**Tools for PowerShell**

**Example 1:** This example adds a single tag to the specified Auto Scaling group. The tag key is 'myTag' and the tag value is 'myTagValue'. Auto Scaling propagates this tag to the subsequent EC2 instances launched by the Auto Scaling group. The syntax used by this example requires PowerShell version 3 or later.

```powershell
Set-ASTag -Tag @( @{ResourceType="auto-scaling-group"; ResourceId="my-asg"; Key="myTag"; Value="myTagValue"; PropagateAtLaunch=$true} )
```

**Example 2:** With PowerShell version 2, you must use `New-Object` to create the tag for the Tag parameter.

```powershell
$tag = New-Object Amazon.AutoScaling.Model.Tag
$tag.ResourceType = "auto-scaling-group"
$tag.ResourceId = "my-asg"
$tag.Key = "myTag"
$tag.Value = "myTagValue"
```
```powershell
$tag.PropagateAtLaunch = $true
Set-ASTag -Tag $tag
```

- For API details, see [CreateOrUpdateTags](https://docs.aws.amazon.com/tools/latest/powershell/doc/aws-tools-for-powershell-cmdlet-reference.html#createorupdatetags) in *AWS Tools for PowerShell Cmdlet Reference*.

### Start-ASPolicy

The following code example shows how to use `Start-ASPolicy`.

#### Tools for PowerShell

**Example 1:** This example executes the specified policy for the specified Auto Scaling group.

```powershell
Start-ASPolicy -AutoScalingGroupName my-asg -PolicyName "myScaleInPolicy"
```

**Example 2:** This example executes the specified policy for the specified Auto Scaling group, after waiting for the cooldown period to complete.

```powershell
Start-ASPolicy -AutoScalingGroupName my-asg -PolicyName "myScaleInPolicy" -HonorCooldown $true
```


### Stop-ASInstanceInAutoScalingGroup

The following code example shows how to use `Stop-ASInstanceInAutoScalingGroup`.

#### Tools for PowerShell

**Example 1:** This example terminates the specified instance and decreases the desired capacity of its Auto Scaling group so that Auto Scaling does not launch a replacement instance.

```powershell
Stop-ASInstanceInAutoScalingGroup -InstanceId i-93633f9b -ShouldDecrementDesiredCapacity $true
```

**Output:**

```
ActivityId : 2e40d9bd-1902-444c-abf3-6ea0002efdc5
```
AutoScalingGroupName : 
Cause : At 2015-11-22T16:09:03Z instance i-93633f9b was taken out of service in response to a user request, shrinking the capacity from 2 to 1.
Description : Terminating EC2 instance: i-93633f9b
Details : \{"Availability Zone":"us-west-2b","Subnet ID":"subnet-5264e837"\}
EndTime : 
Progress : 0
StartTime : 11/22/2015 8:09:03 AM
StatusCode : InProgress
StatusMessage : 

Example 2: This example terminates the specified instance without decreasing the desired capacity of its Auto Scaling group. Auto Scaling launches a replacement instance.

Stop-ASInstanceInAutoScalingGroup -InstanceId i-93633f9b -ShouldDecrementDesiredCapacity $false

Output:

ActivityId : 2e40d9bd-1902-444c-abf3-6ea0002efdc5
AutoScalingGroupName :
Cause : At 2015-11-22T16:09:03Z instance i-93633f9b was taken out of service in response to a user request.
Description : Terminating EC2 instance: i-93633f9b
Details : \{"Availability Zone":"us-west-2b","Subnet ID":"subnet-5264e837"\}
EndTime : 
Progress : 0
StartTime : 11/22/2015 8:09:03 AM
StatusCode : InProgress
StatusMessage : 

• For API details, see TerminateInstanceInAutoScalingGroup in AWS Tools for PowerShell Cmdlet Reference.

Suspend-ASProcess

The following code example shows how to use Suspend-ASProcess.
Tools for PowerShell

Example 1: This example suspends the specified Auto Scaling process for the specified Auto Scaling group.

```
Suspend-ASProcess -AutoScalingGroupName my-asg -ScalingProcess "AlarmNotification"
```

Example 2: This example suspends all Auto Scaling processes for the specified Auto Scaling group.

```
Suspend-ASProcess -AutoScalingGroupName my-asg
```

- For API details, see [SuspendProcesses](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Update-ASAutoScalingGroup

The following code example shows how to use Update-ASAutoScalingGroup.

Tools for PowerShell

Example 1: This example updates the minimum and maximum size of the specified Auto Scaling group.

```
Update-ASAutoScalingGroup -AutoScalingGroupName my-asg -MaxSize 5 -MinSize 1
```

Example 2: This example updates the default cooldown period of the specified Auto Scaling group.

```
Update-ASAutoScalingGroup -AutoScalingGroupName my-asg -DefaultCooldown 10
```

Example 3: This example updates the Availability Zones of the specified Auto Scaling group.

```
Update-ASAutoScalingGroup -AutoScalingGroupName my-asg -AvailabilityZone @"us-west-2a", "us-west-2b"
```

Example 4: This example updates the specified Auto Scaling group to use Elastic Load Balancing health checks.
Update-ASAutoScalingGroup -AutoScalingGroupName my-asg -HealthCheckType ELB -HealthCheckGracePeriod 60

- For API details, see UpdateAutoScalingGroup in AWS Tools for PowerShell Cmdlet Reference.

**Write-ASLifecycleActionHeartbeat**

The following code example shows how to use Write-ASLifecycleActionHeartbeat.

**Tools for PowerShell**

**Example 1:** This example records a heartbeat for the specified lifecycle action. This keeps the instance in a pending state until you complete the custom action.

Write-ASLifecycleActionHeartbeat -AutoScalingGroupName my-asg -LifecycleHookName myLifecycleHook -LifecycleActionToken bcd2f1b8-9a78-44d3-8a7a-4dd07d7cf635

- For API details, see RecordLifecycleActionHeartbeat in AWS Tools for PowerShell Cmdlet Reference.

**Write-ASLifecycleHook**

The following code example shows how to use Write-ASLifecycleHook.

**Tools for PowerShell**

**Example 1:** This example adds the specified lifecycle hook to the specified Auto Scaling group.


- For API details, see PutLifecycleHook in AWS Tools for PowerShell Cmdlet Reference.

**Write-ASNotificationConfiguration**

The following code example shows how to use Write-ASNotificationConfiguration.
Tools for PowerShell

Example 1: This example configures the specified Auto Scaling group to send a notification to the specified SNS topic when it launches EC2 instances.

```
```

Example 2: This example configures the specified Auto Scaling group to send a notification to the specified SNS topic when it launches or terminates EC2 instances.

```
```

- For API details, see [PutNotificationConfiguration](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

Write-ASScalingPolicy

The following code example shows how to use Write-ASScalingPolicy.

Tools for PowerShell

Example 1: This example adds the specified policy to the specified Auto Scaling group. The specified adjustment type determines how to interpret the ScalingAdjustment parameter. With 'ChangeInCapacity', a positive value increases the capacity by the specified number of instances and a negative value decreases the capacity by the specified number of instances.

```
Write-ASScalingPolicy -AutoScalingGroupName my-asg -AdjustmentType "ChangeInCapacity" -PolicyName "myScaleInPolicy" -ScalingAdjustment -1
```

Output:

```
```

- For API details, see [PutScalingPolicy](#) in [AWS Tools for PowerShell Cmdlet Reference](#).
Write-ASScheduledUpdateGroupAction

The following code example shows how to use Write-ASScheduledUpdateGroupAction.

Tools for PowerShell

Example 1: This example creates or updates a one-time scheduled action to change the desired capacity at the specified start time.

```powershell
Write-ASScheduledUpdateGroupAction -AutoScalingGroupName my-asg -ScheduledActionName "myScheduledAction" -StartTime "2015-12-01T00:00:00Z" -DesiredCapacity 10
```

- For API details, see [PutScheduledUpdateGroupAction](#) in *AWS Tools for PowerShell Cmdlet Reference*.

AWS Budgets examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with AWS Budgets.

*Actions* are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

*Scenarios* are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

**Topics**

- [Actions](#)

**Actions**

*New-BGTBudget*

The following code example shows how to use New-BGTBudget.
Tools for PowerShell

**Example 1: Creates a new budget with the specified budgetary and time constraints with email notifications.**

```powershell
$notification = @{
    NotificationType = "ACTUAL"
    ComparisonOperator = "GREATER_THAN"
    Threshold = 80
}

$addressObject = @{
    Address = @("user@domain.com")
    SubscriptionType = "EMAIL"
}

$subscriber = New-Object Amazon.Budgets.Model.NotificationWithSubscribers
$subscriber.Notification = $notification
$subscriber.Subscribers.Add($addressObject)

$startDate = [datetime]::new(2017,09,25)
$endDate = [datetime]::new(2017,10,25)

New-BGTBudget -Budget_BudgetName "Tester" -Budget_BudgetType COST -CostTypes_IncludeTax $true -Budget_TimeUnit MONTHLY -BudgetLimit_Unit USD -TimePeriod_Start $startDate -TimePeriod_End $endDate -AccountId 123456789012 -BudgetLimit_Amount 200 -NotificationsWithSubscriber $subscriber

- For API details, see [CreateBudget](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**AWS Cloud9 examples using Tools for PowerShell**

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with AWS Cloud9.

*Actions* are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

*Scenarios* are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.
Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

- **Actions**

Actions

Get-C9EnvironmentData

The following code example shows how to use Get-C9EnvironmentData.

Tools for PowerShell

**Example 1:** This example gets information about the specified AWS Cloud9 development environments.

```powershell
Get-C9EnvironmentData -EnvironmentId
685f892f431b45c2b28cb69eadcdb0EX,1980b80e5f584920801c09086667f0EX
```

Output:

```
Description : Created from CodeStar.
Id          : 685f892f431b45c2b28cb69eadcdb0EX
Name        : my-demo-ec2-env
OwnerArn    : arn:aws:iam::123456789012:user/MyDemoUser
Type        : ec2

Arn         : arn:aws:cloud9:us-east-1:123456789012:environment:1980b80e5f584920801c09086667f0EX
Description : 
Id          : 1980b80e5f584920801c09086667f0EX
Name        : my-demo-ssh-env
OwnerArn    : arn:aws:iam::123456789012:user/MyDemoUser
Type        : ssh
```
Example 2: This example gets information about the lifecycle status of the specified AWS Cloud9 development environment.

```
(Get-C9EnvironmentData -EnvironmentId 685f892f431b45c2b28cb69eadc0Ex).Lifecycle
```

Output:

<table>
<thead>
<tr>
<th>FailureResource</th>
<th>Reason</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATED</td>
<td>-------</td>
<td>-------</td>
</tr>
</tbody>
</table>

- For API details, see [DescribeEnvironments](https://aws.amazon.com) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-C9EnvironmentList**

The following code example shows how to use Get-C9EnvironmentList.

**Tools for PowerShell**

**Example 1:** This example gets a list of available AWS Cloud9 development environment identifiers.

```
Get-C9EnvironmentList
```

Output:

```
685f892f431b45c2b28cb69eadc0Ex
1980b80e5f584920801c09086667f0Ex
```

- For API details, see [ListEnvironments](https://aws.amazon.com) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-C9EnvironmentMembershipList**

The following code example shows how to use Get-C9EnvironmentMembershipList.

**Tools for PowerShell**

**Example 1:** This example gets information about environment members for the specified AWS Cloud9 development environment.
Get-C9EnvironmentMembershipList -EnvironmentId ffd88420d4824e3eaa8a04bfde8cEX

Output:

EnvironmentId : ffd88420d4824e3eaa8a04bfde8cEX
LastAccess    : 1/1/0001 12:00:00 AM
Permissions   : read-write
UserArn       : arn:aws:iam::123456789012:user/AnotherDemoUser
UserId        : AIDAJ3BA602FMJWCHXHEX

EnvironmentId : ffd88420d4824e3eaa8a04bfde8cEX
LastAccess    : 1/1/0001 12:00:00 AM
Permissions   : owner
UserArn       : arn:aws:iam::123456789012:user/MyDemoUser
UserId        : AIDAJ3LOROMOUXTBSU6EX

Example 2: This example gets information about the owner of the specified AWS Cloud9 development environment.

Get-C9EnvironmentMembershipList -EnvironmentId ffd88420d4824e3eaa8a04bfde8cEX -Permission owner

Output:

EnvironmentId : ffd88420d4824e3eaa8a04bfde8cEX
LastAccess    : 1/1/0001 12:00:00 AM
Permissions   : owner
UserArn       : arn:aws:iam::123456789012:user/MyDemoUser
UserId        : AIDAJ3LOROMOUXTBSU6EX

Example 3: This example gets information about the specified environment member for multiple AWS Cloud9 development environments.

Get-C9EnvironmentMembershipList -UserArn arn:aws:iam::123456789012:user/MyDemoUser

Output:

EnvironmentId : ffd88420d4824e3eaa8a04bfde8cEX
LastAccess    : 1/17/2018 7:48:14 PM
Permissions   : owner
For API details, see [DescribeEnvironmentMemberships](https://docs.aws.amazon.com/tools/latest/dg/aws-iam-powershell-samples.html#describe-environment-memberships) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-C9EnvironmentStatus

The following code example shows how to use `Get-C9EnvironmentStatus`.

**Tools for PowerShell**

**Example 1:** This example gets status information for the specified AWS Cloud9 development environment.

```powershell
Get-C9EnvironmentStatus -EnvironmentId 349c86d4579e4e7298d500ff57a6b2EX
```

**Output:**

<table>
<thead>
<tr>
<th>Message</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment is ready to use</td>
<td>ready</td>
</tr>
</tbody>
</table>

For API details, see [DescribeEnvironmentStatus](https://docs.aws.amazon.com/tools/latest/dg/aws-iam-powershell-samples.html#describe-environment-status) in *AWS Tools for PowerShell Cmdlet Reference*.

### New-C9EnvironmentEC2

The following code example shows how to use `New-C9EnvironmentEC2`.

**Tools for PowerShell**

**Example 1:** This example creates an AWS Cloud9 development environment with the specified settings, launches an Amazon Elastic Compute Cloud (Amazon EC2) instance, and then connects from the instance to the environment.
New-C9EnvironmentEC2 -Name my-demo-env -AutomaticStopTimeMinutes 60 -Description "My demonstration development environment." -InstanceType t2.micro -OwnerArn arn:aws:iam::123456789012:user/MyDemoUser -SubnetId subnet-d43a46EX

**Output:**

```bash
ffd88420d4824eeaeaa8a04bfde8cEX
```

- For API details, see [CreateEnvironmentEc2](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### New-C9EnvironmentMembership

The following code example shows how to use `New-C9EnvironmentMembership`.

**Tools for PowerShell**

**Example 1:** This example adds the specified environment member to the specified AWS Cloud9 development environment.

```bash
New-C9EnvironmentMembership -UserArn arn:aws:iam::123456789012:user/AnotherDemoUser -EnvironmentId ffd88420d4824eeaeaa8a04bfde8cEX -Permission read-write
```

**Output:**

```bash
EnvironmentId : ffd88420d4824eeaeaa8a04bfde8cEX
LastAccess    : 1/1/0001 12:00:00 AM
Permissions   : read-write
UserArn       : arn:aws:iam::123456789012:user/AnotherDemoUser
UserId        : AIDAJ3BA6O2FMJWEX
```

- For API details, see [CreateEnvironmentMembership](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Remove-C9Environment

The following code example shows how to use `Remove-C9Environment`.

```bash
New-C9EnvironmentMembership -UserArn arn:aws:iam::123456789012:user/AnotherDemoUser -EnvironmentId ffd88420d4824eeaeaa8a04bfde8cEX -Permission read-write
```

**Output:**

```bash
EnvironmentId : ffd88420d4824eeaeaa8a04bfde8cEX
LastAccess    : 1/1/0001 12:00:00 AM
Permissions   : read-write
UserArn       : arn:aws:iam::123456789012:user/AnotherDemoUser
UserId        : AIDAJ3BA6O2FMJWEX
```

- For API details, see [CreateEnvironmentMembership](#) in *AWS Tools for PowerShell Cmdlet Reference*. 
Tools for PowerShell

Example 1: This example deletes the specified AWS Cloud9 development environment. If an Amazon EC2 instance is connected to the environment, also terminates the instance.

```
Remove-C9Environment -EnvironmentId ffd88420d4824eeea8a04bfde8cEX
```

• For API details, see DeleteEnvironment in AWS Tools for PowerShell Cmdlet Reference.

Remove-C9EnvironmentMembership

The following code example shows how to use Remove-C9EnvironmentMembership.

Tools for PowerShell

Example 1: This example deletes the specified environment member from the specified AWS Cloud9 development environment.

```
Remove-C9EnvironmentMembership -UserArn arn:aws:iam::123456789012:user/AnotherDemoUser -EnvironmentId ffd88420d4824eeea8a04bfde8cEX
```

• For API details, see DeleteEnvironmentMembership in AWS Tools for PowerShell Cmdlet Reference.

Update-C9Environment

The following code example shows how to use Update-C9Environment.

Tools for PowerShell

Example 1: This example changes the specified settings of the specified existing AWS Cloud9 development environment.

```
Update-C9Environment -EnvironmentId ffd88420d4824eeea8a04bfde8cEX -Description "My changed demonstration development environment." -Name my-changed-demo-env
```

• For API details, see UpdateEnvironment in AWS Tools for PowerShell Cmdlet Reference.
Update-C9EnvironmentMembership

The following code example shows how to use Update-C9EnvironmentMembership.

Tools for PowerShell

Example 1: This example changes the settings of the specified existing environment member for the specified AWS Cloud9 development environment.

```
Update-C9EnvironmentMembership -UserArn arn:aws:iam::123456789012:user/AnotherDemoUser -EnvironmentId ffd88420d4824eeea8a04bfde8cEX -Permission read-only
```

Output:

```
EnvironmentId : ffd88420d4824eeea8a04bfde8cEX
LastAccess    : 1/1/0001 12:00:00 AM
Permissions   : read-only
UserArn       : arn:aws:iam::123456789012:user/AnotherDemoUser
UserId        : AIDAJ3BA6O2FMJW5CWXHEX
```

- For API details, see [UpdateEnvironmentMembership](#) in AWS Tools for PowerShell Cmdlet Reference.

AWS CloudFormation examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with AWS CloudFormation.

**Actions** are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

**Scenarios** are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics
• **Actions**

**Actions**

**Get-CFNStack**

The following code example shows how to use Get-CFNStack.

**Tools for PowerShell**

**Example 1:** Returns a collection of Stack instances describing all of the user's stacks.

```powershell
Get-CFNStack
```

**Example 2:** Returns a Stack instance describing the specified stack

```powershell
Get-CFNStack -StackName "myStack"
```

**Example 3:** Returns a collection of Stack instances describing all of the user's stacks using manual paging. The starting token for the next page is retrieved after every call with $null indicating no more details remain to be retrieved.

```powershell
$nextToken = $null
do {
    Get-CFNStack -NextToken $nextToken
    $nextToken = $AWSHistory.LastServiceResponse.NextToken
} while ($nextToken -ne $null)
```

- For API details, see [DescribeStacks](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-CFNStackEvent**

The following code example shows how to use Get-CFNStackEvent.

**Tools for PowerShell**

**Example 1:** Returns all stack related events for the specified stack.

```powershell
Get-CFNStackEvent -StackName "myStack"
```
Example 2: Returns all stack related events for the specified stack using manual paging starting at the specified token. The starting token for the next page is retrieved after every call with $null indicating no more events remain to be retrieved.

```powershell
$nextToken = $null
do {
    Get-CFNStack -StackName "myStack" -NextToken $nextToken
    $nextToken = $AWSHistory.LastServiceResponse.NextToken
} while ($nextToken -ne $null)
```

- For API details, see [DescribeStackEvents](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-CFNSStackResource

The following code example shows how to use Get-CFNSStackResource.

#### Tools for PowerShell

**Example 1:** Returns the description of a resource identified in the template associated with the specified stack by the logical ID "MyDBInstance".

```powershell
Get-CFNStackResource -StackName "myStack" -LogicalResourceId "MyDBInstance"
```

- For API details, see [DescribeStackResource](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-CFNSStackResourceList

The following code example shows how to use Get-CFNSStackResourceList.

#### Tools for PowerShell

**Example 1:** Returns the AWS resource descriptions for up to 100 resources associated with the specified stack. To obtain details of all resources associated with a stack use the Get-CFNSStackResourceSummary, which also supports manual paging of the results.

```powershell
Get-CFNStackResourceList -StackName "myStack"
```

**Example 2:** Returns the description of the Amazon EC2 instance identified in the template associated with the specified stack by the logical ID "Ec2Instance".
Example 3: Returns the description of up to 100 resources associated with the stack containing an Amazon EC2 instance identified by instance ID "i-123456". To obtain details of all resources associated with a stack use the Get-CFNStackResourceSummary, which also supports manual paging of the results.

```
Get-CFNStackResourceList -StackName "myStack" -LogicalResourceId "Ec2Instance"
```

Example 4: Returns the description of the Amazon EC2 instance identified by the logical ID "Ec2Instance" in the template for a stack. The stack is identified using the physical resource ID of a resource it contains, in this case also an Amazon EC2 instance with instance ID "i-123456". A different physical resource could also be used to identify the stack depending on the template content, for example an Amazon S3 bucket.

```
Get-CFNStackResourceList -PhysicalResourceId "i-123456" -LogicalResourceId "Ec2Instance"
```

- For API details, see [DescribeStackResources](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-CFNStackResourceSummary**

The following code example shows how to use Get-CFNStackResourceSummary.

**Tools for PowerShell**

**Example 1:** Returns descriptions of all the resources associated with the specified stack.

```
Get-CFNStackResourceSummary -StackName "myStack"
```

**Example 2:** Returns descriptions of all the resources associated with the specified stack using manual paging of the results. The starting token for the next page is retrieved after every call with $null indicating no more details remain to be retrieved.

```
$nextToken = $null
do {
    Get-CFNStackResourceSummary -StackName "myStack" -NextToken $nextToken
    $nextToken = $AWSHistory.LastServiceResponse.NextToken
} while ($nextToken)
For API details, see ListStackResources in AWS Tools for PowerShell Cmdlet Reference.

Get-CFNSummary

The following code example shows how to use Get-CFNSummary.

Example 1: Returns summary information for all stacks.

```powershell
Get-CFNSummary
```

Example 2: Returns summary information for all stacks that are currently being created.

```powershell
Get-CFNSummary -StackStatusFilter "CREATE_IN_PROGRESS"
```

Example 3: Returns summary information for all stacks that are currently being created or updated.

```powershell
Get-CFNSummary -StackStatusFilter @("CREATE_IN_PROGRESS", "UPDATE_IN_PROGRESS")
```

Example 4: Returns summary information for all stacks that are currently being created or updated using manual paging of the results. The starting token for the next page is retrieved after every call with $null indicating no more details remain to be retrieved.

```powershell
$nextToken = $null
do {
    Get-CFNSummary -StackStatusFilter @("CREATE_IN_PROGRESS", "UPDATE_IN_PROGRESS") -NextToken $nextToken
    $nextToken = $AWSHistory.LastServiceResponse.NextToken
} while ($nextToken -ne $null)
```

For API details, see ListStacks in AWS Tools for PowerShell Cmdlet Reference.

Get-CFNTemplate

The following code example shows how to use Get-CFNTemplate.
**Tools for PowerShell**

Example 1: Returns the template associated with the specified stack.

```
Get-CFNTemplate -StackName "myStack"
```

- For API details, see [GetTemplate](https://docs.aws.amazon.com/AWSToolsDocumentation/latest/AWSToolsForPowerShellCmdletReference/gt.Template.html) in *AWS Tools for PowerShell Cmdlet Reference*.

**Measure-CFNTemplateCost**

The following code example shows how to use Measure-CFNTemplateCost.

**Tools for PowerShell**

Example 1: Returns an AWS Simple Monthly Calculator URL with a query string that describes the resources required to run the template. The template is obtained from the specified Amazon S3 URL and the single customization parameter applied. The parameter can also be specified using 'Key' and 'Value' instead of 'ParameterKey' and 'ParameterValue'.

```
Measure-CFNTemplateCost -TemplateURL https://s3.amazonaws.com/mytemplates/templatefile.template
                          -Region us-west-1
                          -Parameter @{ ParameterKey="KeyName";
                                   ParameterValue="myKeyPairName" }
```

Example 2: Returns an AWS Simple Monthly Calculator URL with a query string that describes the resources required to run the template. The template is parsed from the supplied content and the customization parameters applied (this example assumes the template content would have declared two parameters, 'KeyName' and 'InstanceType'). The customization parameters can also be specified using 'Key' and 'Value' instead of 'ParameterKey' and 'ParameterValue'.

```
Measure-CFNTemplateCost -TemplateBody "{TEMPLATE CONTENT HERE}" 
                          -Parameter @( @{ ParameterKey="KeyName";
                                       ParameterValue="myKeyPairName" },
                                       @{ ParameterKey="InstanceType";
                                       ParameterValue="m1.large" })
```

Example 3: Uses New-Object to build the set of template parameters and returns an AWS Simple Monthly Calculator URL with a query string that describes the resources
required to run the template. The template is parsed from the supplied content, with customization parameters (this example assumes the template content would have declared two parameters, 'KeyName' and 'InstanceType').

```powershell
$p1 = New-Object -Type Amazon.CloudFormation.Model.Parameter
$p1.ParameterKey = "KeyName"
$p1.ParameterValue = "myKeyPairName"

$p2 = New-Object -Type Amazon.CloudFormation.Model.Parameter
$p2.ParameterKey = "InstanceType"
$p2.ParameterValue = "m1.large"

Measure-CFNTemplateCost -TemplateBody "{TEMPLATE CONTENT HERE}" -Parameter @( $p1, $p2 )
```

- For API details, see [EstimateTemplateCost](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**New-CFNSStack**

The following code example shows how to use `New-CFNSStack`.

**Tools for PowerShell**

**Example 1:** Creates a new stack with the specified name. The template is parsed from the supplied content with customization parameters ('PK1' and 'PK2' represent the names of parameters declared in the template content, 'PV1' and 'PV2' represent the values for those parameters. The customization parameters can also be specified using 'Key' and 'Value' instead of 'ParameterKey' and 'ParameterValue'. If creation of the stack fails, it will not be rolled back.

```powershell
New-CFNSStack -StackName "myStack" ` -TemplateBody "{TEMPLATE CONTENT HERE}" ` -Parameter @(@{ ParameterKey="PK1"; ParameterValue="PV1" }, @{ ParameterKey="PK2"; ParameterValue="PV2" }) ` -DisableRollback $true
```

**Example 2:** Creates a new stack with the specified name. The template is parsed from the supplied content with customization parameters ('PK1' and 'PK2' represent the names of parameters declared in the template content, 'PV1' and 'PV2' represent the values for those parameters. The customization parameters can also be specified using 'Key' and 'Value' instead of 'ParameterKey' and 'ParameterValue'. If creation of the stack fails, it will not be rolled back.
instead of 'ParameterKey' and 'ParameterValue'. If creation of the stack fails, it will be rolled back.

```powershell
$p1 = New-Object -Type Amazon.CloudFormation.Model.Parameter
$p1.ParameterKey = "PK1"
$p1.ParameterValue = "PV1"

$p2 = New-Object -Type Amazon.CloudFormation.Model.Parameter
$p2.ParameterKey = "PK2"
$p2.ParameterValue = "PV2"

New-CFNStack -StackName "myStack" `  
    -TemplateBody "{TEMPLATE CONTENT HERE}" `  
    -Parameter @($p1, $p2) `  
    -OnFailure "ROLLBACK"
```

Example 3: Creates a new stack with the specified name. The template is obtained from the Amazon S3 URL with customization parameters ('PK1' represents the name of a parameter declared in the template content, 'PV1' represents the value for the parameter. The customization parameters can also be specified using 'Key' and 'Value' instead of 'ParameterKey' and 'ParameterValue'. If creation of the stack fails, it will be rolled back (same as specifying -DisableRollback $false).

```powershell
New-CFNStack -StackName "myStack" `  
    -TemplateURL https://s3.amazonaws.com/mytemplates/templatefile.template `  
    -Parameter @{ParameterKey="PK1"; ParameterValue="PV1"}
```

Example 4: Creates a new stack with the specified name. The template is obtained from the Amazon S3 URL with customization parameters ('PK1' represents the name of a parameter declared in the template content, 'PV1' represents the value for the parameter. The customization parameters can also be specified using 'Key' and 'Value' instead of 'ParameterKey' and 'ParameterValue'. If creation of the stack fails, it will be rolled back (same as specifying -DisableRollback $false). The specified notification AENs will receive published stack-related events.

```powershell
New-CFNStack -StackName "myStack" `  
    -TemplateURL https://s3.amazonaws.com/mytemplates/templatefile.template `  
    -Parameter @{ParameterKey="PK1"; ParameterValue="PV1"}
```
-NotificationARN @('arn1', 'arn2')

- For API details, see CreateStack in AWS Tools for PowerShell Cmdlet Reference.

Remove-CFNStack

The following code example shows how to use Remove-CFNStack.

Tools for PowerShell

Example 1: Deletes the specified stack.

Remove-CFNStack -StackName "myStack"

- For API details, see DeleteStack in AWS Tools for PowerShell Cmdlet Reference.

Resume-CFNUpdateRollback

The following code example shows how to use Resume-CFNUpdateRollback.

Tools for PowerShell

Example 1: Continues rollback of the named stack, which should be in the state 'UPDATE_ROLLBACK_FAILED'. If the continued rollback is successful, the stack will enter state 'UPDATE_ROLLBACK_COMPLETE'.

Resume-CFNUpdateRollback -StackName "myStack"

- For API details, see ContinueUpdateRollback in AWS Tools for PowerShell Cmdlet Reference.

Stop-CFNUpdateStack

The following code example shows how to use Stop-CFNUpdateStack.

Tools for PowerShell

Example 1: Cancels an update on the specified stack.

Stop-CFNUpdateStack -StackName "myStack"

- For API details, see CancelUpdateStack in AWS Tools for PowerShell Cmdlet Reference.
Test-CFNStack

The following code example shows how to use Test-CFNStack.

Tools for PowerShell

Example 1: Tests if the stack has reached one of the states UPDATE_ROLLBACK_COMPLETE, CREATE_COMPLETE, ROLLBACK_COMPLETE or UPDATE_COMPLETE.

```
Test-CFNStack -StackName MyStack
```

Output:

```
False
```

Example 2: Tests if the stack has reached a status of either UPDATE_COMPLETE or UPDATE_ROLLBACK_COMPLETE.

```
Test-CFNStack -StackName MyStack -Status UPDATE_COMPLETE,UPDATE_ROLLBACK_COMPLETE
```

Output:

```
True
```

- For API details, see Test-CFNStack in AWS Tools for PowerShell Cmdlet Reference.

Test-CFNTemplate

The following code example shows how to use Test-CFNTemplate.

Tools for PowerShell

Example 1: Validates the specified template content. The output details the capabilities, description and parameters of the template.

```
Test-CFNTemplate -TemplateBody "{TEMPLATE CONTENT HERE}"
```

Example 2: Validates the specified template accessed via an Amazon S3 URL. The output details the capabilities, description and parameters of the template.

- For API details, see ValidateTemplate in AWS Tools for PowerShell Cmdlet Reference.

Update-CFNStack

The following code example shows how to use Update-CFNStack.

Tools for PowerShell

Example 1: Updates the stack 'myStack' with the specified template and customization parameters. 'PK1' represents the name of a parameter declared in the template and 'PV1' represents its value. The customization parameters can also be specified using 'Key' and 'Value' instead of 'ParameterKey' and 'ParameterValue'.

Update-CFNStack -StackName "myStack" ` 
    -TemplateBody "{Template Content Here}" ` 
    -Parameter @{ ParameterKey="PK1"; ParameterValue="PV1" }

Example 2: Updates the stack 'myStack' with the specified template and customization parameters. 'PK1' and 'PK2' represent the names of parameters declared in the template, 'PV1' and 'PV2' represents their requested values. The customization parameters can also be specified using 'Key' and 'Value' instead of 'ParameterKey' and 'ParameterValue'.

Update-CFNStack -StackName "myStack" ` 
    -TemplateBody "{Template Content Here}" ` 
    -Parameter @([ ParameterKey="PK1"; ParameterValue="PV1" ],
      @{ ParameterKey="PK2"; ParameterValue="PV2" })

Example 3: Updates the stack 'myStack' with the specified template and customization parameters. 'PK1' represents the name of a parameter declared in the template and 'PV2' represents its value. The customization parameters can also be specified using 'Key' and 'Value' instead of 'ParameterKey' and 'ParameterValue'.

Update-CFNStack -StackName "myStack" -TemplateBody "{Template Content Here}" -Parameters @([ ParameterKey="PK1"; ParameterValue="PV1" ]

Example 4: Updates the stack 'myStack' with the specified template, obtained from Amazon S3, and customization parameters. 'PK1' and 'PK2' represent the names of parameters declared in the template, 'PV1' and 'PV2' represents their requested values. The customization parameters can also be specified using 'Key' and 'Value' instead of 'ParameterKey' and 'ParameterValue'.

```
Update-CFNStack -StackName "myStack" \\
    -TemplateURL https://s3.amazonaws.com/mytemplates/templatefile.template \\
    -Parameter @(
        @{ ParameterKey="PK1"; ParameterValue="PV1" },
        @{ ParameterKey="PK2"; ParameterValue="PV2" }
    )
```

Example 5: Updates the stack 'myStack', which is assumed in this example to contain IAM resources, with the specified template, obtained from Amazon S3, and customization parameters. 'PK1' and 'PK2' represent the names of parameters declared in the template, 'PV1' and 'PV2' represents their requested values. The customization parameters can also be specified using 'Key' and 'Value' instead of 'ParameterKey' and 'ParameterValue'. Stacks containing IAM resources require you to specify the -Capabilities "CAPABILITY_IAM" parameter otherwise the update will fail with an 'InsufficientCapabilities' error.

```
Update-CFNStack -StackName "myStack" \\
    -TemplateURL https://s3.amazonaws.com/mytemplates/templatefile.template \\
    -Parameter @(
        @{ ParameterKey="PK1"; ParameterValue="PV1" },
        @{ ParameterKey="PK2"; ParameterValue="PV2" }
    ) \\
    -Capabilities "CAPABILITY_IAM"
```

- For API details, see UpdateStack in AWS Tools for PowerShell Cmdlet Reference.

Wait-CFNStack

The following code example shows how to use Wait-CFNStack.

Tools for PowerShell

Example 1: Tests if the stack has reached one of the states UPDATE_ROLLBACK_COMPLETE, CREATE_COMPLETE, ROLLBACK_COMPLETE or UPDATE_COMPLETE. If the stack is not at one of the states the command sleeps for two seconds before testing status again. This is repeated until the stack reaches one of the requested states or the default timeout period.
of 60 seconds elapses. If the timeout period is exceeded an exception is thrown. If the stack reaches one of the requested states within the timeout period it is returned to the pipeline.

```powershell
$stack = Wait-CFNStack -StackName MyStack
```

Example 2: This example waits for a total of 5 minutes (300 seconds) for the stack to reach either of the specified states. In this example the state is reached before the timeout and therefore the stack object is returned to the pipeline.

```powershell
Wait-CFNStack -StackName MyStack -Timeout 300 -Status CREATE_COMPLETE,ROLLBACK_COMPLETE
```

Output:

```
Capabilities      : {CAPABILITY_IAM}
ChangeSetId       :
CreationTime      : 6/1/2017 9:29:33 AM
Description       : AWS CloudFormation Sample Template
c2_instance_with_instance_profile: Create an EC2 instance with an associated instance profile. **WARNING** This template creates one or more Amazon EC2 instances and an Amazon SQS queue. You will be billed for the AWS resources used if you create a stack from this template.
DisableRollback   : False
LastUpdatedTime   : 1/1/0001 12:00:00 AM
NotificationARNs  :
Outputs           :
Parameters        :
RoleARN           :
StackId           : arn:aws:cloudformation:us-west-2:123456789012:stack/MyStack/7ea87b50-46e7-11e7-9c9b-503a90a9c4d1
StackName         : MyStack
StackStatus       : CREATE_COMPLETE
StackStatusReason :
Tags              :
TimeoutInMinutes  : 0
```

Example 3: This example shows the error output when a stack does not reach one of the requested states within the timeout period (in this case the default period of 60 seconds).

```powershell
Wait-CFNStack -StackName MyStack -Status CREATE_COMPLETE,ROLLBACK_COMPLETE
```
Output:

```
Wait-CFNStack : Timed out after 60 seconds waiting for CloudFormation stack MyStack in region us-west-2 to reach one of state(s): UPDATE_ROLLBACK_COMPLETE,CREATE_COMPLETE,ROLLBACK_COMPLETE,UPDATE_COMPLETE
At line:1 char:1
+ Wait-CFNStack -StackName MyStack -State CREATE_COMPLETE,ROLLBACK_COMPLETE
  + CategoryInfo : InvalidOperation:
    (Amazon.PowerShell.Cmdlets.CFNScripting.WaitCFNStackCmdlet:WaitCFNStackCmdlet) [Wait-CFNStack],
    InvalidOperationException
    + FullyQualifiedErrorId :
      InvalidOperationException,Amazon.PowerShell.Cmdlets.CFN.WaitCFNStackCmdlet
```

- For API details, see [Wait-CFNStack](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**CloudFront examples using Tools for PowerShell**

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with CloudFront.

*Actions* are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

*Scenarios* are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

**Topics**

- **Actions**

**Actions**

**Get-CFCloudFrontOriginAccessIdentity**

The following code example shows how to use Get-CFCloudFrontOriginAccessIdentity.
Example 1: This example returns a specific Amazon CloudFront origin access identity, specified by the -Id parameter. Although the -Id parameter is not required, if you do not specify it, no results are returned.

```powershell
Get-CFCloudFrontOriginAccessIdentity -Id E3XXXXXXXXXXRT
```

Output:

```
CloudFrontOriginAccessIdentityConfig    Id
S3CanonicalUserId                      --
------------------------------------    --
-----------------
Amazon.CloudFront.Model.CloudFrontOr... E3XXXXXXXXXXRT
4b6e...
```

- For API details, see [GetCloudFrontOriginAccessIdentity](#) in AWS Tools for PowerShell Cmdlet Reference.

### Get-CFCloudFrontOriginAccessIdentityConfig

The following code example shows how to use Get-CFCloudFrontOriginAccessIdentityConfig.

Example 1: This example returns configuration information about a single Amazon CloudFront origin access identity, specified by the -Id parameter. Errors occur if no -Id parameter is specified.

```powershell
Get-CFCloudFrontOriginAccessIdentityConfig -Id E3XXXXXXXXXXRT
```

Output:

```
CallerReference                                             Comment
---------------                                             -------
mycallerreference: 2/1/2011 1:16:32 PM                      Caller reference:
2/1/2011 1:16:32 PM                                           
```

CloudFront
For API details, see `GetCloudFrontOriginAccessIdentityConfig` in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-CFCloudFrontOriginAccessIdentityList**

The following code example shows how to use `Get-CFCloudFrontOriginAccessIdentityList`.

**Tools for PowerShell**

**Example 1:** This example returns a list of Amazon CloudFront origin access identities. Because the `-MaxItem` parameter specifies a value of 2, the results include two identities.

```
Get-CFCloudFrontOriginAccessIdentityList -MaxItem 2
```

**Output:**

```
IsTruncated : True
Items       : {E326XXXXXXXXXT, E1YWXXXXXXXX9B}
Marker      :
MaxItems    : 2
NextMarker  : E1YXXXXXXXXX9B
Quantity    : 2
```

For API details, see `ListCloudFrontOriginAccessIdentities` in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-CFDistribution**

The following code example shows how to use `Get-CFDistribution`.

**Tools for PowerShell**

**Example 1:** Retrieves the information for a specific distribution.

```
Get-CFDistribution -Id EXAMPLE0000ID
```

For API details, see `GetDistribution` in *AWS Tools for PowerShell Cmdlet Reference*. 

CloudFront
Get-CFDistributionConfig

The following code example shows how to use Get-CFDistributionConfig.

Tools for PowerShell

Example 1: Retrieves the configuration for a specific distribution.

```
Get-CFDistributionConfig -Id EXAMPLE0000ID
```

- For API details, see `GetDistributionConfig` in *AWS Tools for PowerShell Cmdlet Reference*.

Get-CFDistributionList

The following code example shows how to use Get-CFDistributionList.

Tools for PowerShell

Example 1: Returns distributions.

```
Get-CFDistributionList
```

- For API details, see `ListDistributions` in *AWS Tools for PowerShell Cmdlet Reference*.

New-CFDistribution

The following code example shows how to use New-CFDistribution.

Tools for PowerShell

Example 1: Creates a basic CloudFront distribution, configured with logging and caching.

```
$origin = New-Object Amazon.CloudFront.Model.Origin
$origin.DomainName = "ps-cmdlet-sample.s3.amazonaws.com"
$origin.Id = "UniqueOrigin1"
$origin.S3OriginConfig.OriginAccessIdentity = ""
New-CFDistribution
    -DistributionConfig_Enabled $true
    -DistributionConfig_Comment "Test distribution"
```
AWS Tools for PowerShell

Example 1: This example creates a new invalidation on a distribution with an ID of EXAMPLENSTXAXE. The CallerReference is a unique ID chosen by the user; in this case, a time stamp representing May 15, 2019 at 9:00 a.m. is used. The $Paths variable stores three paths to image and media files that the user does not want as part of the distribution's cache. The -Paths_Quantity parameter value is the total number of paths specified in the -Paths_Item parameter.

```powershell
$Paths = "/images/*.gif", "/images/image1.jpg", "/videos/*.mp4"
New-CFInvalidation -DistributionId "EXAMPLENSTXAXE" -InvalidationBatch_CallerReference 20190515090000 -Paths_Item $Paths -Paths_Quantity 3
```

Output:
For API details, see &lt;a href="CreateInvalidation"&gt;CreateInvalidation&lt;/a&gt; in &lt;i&gt;AWS Tools for PowerShell Cmdlet Reference.&lt;/i&gt;
"ExpiresOn"=$start.AddDays(7)
   "ActiveFrom"=$start
}

New-CFSignedCookie @params

Output:

Policy
------
[CloudFront-Policy, eyJTd...wIjo...

Example 3: Creates a signed cookie to the specified resources using a custom policy. The cookie will be valid in 24 hours and will expire one week afterward. Access to the resources is restricted to the specified ip range.

$start = (Get-Date).AddHours(24)
$params = @{
   "ResourceUri"="http://xyz.cloudfront.net/content/*.jpeg"
   "KeyPairId"="AKIAIOSFODNN7EXAMPLE"
   "PrivateKeyFile"="C:\pk-AKIAIOSFODNN7EXAMPLE.pem"
   "ExpiresOn"=$start.AddDays(7)
   "ActiveFrom"=$start
   "IpRange"="192.0.2.0/24"
}

New-CFSignedCookie @params

Output:

Policy
------
[CloudFront-Policy, eyJTd...wIjo...

• For API details, see New-CFSignedCookie in AWS Tools for PowerShell Cmdlet Reference.

New-CFSignedUrl

The following code example shows how to use New-CFSignedUrl.
Tools for PowerShell

Example 1: Creates a signed url to the specified resource using a canned policy. The url will be valid for one hour. A System.Uri object containing the signed url is emitted to the pipeline.

```powershell
$params = @(
    "ResourceUri"="https://cdn.example.com/index.html"
    "KeyPairId"="AKIAIOSFODNN7EXAMPLE"
    "PrivateKeyFile"="C:\pk-AKIAIOSFODNN7EXAMPLE.pem"
    "ExpiresOn"=(Get-Date).AddHours(1)
)
New-CFSignedUrl @params
```

Example 2: Creates a signed url to the specified resource using a custom policy. The url will be valid starting in 24 hours and will expire one week later.

```powershell
$start = (Get-Date).AddHours(24)
$params = @(
    "ResourceUri"="https://cdn.example.com/index.html"
    "KeyPairId"="AKIAIOSFODNN7EXAMPLE"
    "PrivateKeyFile"="C:\pk-AKIAIOSFODNN7EXAMPLE.pem"
    "ExpiresOn"=(Get-Date).AddDays(7)
    "ActiveFrom"=$start
)
New-CFSignedUrl @params
```

Example 3: Creates a signed url to the specified resource using a custom policy. The url will be valid starting in 24 hours and will expire one week later. Access to the resource is restricted to the specified ip range.

```powershell
$start = (Get-Date).AddHours(24)
$params = @(
    "ResourceUri"="https://cdn.example.com/index.html"
    "KeyPairId"="AKIAIOSFODNN7EXAMPLE"
    "PrivateKeyFile"="C:\pk-AKIAIOSFODNN7EXAMPLE.pem"
    "ExpiresOn"=(Get-Date).AddDays(7)
    "ActiveFrom"=$start
    "IpRange"="192.0.2.0/24"
)
New-CFSignedUrl @params
```
CloudTrail examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with CloudTrail.

*Actions* are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

*Scenarios* are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

**Topics**

- **Actions**

**Actions**

**Find-CTEvent**

The following code example shows how to use `Find-CTEvent`.

**Tools for PowerShell**

**Example 1**: Returns all events that have occurred over the last seven days. The cmdlet by default automatically makes multiple calls to deliver all events, exiting when the service indicates no further data is available.

```
Find-CTEvent
```

**Example 2**: Returns all events that have occurred over the last seven days specifying a region that is not the current shell default.

```
Find-CTEvent -Region eu-central-1
```
Example 3: Returns all events that are associated with the RunInstances API call.

Find-CTEvent -LookupAttribute @{ AttributeKey="EventName"; AttributeValue="RunInstances" }

Example 4: Returns the first 5 available events. The token to use to retrieve further events is attached as a note property named 'NextToken' to the $AWSHistory.LastServiceResponse member.

Find-CTEvent -MaxResult 5

Example 5: Returns the next 10 events using the 'next page' token from a previous call to indicate where to start returning events from in the sequence.

Find-CTEvent -MaxResult 10 -NextToken $AWSHistory.LastServiceResponse.NextToken

Example 6: This example shows how to loop through the available events using manual paging, fetching a maximum of 5 events per call.

$nextToken = $null
do
{
    Find-CTEvent -MaxResult 5 -NextToken $nextToken
    $nextToken = $AWSHistory.LastServiceResponse.NextToken
} while ($nextToken -ne $null)

- For API details, see [LookupEvents](#) in *AWS Tools for PowerShell Cmdlet Reference*.  

Get-CTTrail

The following code example shows how to use Get-CTTrail.

Tools for PowerShell

Example 1: Returns the settings of all trails associated with the current region for your account.

Get-CTTrail
Example 2: Returns the settings for the specified trails.

```
Get-CTTrail -TrailNameList trail1,trail2
```

Example 3: Returns the settings for the specified trails that were created in a region other than the current shell default (in this case the Frankfurt (eu-central-1) region).

```
Get-CTTrail -TrailNameList trailABC,trailDEF -Region eu-central-1
```

• For API details, see [DescribeTrails](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

---

**Get-CTTrailStatus**

The following code example shows how to use Get-CTTrailStatus.

**Tools for PowerShell**

Example 1: Returns status information for the trail with name 'myExampleTrail'. Returned data includes information on delivery errors, Amazon SNS and Amazon S3 errors, and start and stop logging times for the trail. This example assumes the trail was created in the same region as the current shell default.

```
Get-CTTrailStatus -Name myExampleTrail
```

Example 2: Returns status information for a trail that was created in a region other than the current shell default (in this case, the Frankfurt (eu-central-1) region).

```
Get-CTTrailStatus -Name myExampleTrail -Region eu-central-1
```

• For API details, see [GetTrailStatus](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

---

**New-CTTrail**

The following code example shows how to use New-CTTrail.

**Tools for PowerShell**

Example 1: Creates a trail that will use the bucket 'mycloudtrailbucket' for log file storage.
New-CTTrail -Name="awscloudtrail-example" -S3BucketName="mycloudtrailbucket"

Example 2: Creates a trail that will use the bucket 'mycloudtrailbucket' for log file storage. The S3 objects representing the logs will have a common key prefix of 'mylogs'. When new logs are delivered to the bucket a notification will be sent to the SNS topic 'mlog-deliverytopic'. This example using splatting to supply the parameter values to the cmdlet.

```powershell
$params = @{
    Name="awscloudtrail-example"
    S3BucketName="mycloudtrailbucket"
    S3KeyPrefix="mylogs"
    SnsTopicName="mlog-deliverytopic"
}
New-CTTrail @params
```

- For API details, see CreateTrail in AWS Tools for PowerShell Cmdlet Reference.

Remove-CTTrail

The following code example shows how to use Remove-CTTrail.

Tools for PowerShell

Example 1: Deletes the specified trail. You will be prompted for confirmation before the command is run. To suppress confirmation, add the -Force switch parameter.

```powershell
Remove-CTTrail -Name "awscloudtrail-example"
```

- For API details, see DeleteTrail in AWS Tools for PowerShell Cmdlet Reference.

Start-CTLogging

The following code example shows how to use Start-CTLogging.

Tools for PowerShell

Example 1: Starts the recording of AWS API calls and log file delivery for the trail named 'myExampleTrail'. This example assumes the trail was created in the same region as the current shell default.
Example 2: Starts the recording of AWS API calls and log file delivery for a trail that was created in a region other than the current shell default (in this case, the Frankfurt (eu-central-1) region).

Start-CTLogging -Name myExampleTrail -Region eu-central-1

For API details, see StartLogging in AWS Tools for PowerShell Cmdlet Reference.

Stop-CTLogging

The following code example shows how to use Stop-CTLogging.

Tools for PowerShell

Example 1: Suspends the recording of AWS API calls and log file delivery for the trail named 'myExampleTrail'. This example assumes the trail was created in the same region as the current shell default.

Stop-CTLogging -Name myExampleTrail

Example 2: Suspends the recording of AWS API calls and log file delivery for a trail that was created in a region other than the current shell default (in this case, the Frankfurt (eu-central-1) region).

Stop-CTLogging -Name myExampleTrail -Region eu-central-1

For API details, see StopLogging in AWS Tools for PowerShell Cmdlet Reference.

Update-CTTrail

The following code example shows how to use Update-CTTrail.

Tools for PowerShell

Example 1: Updates the specified trail so that global service events (such as those from IAM) are recorded and changes the common key prefix of the log files going forwards to be 'globallogs'.
Example 2: Updates the specified trail so notifications about new log deliveries are sent to the specified SNS topic.

Update-CTTrail -Name "awscloudtrail-example" -SnsTopicName "mlog-deliverytopic2"

Example 3: Updates the specified trail so logs are delivered to a different bucket.

Update-CTTrail -Name "awscloudtrail-example" -S3BucketName "otherlogs"

For API details, see UpdateTrail in AWS Tools for PowerShell Cmdlet Reference.

CloudWatch examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with CloudWatch.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

- Actions

Actions

Get-CWDashboard

The following code example shows how to use Get-CWDashboard.
Example 1: Returns the arn the body of the specified dashboard.

Get-CWDashboard -DashboardName Dashboard1

Output:

<table>
<thead>
<tr>
<th>DashboardArn</th>
<th>DashboardBody</th>
</tr>
</thead>
<tbody>
<tr>
<td>arn:aws:cloudwatch::123456789012:dashboard/Dashboard1 {</td>
<td></td>
</tr>
</tbody>
</table>

- For API details, see [GetDashboard](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-CWDashboardList

The following code example shows how to use Get-CWDashboardList.

Example 1: Returns the collection of dashboards for your account.

Get-CWDashboardList

Output:

<table>
<thead>
<tr>
<th>DashboardArn</th>
<th>DashboardName</th>
<th>LastModified</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>arn:...</td>
<td>Dashboard1</td>
<td>7/6/2017 8:14:15 PM</td>
<td>252</td>
</tr>
</tbody>
</table>

Example 2: Returns the collection of dashboards for your account whose names start with the prefix 'dev'.

Get-CWDashboardList -DashboardNamePrefix dev

- For API details, see [ListDashboards](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Remove-CWDashboard

The following code example shows how to use Remove-CWDashboard.
Tools for PowerShell

Example 1: Deletes the specified dashboard, promoting for confirmation before proceeding. To bypass confirmation add the -Force switch to the command.

Remove-CWDashboard -DashboardName Dashboard1

• For API details, see DeleteDashboards in AWS Tools for PowerShell Cmdlet Reference.

Write-CWDashboard

The following code example shows how to use Write-CWDashboard.

Tools for PowerShell

Example 1: Creates or updates the dashboard named 'Dashboard1' to include two metric widgets side by side.

```
$dashBody = @
{
    "widgets": [
        {
            "type": "metric",
            "x": 0,
            "y": 0,
            "width": 12,
            "height": 6,
            "properties": {
                "metrics": [
                    ["AWS/EC2",
                    "CPUUtilization",
                    "InstanceId",
                    "i-012345"
                    ],
                "period": 300,
                "stat": "Average",
                "region": "us-east-1",
                "title": "EC2 Instance CPU"
                ]
            }
        }
    ],
    "period": 300,
    "stat": "Average",
    "region": "us-east-1",
    "title": "EC2 Instance CPU"
}
```
Example 2: Creates or updates the dashboard, piping the content describing the dashboard into the cmdlet.

```powershell
$dashBody = @"{
  ...
}"

$dashBody | Write-CWDashboard -DashboardName Dashboard1
```

- For API details, see **PutDashboard** in *AWS Tools for PowerShell Cmdlet Reference*. 

```powershell
Write-CWDashboard -DashboardName Dashboard1 -DashboardBody $dashBody
```
Write-CWMetricData

The following code example shows how to use Write-CWMetricData.

Tools for PowerShell

Example 1: Creates a new MetricDatum object, and writes it to Amazon Web Services CloudWatch Metrics.

```powershell
### Create a MetricDatum .NET object
$Metric = New-Object -TypeName Amazon.CloudWatch.Model.MetricDatum
$Metric.Timestamp = [DateTime]:UTCNow
$Metric.MetricName = 'CPU'
$Metric.Value = 50

### Write the metric data to the CloudWatch service
Write-CWMetricData -Namespace instance1 -MetricData $Metric
```

- For API details, see [PutMetricData](#) in *AWS Tools for PowerShell Cmdlet Reference*.

CodeCommit examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with CodeCommit.

*Actions* are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

*Scenarios* are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

**Topics**

- [Actions](#)
Actions

Get-CCBranch

The following code example shows how to use Get-CCBranch.

Tools for PowerShell

Example 1: This example gets information about the specified branch for the specified repository.

Get-CCBranch -RepositoryName MyDemoRepo -BranchName MyNewBranch

Output:

<table>
<thead>
<tr>
<th>BranchName</th>
<th>CommitId</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyNewBranch</td>
<td>7763222d...561fc9c9</td>
</tr>
</tbody>
</table>

For API details, see GetBranch in AWS Tools for PowerShell Cmdlet Reference.

Get-CCBranchList

The following code example shows how to use Get-CCBranchList.

Tools for PowerShell

Example 1: This example gets a list of branch names for the specified repository.

Get-CCBranchList -RepositoryName MyDemoRepo

Output:

master
MyNewBranch

For API details, see ListBranches in AWS Tools for PowerShell Cmdlet Reference.

Get-CCRepository

The following code example shows how to use Get-CCRepository.
Tools for PowerShell

**Example 1: This example gets information for the specified repository.**

```powershell
Get-CCRepository -RepositoryName MyDemoRepo
```

**Output:**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccountId</td>
<td>80398EXAMPLE</td>
</tr>
<tr>
<td>Arn</td>
<td>arn:aws:codecommit:us-east-1:80398EXAMPLE:MyDemoRepo</td>
</tr>
<tr>
<td>CloneUrlHttp</td>
<td><a href="https://git-codecommit.us-east-1.amazonaws.com/v1/repos/MyDemoRepo">https://git-codecommit.us-east-1.amazonaws.com/v1/repos/MyDemoRepo</a></td>
</tr>
<tr>
<td>CloneUrlSsh</td>
<td>ssh://git-codecommit.us-east-1.amazonaws.com/v1/repos/MyDemoRepo</td>
</tr>
<tr>
<td>CreationDate</td>
<td>9/8/2015 3:21:33 PM</td>
</tr>
<tr>
<td>DefaultBranch</td>
<td></td>
</tr>
<tr>
<td>LastModifiedDate</td>
<td>9/8/2015 3:21:33 PM</td>
</tr>
<tr>
<td>RepositoryDescription</td>
<td>This is a repository for demonstration purposes.</td>
</tr>
<tr>
<td>RepositoryId</td>
<td>c7d0d2b0-ce40-4303-b4c3-38529EXAMPLE</td>
</tr>
<tr>
<td>RepositoryName</td>
<td>MyDemoRepo</td>
</tr>
</tbody>
</table>

- For API details, see [GetRepository](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-CCRepositoryBatch**

The following code example shows how to use `Get-CCRepositoryBatch`.

**Tools for PowerShell**

**Example 1: This example confirms which of the specified repositories are found and not found.**

```powershell
Get-CCRepositoryBatch -RepositoryName MyDemoRepo, MyNewRepo, AMissingRepo
```

**Output:**

<table>
<thead>
<tr>
<th>Repositories</th>
<th>RepositoriesNotFound</th>
</tr>
</thead>
<tbody>
<tr>
<td>{MyDemoRepo, MyNewRepo}</td>
<td>{AMissingRepo}</td>
</tr>
</tbody>
</table>
For API details, see [BatchGetRepositories](#) in *AWS Tools for PowerShell Cmdlet Reference*. 

### Get-CCRepositoryList

The following code example shows how to use Get-CCRepositoryList.

**Tools for PowerShell**

**Example 1:** This example lists all repositories in ascending order by repository name.

```powershell
Get-CCRepositoryList -Order Ascending -SortBy RepositoryName
```

**Output:**

<table>
<thead>
<tr>
<th>RepositoryId</th>
<th>RepositoryName</th>
</tr>
</thead>
<tbody>
<tr>
<td>c7d0d2b0-ce40-4303-b4c3-38529EXAMPLE</td>
<td>MyDemoRepo</td>
</tr>
<tr>
<td>05f30c66-e3e3-4f91-a0cd-1c84aEXAMPLE</td>
<td>MyNewRepo</td>
</tr>
</tbody>
</table>

For API details, see [ListRepositories](#) in *AWS Tools for PowerShell Cmdlet Reference*. 

### New-CCBranch

The following code example shows how to use New-CCBranch.

**Tools for PowerShell**

**Example 1:** This example creates a new branch with the specified name for the specified repository and the specified commit ID.

```powershell
New-CCBranch -RepositoryName MyDemoRepo -BranchName MyNewBranch -CommitId 7763222d...561fc9c9
```

For API details, see [CreateBranch](#) in *AWS Tools for PowerShell Cmdlet Reference*. 

### New-CCRepository

The following code example shows how to use New-CCRepository.
Example 1: This example creates a new repository with the specified name and the specified description.

```
New-CCRepository -RepositoryName MyDemoRepo -RepositoryDescription "This is a repository for demonstration purposes."
```

Output:

<table>
<thead>
<tr>
<th>AccountId</th>
<th>80398EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arn</td>
<td>arn:aws:codecommit:us-east-1:80398EXAMPLE:MyDemoRepo</td>
</tr>
<tr>
<td>CloneUrlHttp</td>
<td><a href="https://git-codecommit.us-east-1.amazonaws.com/v1/repos/MyDemoRepo">https://git-codecommit.us-east-1.amazonaws.com/v1/repos/MyDemoRepo</a></td>
</tr>
<tr>
<td>MyDemoRepo</td>
<td></td>
</tr>
<tr>
<td>CloneUrlSsh</td>
<td>ssh://git-codecommit.us-east-1.amazonaws.com/v1/repos/MyDemoRepo</td>
</tr>
<tr>
<td>CreationDate</td>
<td>9/18/2015 4:13:25 PM</td>
</tr>
<tr>
<td>DefaultBranch</td>
<td></td>
</tr>
<tr>
<td>LastModifiedDate</td>
<td>9/18/2015 4:13:25 PM</td>
</tr>
<tr>
<td>RepositoryDescription</td>
<td>This is a repository for demonstration purposes.</td>
</tr>
<tr>
<td>RepositoryId</td>
<td>43ef2443-3372-4b12-9e78-65c27EXAMPLE</td>
</tr>
<tr>
<td>RepositoryName</td>
<td>MyDemoRepo</td>
</tr>
</tbody>
</table>

For API details, see [CreateRepository](#) in AWS Tools for PowerShell Cmdlet Reference.

Remove-CCRepository

The following code example shows how to use Remove-CCRepository.

Example 1: This example forcibly deletes the specified repository. The command will prompt for confirmation before proceeding. Add the -Force parameter to delete the repository without a prompt.

```
Remove-CCRepository -RepositoryName MyDemoRepo
```

Output:

```
43ef2443-3372-4b12-9e78-65c27EXAMPLE
```
For API details, see DeleteRepository in AWS Tools for PowerShell Cmdlet Reference.

**Update-CCDefaultBranch**

The following code example shows how to use Update-CCDefaultBranch.

**Tools for PowerShell**

**Example 1: This example changes the default branch for the specified repository to the specified branch.**

```
Update-CCDefaultBranch -RepositoryName MyDemoRepo -DefaultBranchName MyNewBranch
```

For API details, see UpdateDefaultBranch in AWS Tools for PowerShell Cmdlet Reference.

**Update-CCRepositoryDescription**

The following code example shows how to use Update-CCRepositoryDescription.

**Tools for PowerShell**

**Example 1: This example changes the description for the specified repository.**

```
Update-CCRepositoryDescription -RepositoryName MyDemoRepo -RepositoryDescription "This is an updated description."
```

For API details, see UpdateRepositoryDescription in AWS Tools for PowerShell Cmdlet Reference.

**Update-CCRepositoryName**

The following code example shows how to use Update-CCRepositoryName.

**Tools for PowerShell**

**Example 1: This example changes the name of the specified repository.**

```
Update-CCRepositoryName -NewName MyDemoRepo2 -OldName MyDemoRepo
```

For API details, see UpdateRepositoryName in AWS Tools for PowerShell Cmdlet Reference.
CodeDeploy examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with CodeDeploy.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

- Actions

Actions

Add-CDOnPremiseInstanceTag

The following code example shows how to use Add-CDOnPremiseInstanceTag.

Tools for PowerShell

Example 1: This example adds an on-premises instance tag with the specified key and value for the specified on-premises instance.

Add-CDOnPremiseInstanceTag -InstanceName AssetTag12010298EX -Tag @"Key" = "Name"; "Value" = "CodeDeployDemo-OnPrem"

- For API details, see AddTagsToOnPremisesInstances in AWS Tools for PowerShell Cmdlet Reference.

Get-CDApplication

The following code example shows how to use Get-CDApplication.
Example 1: This example gets information about the specified application.

```
Get-CDAplication -ApplicationName CodeDeployDemoApplication
```

Output:

```
ApplicationId                           ApplicationName          CreateTime
-------------                           ---------------          ----------
--------------                           ---------------          --------
e07fb938-091e-4f2f-8963-4d3e8EX         CodeDeployDemoApplication 7/20/2015 9:49:48 PM    False
```

For API details, see [GetApplication](https://aws.amazon.com) in AWS Tools for PowerShell Cmdlet Reference.

Get-CDAplicationBatch

The following code example shows how to use Get-CDAplicationBatch.

Example 1: This example gets information about the specified applications.

```
Get-CDAplicationBatch -ApplicationName CodeDeployDemoApplication, CodePipelineDemoApplication
```

Output:

```
ApplicationId                           ApplicationName          CreateTime
-------------                           ---------------          ----------
--------------                           ---------------          --------
e07fb938-091e-4f2f-8963-4d3e8EX         CodeDeployDemoApplication 7/20/2015 9:49:48 PM    False
1ecfd602-62f1-4038-8f0d-06688EX          CodePipelineDemoApplication 8/13/2015 5:53:26 PM    False
```

For API details, see [BatchGetApplications](https://aws.amazon.com) in AWS Tools for PowerShell Cmdlet Reference.
Get-CDAplicationList

The following code example shows how to use Get-CDAplicationList.

Tools for PowerShell

Example 1: This example gets a list of available applications.

Get-CDAplicationList

Output:

CodeDeployDemoApplication
CodePipelineDemoApplication

• For API details, see ListApplications in AWS Tools for PowerShell Cmdlet Reference.

Get-CDAplicationRevision

The following code example shows how to use Get-CDAplicationRevision.

Tools for PowerShell

Example 1: This example gets information about the specified application revision.

```powershell
$revision = Get-CDAplicationRevision -ApplicationName CodeDeployDemoApplication -S3Location_Bucket MyBucket -Revision_RevisionType S3 -S3Location_Key 5xd27EX.zip -S3Location_BundleType zip -S3Location_ETag 4565c1ac97187f190c1a90265EXAMPLE
Write-Output ("Description = " + $revision.RevisionInfo.Description + ", RegisterTime = " + $revision.RevisionInfo.RegisterTime)
```

Output:

Description = Application revision registered by Deployment ID: d-CX9CHN3EX,
RegisterTime = 07/20/2015 23:46:42

• For API details, see GetApplicationRevision in AWS Tools for PowerShell Cmdlet Reference.

Get-CDAplicationRevisionList

The following code example shows how to use Get-CDAplicationRevisionList.
Example 1: This example gets information about available revisions for the specified application.

```powershell
ForEach ($revision in (Get-CDAplicationRevisionList -ApplicationName CodeDeployDemoApplication -Deployed Ignore)) {
    >> If ($revision.RevisionType -Eq "S3") {
    >>     Write-Output ("Type = S3, Bucket = " + $revision.S3Location.Bucket + ", BundleType = " + $revision.S3Location.BundleType + ", ETag = " + $revision.S3Location.ETag + ", Key = " + $revision.S3Location.Key)
    >> }
    >> If ($revision.RevisionType -Eq "GitHub") {
    >>     Write-Output ("Type = GitHub, CommitId = " + $revision.GitHubLocation.CommitId + ", Repository = " + $revision.GitHubLocation.Repository)
    >> }
}>
```

Output:

```
Type = S3, Bucket = MyBucket, BundleType = zip, ETag = 4565c1ac97187f190c1a90265EXAMPLE, Key = 5xd27EX.zip
Type = GitHub, CommitId = f48933c3...76405362, Repository = MyGitHubUser/CodeDeployDemoRepo
```

- For API details, see [ListApplicationRevisions](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-CDDeployment

The following code example shows how to use Get-CDDeployment.

Tools for PowerShell

Example 1: This example gets summary information about the specified deployment.

```
Get-CDDeployment -DeploymentId d-QZMRGSTEX
```

Output:

```
ApplicationName : CodeDeployDemoApplication
```
Example 2: This example gets information about the status of instances that are participating in the specified deployment.

```powershell
(Get-CDDeployment -DeploymentId d-QZMRGSTEX).DeploymentOverview
```

Output:

```
Failed : 0
InProgress : 0
Pending : 0
Skipped : 0
Succeeded : 3
```

Example 3: This example gets information about the application revision for the specified deployment.

```powershell
(Get-CDDeployment -DeploymentId d-QZMRGSTEX).Revision.S3Location
```

Output:

```
Bucket : MyBucket
BundleType : zip
ETag : cfbb81b304ee5e27efc21adaed3EXAMPLE
Key : clzfqEX
Version :
```

- For API details, see [GetDeployment](https://aws.amazon.com) in *AWS Tools for PowerShell Cmdlet Reference.*
Get-CDDeploymentBatch

The following code example shows how to use Get-CDDeploymentBatch.

Tools for PowerShell

Example 1: This example gets information about the specified deployments.

```powershell
Get-CDDeploymentBatch -DeploymentId d-QZMRGSTE, d-RR0T5KTEX
```

Output:

```
<table>
<thead>
<tr>
<th>ApplicationName</th>
<th>CodeDeployDemoApplication</th>
</tr>
</thead>
<tbody>
<tr>
<td>CompleteTime</td>
<td>7/23/2015 11:26:04 PM</td>
</tr>
<tr>
<td>CreateTime</td>
<td>7/23/2015 11:24:43 PM</td>
</tr>
<tr>
<td>Creator</td>
<td>user</td>
</tr>
<tr>
<td>DeploymentConfigName</td>
<td>CodeDeployDefault.OneAtATime</td>
</tr>
<tr>
<td>DeploymentGroupName</td>
<td>CodeDeployDemoFleet</td>
</tr>
<tr>
<td>DeploymentId</td>
<td>d-QZMRGSTE</td>
</tr>
<tr>
<td>DeploymentOverview</td>
<td>Amazon.CodeDeploy.Model.DeploymentOverview</td>
</tr>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>ErrorInformation</td>
<td></td>
</tr>
<tr>
<td>IgnoreApplicationStopFailures</td>
<td>False</td>
</tr>
<tr>
<td>Revision</td>
<td>Amazon.CodeDeploy.Model.RevisionLocation</td>
</tr>
<tr>
<td>StartTime</td>
<td>1/1/0001 12:00:00 AM</td>
</tr>
<tr>
<td>Status</td>
<td>Succeeded</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ApplicationName</th>
<th>CodePipelineDemoApplication</th>
</tr>
</thead>
<tbody>
<tr>
<td>CompleteTime</td>
<td>7/23/2015 6:07:30 PM</td>
</tr>
<tr>
<td>CreateTime</td>
<td>7/23/2015 6:06:29 PM</td>
</tr>
<tr>
<td>Creator</td>
<td>user</td>
</tr>
<tr>
<td>DeploymentConfigName</td>
<td>CodeDeployDefault.OneAtATime</td>
</tr>
<tr>
<td>DeploymentGroupName</td>
<td>CodePipelineDemoFleet</td>
</tr>
<tr>
<td>DeploymentId</td>
<td>d-RR0T5KTEX</td>
</tr>
<tr>
<td>DeploymentOverview</td>
<td>Amazon.CodeDeploy.Model.DeploymentOverview</td>
</tr>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>ErrorInformation</td>
<td></td>
</tr>
<tr>
<td>IgnoreApplicationStopFailures</td>
<td>False</td>
</tr>
<tr>
<td>Revision</td>
<td>Amazon.CodeDeploy.Model.RevisionLocation</td>
</tr>
<tr>
<td>StartTime</td>
<td>1/1/0001 12:00:00 AM</td>
</tr>
<tr>
<td>Status</td>
<td>Succeeded</td>
</tr>
</tbody>
</table>
```

- For API details, see [BatchGetDeployments](https://aws.amazon.com) in *AWS Tools for PowerShell Cmdlet Reference*.
Get-CDDeploymentConfig

The following code example shows how to use Get-CDDeploymentConfig.

Tools for PowerShell

Example 1: This example gets summary information about the specified deployment configuration.

```
Get-CDDeploymentConfig -DeploymentConfigName ThreeQuartersHealthy
```

Output:

```
<table>
<thead>
<tr>
<th>CreateTime</th>
<th>DeploymentConfigId</th>
<th>DeploymentConfigName</th>
</tr>
</thead>
<tbody>
<tr>
<td>10/3/2014 4:32:30 PM</td>
<td>518a3950-d034-46a1-9d2c-3949EXAMPLE</td>
<td>ThreeQuartersHealthy</td>
</tr>
<tr>
<td></td>
<td>Amazon.CodeDeploy.Model.MinimumHealthyHosts</td>
<td></td>
</tr>
</tbody>
</table>
```

Example 2: This example gets information about the definition of the specified deployment configuration.

```
Write-Output ((Get-CDDeploymentConfig -DeploymentConfigName ThreeQuartersHealthy).MinimumHealthyHosts)
```

Output:

```
<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLEET_PERCENT</td>
<td>75</td>
</tr>
</tbody>
</table>
```


Get-CDDeploymentConfigList

The following code example shows how to use Get-CDDeploymentConfigList.

Tools for PowerShell

Example 1: This example gets a list of available deployment configurations.
Get-CDDeploymentConfigList

Output:

ThreeQuartersHealthy
CodeDeployDefault.OneAtATime
CodeDeployDefault.AllAtOnce
CodeDeployDefault.HalfAtATime

• For API details, see ListDeploymentConfigs in AWS Tools for PowerShell Cmdlet Reference.

Get-CDDeploymentGroup

The following code example shows how to use Get-CDDeploymentGroup.

Tools for PowerShell

Example 1: This example gets information about the specified deployment group.

Get-CDDeploymentGroup -ApplicationName CodeDeployDemoApplication -DeploymentGroupName CodeDeployDemoFleet

Output:

ApplicationName : CodeDeployDemoApplication
AutoScalingGroups : {}
DeploymentConfigName : CodeDeployDefault.OneAtATime
DeploymentGroupId : 7d7c098a-b444-4b27-96ef-22791EXAMPLE
DeploymentGroupName : CodeDeployDemoFleet
Ec2TagFilters : {Name}
OnPremisesInstanceTagFilters : {}
ServiceRoleArn : arn:aws:iam::80398EXAMPLE:role/CodeDeploySampleStack-4ph6EX-CodeDeployTrustRole-09MWP7XTL8EX
TargetRevision : Amazon.CodeDeploy.Model.RevisionLocation

• For API details, see GetDeploymentGroup in AWS Tools for PowerShell Cmdlet Reference.

Get-CDDeploymentGroupList

The following code example shows how to use Get-CDDeploymentGroupList.
Tools for PowerShell

Example 1: This example gets a list of deployment groups for the specified application.

```powershell
Get-CDDeploymentGroupList -ApplicationName CodeDeployDemoApplication
```

Output:

<table>
<thead>
<tr>
<th>ApplicationName</th>
<th>DeploymentGroups</th>
</tr>
</thead>
<tbody>
<tr>
<td>CodeDeployDemoApplication</td>
<td>{CodeDeployDemoFleet, CodeDeployProductionFleet}</td>
</tr>
</tbody>
</table>

- For API details, see [ListDeploymentGroups](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-CDDeploymentInstance

The following code example shows how to use Get-CDDeploymentInstance.

Tools for PowerShell

Example 1: This example gets information about the specified instance for the specified deployment.

```powershell
Get-CDDeploymentInstance -DeploymentId d-QZMRGSTEX -InstanceId i-254e22EX
```

Output:

```
DeploymentId : d-QZMRGSTEX
InstanceId   : arn:aws:ec2:us-east-1:80398EXAMPLE:instance/i-254e22EX
LastUpdatedAt: 7/23/2015 11:25:24 PM
LifecycleEvents: {ApplicationStop, DownloadBundle, BeforeInstall, Install...}
Status        : Succeeded
```

- For API details, see [GetDeploymentInstance](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-CDDeploymentInstanceList

The following code example shows how to use Get-CDDeploymentInstanceList.
Example 1: This example gets a list of instance IDs for the specified deployment.

```
Get-CDDeploymentInstanceList -DeploymentId d-QZMRGSTEX
```

Output:

```
i-254e22EX
i-274e22EX
i-3b4e22EX
```

For API details, see ListDeploymentInstances in AWS Tools for PowerShell Cmdlet Reference.

Get-CDDeploymentList

The following code example shows how to use Get-CDDeploymentList.

Example 1: This example gets a list of deployment IDs for the specified application and deployment group.

```
Get-CDDeploymentList -ApplicationName CodeDeployDemoApplication -DeploymentGroupName CodeDeployDemoFleet
```

Output:

```
d-QZMRGSTEX
d-RR0T5KTEX
```

For API details, see ListDeployments in AWS Tools for PowerShell Cmdlet Reference.

Get-CDOnPremiseInstance

The following code example shows how to use Get-CDOnPremiseInstance.

Example 1: This example gets information about the specified on-premises instance.
Get-CDOnPremiseInstance -InstanceName AssetTag12010298EX

Output:

DeregisterTime : 1/1/0001 12:00:00 AM
IamUserArn      : arn:aws:iam::80398EXAMPLE:user/CodeDeployDemoUser
AssetTag12010298EX_rDH556dxEX
InstanceName    : AssetTag12010298EX
RegisterTime    : 4/3/2015 6:36:24 PM
Tags            : {Name}

For API details, see GetOnPremisesInstance in AWS Tools for PowerShell Cmdlet Reference.

Get-CDOnPremiseInstanceBatch

The following code example shows how to use Get-CDOnPremiseInstanceBatch.

Tools for PowerShell

Example 1: This example gets information about the specified on-premises instances.

Get-CDOnPremiseInstanceBatch -InstanceName AssetTag12010298EX, AssetTag12010298EX-2

Output:

DeregisterTime : 1/1/0001 12:00:00 AM
IamUserArn      : arn:aws:iam::80398EXAMPLE:user/CodeDeployFRWUser
AssetTag12010298EX_2_XmeSz18rEX
InstanceName    : AssetTag12010298EX-2
RegisterTime    : 4/3/2015 6:38:52 PM
Tags            : {Name}

DeregisterTime : 1/1/0001 12:00:00 AM
IamUserArn      : arn:aws:iam::80398EXAMPLE:user/CodeDeployDemoUser
AssetTag12010298EX_rDH556dxEX
InstanceName    : AssetTag12010298EX
RegisterTime    : 4/3/2015 6:36:24 PM
Tags            : {Name}
• For API details, see BatchGetOnPremisesInstances in AWS Tools for PowerShell Cmdlet Reference.

Get-CDOnPremiseInstanceList

The following code example shows how to use Get-CDOnPremiseInstanceList.

Tools for PowerShell

Example 1: This example gets a list of available on-premises instance names.

```powershell
Get-CDOnPremiseInstanceList
```

Output:

```
AssetTag12010298EX
AssetTag12010298EX-2
```

• For API details, see ListOnPremisesInstances in AWS Tools for PowerShell Cmdlet Reference.

New-CDAplication

The following code example shows how to use New-CDAplication.

Tools for PowerShell

Example 1: This example creates a new application with the specified name.

```powershell
New-CDAplication -ApplicationName MyNewApplication
```

Output:

```
f19e4b61-2231-4328-b0fd-e57f5EXAMPLE
```

• For API details, see CreateApplication in AWS Tools for PowerShell Cmdlet Reference.

New-CDDeployment

The following code example shows how to use New-CDDeployment.
Tools for PowerShell

Example 1: This example creates a new deployment for the specified application and deployment group with the specified deployment configuration and application revision.

```powershell
New-CDDeployment -ApplicationName MyNewApplication -S3Location_Bucket MyBucket -S3Location_BundleType zip -DeploymentConfigName CodeDeployDefault.OneAtATime -DeploymentGroupName MyNewDeploymentGroup -IgnoreApplicationStopFailures $True -S3Location_Key aws-codedeploy_linux-master.zip -RevisionType S3
```

Output:

```
d-ZHROG7UEX
```

Example 2: This example shows how to specify groups of EC2 instance tags that an instance must be identified by in order for it to be included in the replacement environment for a blue/green deployment.

```powershell
New-CDDeployment -ApplicationName MyNewApplication -S3Location_Bucket MyBucket -S3Location_BundleType zip -DeploymentConfigName CodeDeployDefault.OneAtATime -DeploymentGroupName MyNewDeploymentGroup -IgnoreApplicationStopFailures $True -S3Location_Key aws-codedeploy_linux-master.zip -RevisionType S3 -Ec2TagSetList @(@{Key="key1";Type="KEY_ONLY"},@{Key="Key2";Type="KEY_AND_VALUE";Value="Value2"}),@({Key="Key3";Type="VALUE_ONLY";Value="Value3"})
```

Output:

```
d-ZHROG7UEX
```

- For API details, see [CreateDeployment](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**New-CDDeploymentConfig**

The following code example shows how to use `New-CDDeploymentConfig`.

Tools for PowerShell

Example 1: This example creates a new deployment configuration with the specified name and behavior.
New-CDDeploymentConfig -DeploymentConfigName AtLeastTwoHealthyHosts -MinimumHealthyHosts_Type HOST_COUNT -MinimumHealthyHosts_Value 2

Output:

0f3e8187-44ef-42da-aeed-b6823EXAMPLE

- For API details, see [CreateDeploymentConfig](#) in *AWS Tools for PowerShell Cmdlet Reference*.

New-CDDeploymentGroup

The following code example shows how to use New-CDDeploymentGroup.

**Tools for PowerShell**

**Example 1:** This example creates a deployment group with the specified name, Auto Scaling group, deployment configuration, tag, and service role, for the specified application.

```powershell
```

Output:

16bbf199-95fd-40fc-a909-0bbcfEXAMPLE

**Example 2:** This example shows how to specify groups of EC2 instance tags that an instance must be identified by in order for it to be included in the replacement environment for a blue/green deployment.

```powershell
New-CDDeploymentGroup -ApplicationName MyNewApplication -AutoScalingGroup CodeDeployDemo-ASG -DeploymentConfigName CodeDeployDefault.OneAtATime -DeploymentGroupName MyNewDeploymentGroup -Ec2TagFilter @$([Key="key1";Type="KEY_ONLY"],@{Key="Key2";Type="KEY_AND_VALUE";Value="Value2"}),@{Key="key3";Type="VALUE_ONLY";Value="Value3"}
```

CodeDeploy
Output:

16bbf199-95fd-40fc-a909-0bbcfEXmple

- For API details, see [CreateDeploymentGroup](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Register-CDAplicationRevision**

The following code example shows how to use `Register-CDAplicationRevision`.

**Tools for PowerShell**

**Example 1:** This example registers an application revision with the specified Amazon S3 location, for the specified application.

```powershell
Register-CDAplicationRevision -ApplicationName MyNewApplication -S3Location_Bucket MyBucket -S3Location_BundleType zip -S3Location_Key aws-codedeploy_linux-master.zip -Revision_RevisionType S3
```

- For API details, see [RegisterApplicationRevision](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Register-CDOnPremiseInstance**

The following code example shows how to use `Register-CDOnPremiseInstance`.

**Tools for PowerShell**

**Example 1:** This example registers an on-premises instance with the specified name and IAM user.

```powershell
Register-CDOnPremiseInstance -IamUserArn arn:aws:iam::80398EXmple:user/CodeDeployDemoUser -InstanceName AssetTag12010298EX
```

- For API details, see [RegisterOnPremisesInstance](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-CDAplication**

The following code example shows how to use `Remove-CDAplication`.
Tools for PowerShell

Example 1: This example deletes the application with the specified name. The command will prompt for confirmation before proceeding. Add the -Force parameter to delete the application without a prompt.

```powershell
Remove-CDAppliance -ApplicationName MyNewApplication
```

- For API details, see [DeleteApplication](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Remove-CDDeploymentConfig

The following code example shows how to use Remove-CDDeploymentConfig.

Tools for PowerShell

Example 1: This example deletes the deployment configuration with the specified name. The command will prompt for confirmation before proceeding. Add the -Force parameter to delete the deployment configuration without a prompt.

```powershell
Remove-CDDeploymentConfig -DeploymentConfigName AtLeastTwoHealthyHosts
```

- For API details, see [DeleteDeploymentConfig](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Remove-CDDeploymentGroup

The following code example shows how to use Remove-CDDeploymentGroup.

Tools for PowerShell

Example 1: This example deletes the deployment group with the specified name for the specified application. The command will prompt for confirmation before proceeding. Add the -Force parameter to delete the deployment group without a prompt.

```powershell
Remove-CDDeploymentGroup -ApplicationName MyNewApplication -DeploymentGroupName MyNewDeploymentGroup
```

- For API details, see [DeleteDeploymentGroup](#) in *AWS Tools for PowerShell Cmdlet Reference*. 

---

*CodeDeploy*
Remove-CDOnPremiseInstanceTag

The following code example shows how to use Remove-CDOnPremiseInstanceTag.

**Tools for PowerShell**

**Example 1:** This example deletes the specified tag for the on-premises instance with the specified name. The command will prompt for confirmation before proceeding. Add the -Force parameter to delete the tag without a prompt.

```
Remove-CDOnPremiseInstanceTag -InstanceName AssetTag12010298EX -Tag @{"Key" = "Name"; "Value" = "CodeDeployDemo-OnPrem"}
```

- For API details, see [RemoveTagsFromOnPremisesInstances](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Stop-CDDeployment

The following code example shows how to use Stop-CDDeployment.

**Tools for PowerShell**

**Example 1:** This example attempts to stop the deployment with the specified deployment ID.

```
Stop-CDDeployment -DeploymentId d-LJQNREYEX
```

**Output:**

<table>
<thead>
<tr>
<th>Status</th>
<th>StatusMessage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pending</td>
<td>Stopping Pending. Stopping to schedule commands in the deployment instances</td>
</tr>
</tbody>
</table>

- For API details, see [StopDeployment](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Unregister-CDOnPremiseInstance

The following code example shows how to use Unregister-CDOnPremiseInstance.
Tools for PowerShell

Example 1: This example deregisters the on-premises instance with the specified name.

```powershell
Unregister-CDOnPremiseInstance -InstanceName AssetTag12010298EX
```

- For API details, see [DeregisterOnPremisesInstance](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Update-CDAplication

The following code example shows how to use Update-CDAplication.

Tools for PowerShell

Example 1: This example changes the name of the specified application.

```powershell
Update-CDAplication -ApplicationName MyNewApplication -NewApplicationName MyNewApplication-2
```

- For API details, see [UpdateApplication](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Update-CDDeploymentGroup

The following code example shows how to use Update-CDDeploymentGroup.

Tools for PowerShell

Example 1: This example changes the name of the specified deployment group for the specified application.

```powershell
```

Example 2: This example shows how to specify groups of EC2 instance tags that an instance must be identified by in order for it to be included in the replacement environment for a blue/green deployment.

```powershell
```
CodePipeline examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with CodePipeline.

*Actions* are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

*Scenarios* are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

**Topics**

- **Actions**

**Actions**

**Confirm-CPJob**

The following code example shows how to use *Confirm-CPJob*.

**Tools for PowerShell**

**Example 1: This example gets the status of the specified job.**

```
Confirm-CPJob -JobId f570dc12-5ef3-44bc-945a-6e133EXAMPLE -Nonce 3
```

**Output:**

```
Value
-----
```
• For API details, see AcknowledgeJob in AWS Tools for PowerShell Cmdlet Reference.

Disable-CPStageTransition

The following code example shows how to use Disable-CPStageTransition.

**Tools for PowerShell**

**Example 1:** This example disables the inbound transition for the specified stage in the specified pipeline.

```
Disable-CPStageTransition -PipelineName CodePipelineDemo -Reason "Disabling temporarily." -StageName Beta -TransitionType Inbound
```

• For API details, see DisableStageTransition in AWS Tools for PowerShell Cmdlet Reference.

Enable-CPStageTransition

The following code example shows how to use Enable-CPStageTransition.

**Tools for PowerShell**

**Example 1:** This example enables the inbound transition for the specified stage in the specified pipeline.

```
Enable-CPStageTransition -PipelineName CodePipelineDemo -StageName Beta -TransitionType Inbound
```

• For API details, see EnableStageTransition in AWS Tools for PowerShell Cmdlet Reference.

Get-CPActionType

The following code example shows how to use Get-CPActionType.

**Tools for PowerShell**

**Example 1:** This example gets information about all available actions for the specified owner.
ForEach ($actionType in (Get-CPActionType -ActionOwnerFilter AWS)) {
    Write-Output ("For Category = " + $actionType.Id.Category + ", Owner = " + $actionType.Id.Owner + ", Provider = " + $actionType.Id.Provider + ", Version = " + $actionType.Id.Version + ":")
    Write-Output ("ActionConfigurationProperties:")
    ForEach ($acp in $actionType.ActionConfigurationProperties) {
        Write-Output ("For " + $acp.Name + ":")
        Write-Output ("Description = " + $acp.Description)
        Write-Output ("Key = " + $acp.Key)
        Write-Output ("Queryable = " + $acp.Queryable)
        Write-Output ("Required = " + $acp.Required)
        Write-Output ("Secret = " + $acp.Secret)
    }
    Write-Output ("InputArtifactDetails:")
    Write-Output ("MaximumCount = " + $actionType.InputArtifactDetails.MaximumCount)
    Write-Output ("MinimumCount = " + $actionType.InputArtifactDetails.MinimumCount)
    Write-Output ("OutputArtifactDetails:")
    Write-Output ("MaximumCount = " + $actionType.OutputArtifactDetails.MaximumCount)
    Write-Output ("MinimumCount = " + $actionType.OutputArtifactDetails.MinimumCount)
    Write-Output ("Settings:")
    Write-Output ("EntityUrlTemplate = " + $actionType.Settings.EntityUrlTemplate)
    Write-Output ("ExecutionUrlTemplate = " + $actionType.Settings.ExecutionUrlTemplate)
}

Output:

For Category = Deploy, Owner = AWS, Provider = ElasticBeanstalk, Version = 1:
ActionConfigurationProperties:
    For ApplicationName:
        Description = The AWS Elastic Beanstalk Application name
        Key = True
        Queryable = False
        Required = True
        Secret = False
    For EnvironmentName:
        Description = The AWS Elastic Beanstalk Environment name
        Key = True
        Queryable = False
For API details, see ListActionTypes in AWS Tools for PowerShell Cmdlet Reference.
Get-CPActionableJobList

The following code example shows how to use Get-CPActionableJobList.

Tools for PowerShell

Example 1: This example gets information about all actionable jobs for the specified action category, owner, provider, version, and query parameters.

```
Get-CPActionableJobList -ActionTypeId_Category Build -ActionTypeId_Owner Custom -ActionTypeId_Provider MyCustomProviderName -ActionTypeId_Version 1 -QueryParam @{"ProjectName" = "MyProjectName"}
```

Output:

<table>
<thead>
<tr>
<th>AccountId</th>
<th>Data</th>
<th>Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>80398EXAMPLE</td>
<td>Amazon.CodePipeline.Model.JobData</td>
<td>0de392f5-712d-4f41-ace3-f57a0EXAMPLE</td>
</tr>
</tbody>
</table>

- For API details, see PollForJobs in AWS Tools for PowerShell Cmdlet Reference.

Get-CPJobDetail

The following code example shows how to use Get-CPJobDetail.

Tools for PowerShell

Example 1: This example gets general information about the specified job.

```
Get-CPJobDetail -JobId f570dc12-5ef3-44bc-945a-6e133EXAMPLE
```

Output:

<table>
<thead>
<tr>
<th>AccountId</th>
<th>Data</th>
<th>Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>80398EXAMPLE</td>
<td>Amazon.CodePipeline.Model.JobData</td>
<td>0de392f5-712d-4f41-ace3-f57a0EXAMPLE</td>
</tr>
</tbody>
</table>
Example 2: This example gets detailed information about the specified job.

```powershell
$jobDetails = Get-CPJobDetail -JobId f570dc12-5ef3-44bc-945a-6e133EXAMPLE
Write-Output ("For Job " + $jobDetails.Id + ":")
Write-Output ("  AccountId = " + $jobDetails.AccountId)
$objData = $jobDetails.Data
Write-Output ("  Configuration:" )
ForEach ($key in $jobData.ActionConfiguration.Keys) {
    $value = $jobData.ActionConfiguration.$key
    Write-Output ("    " + $key + " = " + $value)
}
Write-Output ("  ActionTypeId:" )
Write-Output ("    Category = " + $jobData.ActionTypeId.Category)
Write-Output ("    Owner = " + $jobData.ActionTypeId.Owner)
Write-Output ("    Provider = " + $jobData.ActionTypeId.Provider)
Write-Output ("    Version = " + $jobData.ActionTypeId.Version)
Write-Output ("  ArtifactCredentials:" )
Write-Output ("    AccessKeyId = " + $jobData.ArtifactCredentials.AccessKeyId)
Write-Output ("    SecretAccessKey = " +
    $jobData.ArtifactCredentials.SecretAccessKey)
Write-Output ("    SessionToken = " + $jobData.ArtifactCredentials.SessionToken)
Write-Output ("  InputArtifacts:" )
ForEach ($ia in $jobData.InputArtifacts) {
    Write-Output ("    " + $ia.Name)
}
Write-Output ("  OutputArtifacts:" )
ForEach ($oa in $jobData.OutputArtifacts) {
    Write-Output ("    " + $oa.Name)
}
Write-Output ("  PipelineContext:" )
$context = $jobData.PipelineContext
Write-Output ("    Name = " + $context.Action.Name)
Write-Output ("    PipelineName = " + $context.PipelineName)
Write-Output ("    Stage = " + $context.Stage.Name)
```

Output:

For Job f570dc12-5ef3-44bc-945a-6e133EXAMPLE:

  AccountId = 80398EXAMPLE

  Configuration:

  ActionTypeId:
    Category = Build
    Owner = Custom
AWS Tools for PowerShell

Provider = MyCustomProviderName
Version = 1
ArtifactCredentials:
  AccessKeyId = ASIAIEI3...IXI6YREX
  SecretAccessKey = cqAFDhEi...RdQyfa2u
  SessionToken = AQoDYXdz...5u+lsAU=
InputArtifacts:
  MyApp
OutputArtifacts:
  MyAppBuild
PipelineContext:
  Name = Build
  PipelineName = CodePipelineDemo
  Stage = Build

• For API details, see GetJobDetails in AWS Tools for PowerShell Cmdlet Reference.

Get-CPPipeline

The following code example shows how to use Get-CPPipeline.

Tools for PowerShell

Example 1: This example gets general information about the specified pipeline.

Get-CPPipeline -Name CodePipelineDemo -Version 1

Output:

ArtifactStore : Amazon.CodePipeline.Model.ArtifactStore
Name : CodePipelineDemo
RoleArn : arn:aws:iam::80398EXAMPLE:role/CodePipelineServiceRole
Stages : {Source, Build, Beta, TestStage}
Version : 1

Example 2: This example gets detailed information about the specified pipeline.

$pipeline = Get-CPPipeline -Name CodePipelineDemo
Write-Output ("Name = " + $pipeline.Name)
Write-Output ("RoleArn = " + $pipeline.RoleArn)
Write-Output ("Version = " + $pipeline.Version)
Output:

Name = CodePipelineDemo
RoleArn = arn:aws:iam::80398EXAMPLE:role/CodePipelineServiceRole
Version = 3
ArtifactStore:
  Location = MyBucketName
  Type = S3
Stages:
  Name = Source
  Actions:
    Name = Source
    Category = Source
    Owner = ThirdParty
Provider = GitHub
Version = 1
Configuration:
  Branch = master
  OAuthToken = ****
  Owner = my-user-name
  Repo = MyRepoName
InputArtifacts:
  MyApp
RunOrder = 1
Name = Build
Actions:
  Name = Build
  Category = Build
  Owner = Custom
  Provider = MyCustomProviderName
  Version = 1
  Configuration:
    ProjectName = MyProjectName
InputArtifacts:
  MyApp
  MyAppBuild
RunOrder = 1
Name = Beta
Actions:
  Name = CodePipelineDemoFleet
  Category = Deploy
  Owner = AWS
  Provider = CodeDeploy
  Version = 1
  Configuration:
    ApplicationName = CodePipelineDemoApplication
    DeploymentGroupName = CodePipelineDemoFleet
InputArtifacts:
  MyAppBuild
RunOrder = 1
Name = TestStage
Actions:
  Name = MyJenkinsTestAction
  Category = Test
  Owner = Custom
  Provider = MyCustomTestProvider
  Version = 1
  Configuration:
ProjectName = MyJenkinsProjectName
InputArtifacts:
  MyAppBuild
  RunOrder = 1

• For API details, see GetPipeline in AWS Tools for PowerShell Cmdlet Reference.

Get-CPPipelineList

The following code example shows how to use Get-CPPipelineList.

Tools for PowerShell

Example 1: This example gets a list of available pipelines.

Get-CPPipelineList

Output:

<table>
<thead>
<tr>
<th>Created</th>
<th>Name</th>
<th>Updated</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>8/13/2015 10:17:54 PM</td>
<td>CodePipelineDemo</td>
<td>8/13/2015 10:17:54 PM</td>
<td>3</td>
</tr>
</tbody>
</table>

• For API details, see ListPipelines in AWS Tools for PowerShell Cmdlet Reference.

Get-CPPipelineState

The following code example shows how to use Get-CPPipelineState.

Tools for PowerShell

Example 1: This example gets general information about the stages for the specified pipeline.

Get-CPPipelineState -Name CodePipelineDemo

Output:

<table>
<thead>
<tr>
<th>Created</th>
</tr>
</thead>
<tbody>
<tr>
<td>8/13/2015 10:17:54 PM</td>
</tr>
</tbody>
</table>
Example 2: This example gets detailed information about the state of the specified pipeline.

```powershell
ForEach ($stageState in (Get-CPPipelineState -Name $arg).StageStates) {
    Write-Output ("For " + $stageState.StageName + ":")
    Write-Output ("  InboundTransitionState:")
    Write-Output ("    DisabledReason = " + $stageState.InboundTransitionState.DisabledReason)
    Write-Output ("    Enabled = " + $stageState.InboundTransitionState.Enabled)
    Write-Output ("    LastChangedAt = " + $stageState.InboundTransitionState>LastChangedAt)
    Write-Output ("    LastChangedBy = " + $stageState.InboundTransitionState.LastChangedBy)
    Write-Output ("  ActionStates:")
    ForEach ($actionState in $stageState.ActionStates) {
        Write-Output ("    For " + $actionState.ActionName + ":")
        Write-Output ("      CurrentRevision:")
        Write-Output ("        Created = " + $actionState.CurrentRevision.Created)
        Write-Output ("        RevisionChangeId = " + $actionState.CurrentRevision.RevisionChangeId)
        Write-Output ("        RevisionId = " + $actionState.CurrentRevision.RevisionId)
        Write-Output ("        EntityUrl = " + $actionState.EntityUrl)
        Write-Output ("      LatestExecution:")
        Write-Output ("        ErrorDetails:")
        Write-Output ("        Message = " + $actionState.LatestExecution.ErrorDetails.Message)
        Write-Output ("        ExternalExecutionId = " + $actionState.LatestExecution.ExternalExecutionId)
        Write-Output ("        ExternalExecutionUrl = " + $actionState.LatestExecution.ExternalExecutionUrl)
        Write-Output ("        LastStatusChange = " + $actionState.LatestExecution.LastStatusChange)
        Write-Output ("        PercentComplete = " + $actionState.LatestExecution.PercentComplete)
        Write-Output ("        Status = " + $actionState.LatestExecution.Status)
        Write-Output ("        Summary = " + $actionState.LatestExecution.Summary)
        Write-Output ("        RevisionUrl = " + $actionState.RevisionUrl)
```
Output:

For Source:
   InboundTransitionState:
      DisabledReason =
      Enabled =
      LastChangedAt =
      LastChangedBy =
   ActionStates:
      For Source:
         CurrentRevision:
            Created =
            RevisionChangeId =
            RevisionId =
            EntityUrl = https://github.com/my-user-name/MyRepoName/tree/master
   LatestExecution:
      ErrorDetails:
         Code =
         Message =
         ExternalExecutionId =
         ExternalExecutionUrl =
         LastStatusChange = 07/20/2015 23:28:45
         PercentComplete = 0
         Status = Succeeded
         Summary =
         RevisionUrl =

For Build:
   InboundTransitionState:
      DisabledReason =
      Enabled = True
      LastChangedAt = 01/01/0001 00:00:00
      LastChangedBy =
   ActionStates:
      For Build:
         CurrentRevision:
            Created =
            RevisionChangeId =
            RevisionId =
            EntityUrl = http://54.174.131.1EX/job/MyJenkinsDemo
   LatestExecution:
ErrorDetails:
  Code = TimeoutError
  Message = The action failed because a job worker exceeded its time limit.

If this is a custom action, make sure that the job worker is configured correctly.
  ExternalExecutionId =
  ExternalExecutionUrl =
  LastStatusChange = 07/21/2015 00:29:29
  PercentComplete = 0
  Status = Failed
  Summary =
  RevisionUrl =

For Beta:
  InboundTransitionState:
    DisabledReason =
    Enabled = True
    LastChangedAt = 01/01/0001 00:00:00
    LastChangedBy =
  ActionStates:
    For CodePipelineDemoFleet:
      CurrentRevision:
        Created =
        RevisionChangeId =
        RevisionId =
        EntityUrl = https://console.aws.amazon.com/codedeploy/home#/applications/
          CodePipelineDemoApplication/deployment-groups/CodePipelineDemoFleet
      LatestExecution:
        ErrorDetails:
          Code =
          Message =
          ExternalExecutionId = d-D5LTCZXEX
          ExternalExecutionUrl = https://console.aws.amazon.com/codedeploy/home#/deployments/d-D5LTCZXEX
          LastStatusChange = 07/08/2015 22:07:42
          PercentComplete = 0
          Status = Succeeded
          Summary = Deployment Succeeded
          RevisionUrl =
    For TestStage:
      InboundTransitionState:
        DisabledReason =
        Enabled = True
        LastChangedAt = 01/01/0001 00:00:00
        LastChangedBy =
      ActionStates:
For MyJenkinsTestAction25:
CurrentRevision:
Created =
RevisionChangeId =
RevisionId =
EntityUrl = http://54.174.131.1EX/job/MyJenkinsDemo
LatestExecution:
ErrorDetails:
  Code =
  Message =
  ExternalExecutionId = 5
  ExternalExecutionUrl = http://54.174.131.1EX/job/MyJenkinsDemo/5
  LastStatusChange = 07/08/2015 22:09:03
  PercentComplete = 0
  Status = Succeeded
  Summary = Finished
RevisionUrl =

- For API details, see GetPipelineState in AWS Tools for PowerShell Cmdlet Reference.

New-CPCustomActionType

The following code example shows how to use New-CPCustomActionType.

Tools for PowerShell

Example 1: This example creates a new custom action with the specified properties.

```
New-CPCustomActionType -Category Build -ConfigurationProperty @{"Description" = "The name of the build project must be provided when this action is added to the pipeline."; "Key" = $True; "Name" = "ProjectName"; "Queryable" = $False; "Required" = $True; "Secret" = $False; "Type" = "String"} -Settings_EntityUrlTemplate "https://my-build-instance/job/{Config:ProjectName}/" -Settings_ExecutionUrlTemplate "https://my-build-instance/job/mybuildjob/lastSuccessfulBuild{ExternalExecutionId}/" -InputArtifactDetails_MaximumCount 1 -OutputArtifactDetails_MaximumCount 1 -InputArtifactDetails_MinimumCount 0 -OutputArtifactDetails_MinimumCount 0 -Provider "MyBuildProviderName" -Version 1
```

Output:

```
ActionConfigurationProperties : {ProjectName}
Id : Amazon.CodePipeline.Model.ActionTypeId
```
AWS Tools for PowerShell

User Guide

<table>
<thead>
<tr>
<th>InputArtifactDetails</th>
<th>: Amazon.CodePipeline.Model.ArtifactDetails</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutputArtifactDetails</td>
<td>: Amazon.CodePipeline.Model.ArtifactDetails</td>
</tr>
<tr>
<td>Settings</td>
<td>: Amazon.CodePipeline.Model.ActionTypeSettings</td>
</tr>
</tbody>
</table>

- For API details, see [CreateCustomActionType](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### New-CPPipeline

The following code example shows how to use `New-CPPipeline`.

#### Tools for PowerShell

**Example 1:** This example creates a new pipeline with the specified settings.

```powershell
$pipeline = New-Object Amazon.CodePipeline.Model.PipelineDeclaration
$sourceStageAction = New-Object Amazon.CodePipeline.Model.ActionDeclaration
$deployStageAction = New-Object Amazon.CodePipeline.Model.ActionDeclaration

$sourceStageActionOutputArtifact = New-Object Amazon.CodePipeline.Model.OutputArtifact
$sourceStageActionOutputArtifact.Name = "MyApp"

$sourceStageAction.ActionTypeId = @{"Category" = "Source"; "Owner" = "AWS"; "Provider" = "S3"; "Version" = 1}
$sourceStageAction.Configuration.Add("S3Bucket", "MyBucketName")
$sourceStageAction.Configuration.Add("S3ObjectKey", "my-object-key-name.zip")
$sourceStageAction.OutputArtifacts.Add($sourceStageActionOutputArtifact)
$sourceStageAction.Name = "Source"

$deployStageActionInputArtifact = New-Object Amazon.CodePipeline.Model.InputArtifact
$deployStageActionInputArtifact.Name = "MyApp"

$deployStageAction.ActionTypeId = @{"Category" = "Deploy"; "Owner" = "AWS"; "Provider" = "CodeDeploy"; "Version" = 1}
$deployStageAction.Configuration.Add("ApplicationName", "CodePipelineDemoApplication")
$deployStageAction.Configuration.Add("DeploymentGroupName", "CodePipelineDemoFleet")
$deployStageAction.InputArtifacts.Add($deployStageActionInputArtifact)
$deployStageAction.Name = "CodePipelineDemoFleet"

$sourceStage = New-Object Amazon.CodePipeline.Model.StageDeclaration
$deployStage = New-Object Amazon.CodePipeline.Model.StageDeclaration
```
$sourceStage.Name = "Source"
$deployStage.Name = "Beta"

$sourceStage.Actions.Add($sourceStageAction)
$deployStage.Actions.Add($deployStageAction)

$pipeline.ArtifactStore = @{"Location" = "MyBucketName"; "Type" = "S3"}
$pipeline.Name = "CodePipelineDemo"
$pipeline.RoleArn = "arn:aws:iam::80398EXAMPLE:role/CodePipelineServiceRole"
$pipeline.Stages.Add($sourceStage)
$pipeline.Stages.Add($deployStage)
$pipeline.Version = 1

New-CPPipeline -Pipeline $pipeline

Output:

ArtifactStore : Amazon.CodePipeline.Model.ArtifactStore
Name          : CodePipelineDemo
RoleArn       : arn:aws:iam::80398EXAMPLE:role/CodePipelineServiceRole
Stages        : [Source, Beta]
Version       : 1

- For API details, see CreatePipeline in AWS Tools for PowerShell Cmdlet Reference.

Remove-CPCustomActionType

The following code example shows how to use Remove-CPCustomActionType.

Tools for PowerShell

Example 1: This example deletes the specified custom action. The command will prompt for confirmation before proceeding. Add the -Force parameter to delete the custom action without a prompt.

Remove-CPCustomActionType -Category Build -Provider MyBuildProviderName -Version 1

- For API details, see DeleteCustomActionType in AWS Tools for PowerShell Cmdlet Reference.
**Remove-CPPipeline**

The following code example shows how to use Remove-CPPipeline.

**Tools for PowerShell**

*Example 1: This example deletes the specified pipeline. The command will prompt for confirmation before proceeding. Add the -Force parameter to delete the pipeline without a prompt.*

```powershell
Remove-CPPipeline -Name CodePipelineDemo
```

- For API details, see [DeletePipeline](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Start-CPPipelineExecution**

The following code example shows how to use Start-CPPipelineExecution.

**Tools for PowerShell**

*Example 1: This example starts running the specified pipeline.*

```powershell
Start-CPPipelineExecution -Name CodePipelineDemo
```

- For API details, see [StartPipelineExecution](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Update-CPPipeline**

The following code example shows how to use Update-CPPipeline.

**Tools for PowerShell**

*Example 1: This example updates the specified existing pipeline with the specified settings.*

```powershell
$pipeline = New-Object Amazon.CodePipeline.Model.PipelineDeclaration
$sourceStageAction = New-Object Amazon.CodePipeline.Model.ActionDeclaration
$deployStageAction = New-Object Amazon.CodePipeline.Model.ActionDeclaration
$sourceStageActionOutputArtifact = New-Object Amazon.CodePipeline.Model.OutputArtifact
```
$sourceStageActionOutputArtifact.Name = "MyApp"

$sourceStageAction.ActionTypeId = @{"Category" = "Source"; "Owner" = "AWS";
"Provider" = "S3"; "Version" = 1}
$sourceStageAction.Configuration.Add("S3Bucket", "MyBucketName")
$sourceStageAction.Configuration.Add("S3ObjectKey", "my-object-key-name.zip")
$sourceStageAction.OutputArtifacts.Add($sourceStageActionOutputArtifact)
$sourceStageAction.Name = "Source"

$deployStageActionInputArtifact = New-Object Amazon.CodePipeline.Model.InputArtifact
$deployStageActionInputArtifact.Name = "MyApp"

$deployStageAction.ActionTypeId = @{"Category" = "Deploy"; "Owner" = "AWS";
"Provider" = "CodeDeploy"; "Version" = 1}
$deployStageAction.Configuration.Add("ApplicationName",
"CodePipelineDemoApplication")
$deployStageAction.Configuration.Add("DeploymentGroupName", "CodePipelineDemoFleet")
$deployStageAction.InputArtifacts.Add($deployStageActionInputArtifact)
$deployStageAction.Name = "CodePipelineDemoFleet"

$sourceStage = New-Object Amazon.CodePipeline.Model.StageDeclaration
$deployStage = New-Object Amazon.CodePipeline.Model.StageDeclaration

$sourceStage.Name = "MyInputFiles"
$deployStage.Name = "MyTestDeployment"

$sourceStage.Actions.Add($sourceStageAction)
$deployStage.Actions.Add($deployStageAction)

$pipeline.ArtifactStore = @{"Location" = "MyBucketName"; "Type" = "S3"}
$pipeline.Name = "CodePipelineDemo"
$pipeline.RoleArn = "arn:aws:iam::80398EXAMPLE:role/CodePipelineServiceRole"
$pipeline.Stages.Add($sourceStage)
$pipeline.Stages.Add($deployStage)
$pipeline.Version = 1

Update-CPPipeline -Pipeline $pipeline

**Output:**

<table>
<thead>
<tr>
<th>ArtifactStore</th>
<th>Amazon.CodePipeline.Model.ArtifactStore</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>CodePipelineDemo</td>
</tr>
<tr>
<td>RoleArn</td>
<td>arn:aws:iam::80398EXAMPLE:role/CodePipelineServiceRole</td>
</tr>
</tbody>
</table>
Amazon Cognito Identity examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with Amazon Cognito Identity.

**Actions** are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

**Scenarios** are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

- **Actions**

**Actions**

Get-CGIIdentityPool

The following code example shows how to use `Get-CGIIdentityPool`.

**Tools for PowerShell**

**Example 1: Retrieves information about a specific Identity Pool by its id.**

```
Get-CGIIdentityPool -IdentityPoolId us-east-1:0de2af35-2988-4d0b-b22d-EXAMPLEGUID1
```

**Output:**

```
LoggedAt                       : 8/12/2015 4:29:40 PM
AllowUnauthenticatedIdentities : True
DeveloperProviderName          :
```
IdentityPoolId                  : us-east-1:0de2af35-2988-4d0b-b22d-EXAMPLEGUID1
IdentityPoolName               : CommonTests1
OpenIdConnectProviderARNs      : {}
SupportedLoginProviders        : {}
ResponseMetadata               : Amazon.Runtime.ResponseMetadata
ContentLength                  : 142
HttpStatusCode                 : OK

- For API details, see [DescribeIdentityPool](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-CGIIIdentityPoolList

The following code example shows how to use Get-CGIIIdentityPoolList.

**Tools for PowerShell**

**Example 1: Retrieves a list of existing Identity Pools.**

```
Get-CGIIIdentityPoolList
```

**Output:**

<table>
<thead>
<tr>
<th>IdentityPoolId</th>
<th>IdentityPoolName</th>
</tr>
</thead>
<tbody>
<tr>
<td>us-east-1:0de2af35-2988-4d0b-b22d-EXAMPLEGUID1</td>
<td>CommonTests1</td>
</tr>
<tr>
<td>us-east-1:118d242d-204e-4b88-b803-EXAMPLEGUID2</td>
<td>Tests2</td>
</tr>
<tr>
<td>us-east-1:15d49393-ab16-431a-b26e-EXAMPLEGUID3</td>
<td>CommonTests13</td>
</tr>
</tbody>
</table>

- For API details, see [ListIdentityPools](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-CGIIIdentityPoolRole

The following code example shows how to use Get-CGIIIdentityPoolRole.

**Tools for PowerShell**

**Example 1: Gets the information about roles for a specific Identity Pool.**

```
Get-CGIIIdentityPoolRole -IdentityPoolId us-east-1:0de2af35-2988-4d0b-b22d-EXAMPLEGUID1
```
Output:

```
LoggedAt : 8/12/2015 4:33:51 PM
IdentityPoolId : us-east-1:0de2af35-2988-4d0b-b22d-EXAMPLEGUID1
Roles : [{unauthenticated, arn:aws:iam::123456789012:role/
CommonTests1Role}]
ResponseMetadata : Amazon.Runtime.ResponseMetadata
ContentLength : 165
HttpStatusCode : OK
```

- For API details, see [GetIdentityPoolRoles](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**New-CGIIdentityPool**

The following code example shows how to use New-CGIIdentityPool.

**Tools for PowerShell**

**Example 1: Creates a new Identity Pool which allows unauthenticated identities.**

```
New-CGIIdentityPool -AllowUnauthenticatedIdentities $true -IdentityPoolName CommonTests13
```

Output:

```
LoggedAt : 8/12/2015 4:56:07 PM
AllowUnauthenticatedIdentities : True
DeveloperProviderName :
IdentityPoolId : us-east-1:15d49393-ab16-431a-b26e-EXAMPLEGUID3
IdentityPoolName : CommonTests13
OpenIdConnectProviderARNs : {}
SupportedLoginProviders : {}
ResponseMetadata : Amazon.Runtime.ResponseMetadata
ContentLength : 136
HttpStatusCode : OK
```

- For API details, see [CreateIdentityPool](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-CGIIdentityPool**

The following code example shows how to use Remove-CGIIdentityPool.
Tools for PowerShell

Example 1: Deletes a specific Identity Pool.

```powershell
Remove-CGIIdentityPool -IdentityPoolId us-east-1:0de2af35-2988-4d0b-b22d-EXAMPLEGUID1
```

• For API details, see [DeleteIdentityPool](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Set-CGIIdentityPoolRole

The following code example shows how to use Set-CGIIdentityPoolRole.

Tools for PowerShell

Example 1: Configures the specific Identity Pool to have an unauthenticated IAM role.

```powershell
Set-CGIIdentityPoolRole -IdentityPoolId us-east-1:0de2af35-2988-4d0b-b22d-EXAMPLEGUID1 -Role @{ "unauthenticated" = "arn:aws:iam::123456789012:role/CommonTests1Role" }
```

• For API details, see [SetIdentityPoolRoles](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Update-CGIIdentityPool

The following code example shows how to use Update-CGIIdentityPool.

Tools for PowerShell

Example 1: Updates some of the Identity Pool properties, in this case the name of the Identity Pool.

```powershell
Update-CGIIdentityPool -IdentityPoolId us-east-1:0de2af35-2988-4d0b-b22d-EXAMPLEGUID1 -IdentityPoolName NewPoolName
```

Output:

```
LoggedAt                       : 8/12/2015 4:53:33 PM
AllowUnauthenticatedIdentities : False
DeveloperProviderName         :
IdentityPoolId                 : us-east-1:0de2af35-2988-4d0b-b22d-EXAMPLEGUID1
```
AWS Config examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with AWS Config.

**Actions** are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

**Scenarios** are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

**Topics**
- **Actions**

**Actions**

**Add-CFGResourceTag**

The following code example shows how to use Add-CFGResourceTag.

**Tools for PowerShell**

**Example 1: This example associates specified tag to the resource ARN, which is config-rule/config-rule-16iyn0 in this case.**

```
Add-CFGResourceTag -ResourceArn arn:aws:config:eu-west-1:123456789012:config-rule/config-rule-16iyn0 -Tag @{Key="Release";Value="Beta"}
```
For API details, see TagResource in AWS Tools for PowerShell Cmdlet Reference.

Get-CFGAggregateComplianceByConfigRuleList

The following code example shows how to use Get-CFGAggregateComplianceByConfigRuleList.

Tools for PowerShell

Example 1: This example fetches the details from ConfigurationAggregator 'kaju' filtering for the given config rule and expands/returns the 'Compliance' of the rule.

Get-CFGAggregateComplianceByConfigRuleList -ConfigurationAggregatorName kaju -Filters_ConfigRuleName ALB_HTTP_TOHTTPS_REDIRECTCHECK | Select-Object -ExpandProperty Compliance

Output:

<table>
<thead>
<tr>
<th>ComplianceContributorCount</th>
<th>ComplianceType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon.ConfigService.Model.ComplianceContributorCount</td>
<td>NON_COMPLIANT</td>
</tr>
</tbody>
</table>

Example 2: This example fetches details from the given ConfigurationAggregator, filters it for the given account for all regions covered in the aggregator and further returns the compliance for all the rules.

Get-CFGAggregateComplianceByConfigRuleList -ConfigurationAggregatorName kaju -Filters_AccountId 123456789012 | Select-Object ConfigRuleName, @{N="Compliance";E={$_.Compliance.ComplianceType}}

Output:

<table>
<thead>
<tr>
<th>ConfigRuleName</th>
<th>Compliance</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALB_HTTP_TOHTTPS_REDIRECTCHECK</td>
<td>NON_COMPLIANT</td>
</tr>
<tr>
<td>ec2-instance-no-public-ip</td>
<td>NON_COMPLIANT</td>
</tr>
<tr>
<td>desired-instance-type</td>
<td>NON_COMPLIANT</td>
</tr>
</tbody>
</table>

For API details, see DescribeAggregateComplianceByConfigRules in AWS Tools for PowerShell Cmdlet Reference.
Get-CFGAggregateComplianceDetailsByConfigRule

The following code example shows how to use Get-CFGAggregateComplianceDetailsByConfigRule.

Tools for PowerShell

Example 1: This example returns the evaluation results selecting the output with resource-id and resource-type for the AWS Config rule 'desired-instance-type' which are in 'COMPLIANT' state for the given account, aggregator, region and config rule

```powershell
Get-CFGAggregateComplianceDetailsByConfigRule -AccountId 123456789012 -AwsRegion eu-west-1 -ComplianceType COMPLIANT -ConfigRuleName desired-instance-type -ConfigurationAggregatorName raju | Select-Object -ExpandProperty EvaluationResultIdentifier | Select-Object -ExpandProperty EvaluationResultQualifier
```

Output:

<table>
<thead>
<tr>
<th>ConfigRuleName</th>
<th>ResourceId</th>
<th>ResourceType</th>
</tr>
</thead>
<tbody>
<tr>
<td>desired-instance-type</td>
<td>i-0f1bf2f34c5678d12</td>
<td>AWS::EC2::Instance</td>
</tr>
<tr>
<td>desired-instance-type</td>
<td>i-0fd12dd3456789123</td>
<td>AWS::EC2::Instance</td>
</tr>
</tbody>
</table>

- For API details, see [GetAggregateComplianceDetailsByConfigRule](#) in AWS Tools for PowerShell Cmdlet Reference.

Get-CFGAggregateConfigRuleComplianceSummary

The following code example shows how to use Get-CFGAggregateConfigRuleComplianceSummary.

Tools for PowerShell

Example 1: This example returns the number of noncompliant rules for the given aggregator.

```powershell
```
Output:

<table>
<thead>
<tr>
<th>CapExceeded</th>
<th>CappedCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>5</td>
</tr>
</tbody>
</table>

- For API details, see [GetAggregateConfigRuleComplianceSummary](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-CFGAggregateDiscoveredResourceCount

The following code example shows how to use `Get-CFGAggregateDiscoveredResourceCount`.

**Tools for PowerShell**

**Example 1:** This example returns the resource count for the given aggregator filtered for region `us-east-1`.

```powershell
Get-CFGAggregateDiscoveredResourceCount -ConfigurationAggregatorName Master -Filters.Region us-east-1
```

**Output:**

```
GroupByKey GroupedResourceCounts NextToken TotalDiscoveredResources
--------- --------------------- --------- ------------------------
{}                              455
```

**Example 2:** This example returns the resource count grouped by `RESOURCE_TYPE` for the filtered region for the given aggregator.

```powershell
Get-CFGAggregateDiscoveredResourceCount -ConfigurationAggregatorName Master -Filters.Region us-east-1 -GroupByKey RESOURCE_TYPE | Select-Object -ExpandProperty GroupedResourceCounts
```

**Output:**

```
GroupName                          ResourceCount
---------                          -------------
AWS::CloudFormation::Stack         12
AWS::CloudFront::Distribution      1
```

AWS Config

---

**Get-CFGAggregateDiscoveredResourceList**

The following code example shows how to use `Get-CFGAggregateDiscoveredResourceList`.

**Tools for PowerShell**

**Example 1:** This example returns the resource identifiers for the given resource type aggregated in 'Ireland' aggregator. For the list of resource types, please check [https://docs.aws.amazon.com/sdkfornet/v3/apidocs/index.html?page=ConfigService/TConfigServiceResourceType.html&tocid=Amazon_ConfigService_ResourceType](https://docs.aws.amazon.com/sdkfornet/v3/apidocs/index.html?page=ConfigService/TConfigServiceResourceType.html&tocid=Amazon_ConfigService_ResourceType).

```powershell
Get-CFGAggregateDiscoveredResourceList -ConfigurationAggregatorName Ireland -ResourceType ([Amazon.ConfigService.ResourceType]::AWSAutoScalingAutoScalingGroup)
```

**Output:**

AWS::CloudTrail::Trail 1
AWS::DynamoDB::Table 1
AWS::EC2::EIP 2
AWS::EC2::FlowLog 2
AWS::EC2::InternetGateway 4
AWS::EC2::NatGateway 2
AWS::EC2::NetworkAcl 4
AWS::EC2::NetworkInterface 12
AWS::EC2::RouteTable 13
AWS::EC2::SecurityGroup 18
AWS::EC2::Subnet 16
AWS::EC2::VPC 4
AWS::EC2::VPCEndpoint 2
AWS::EC2::VPCPeeringConnection 1
AWS::IAM::Group 2
AWS::IAM::Policy 51
AWS::IAM::Role 78
AWS::IAM::User 7
AWS::Lambda::Function 3
AWS::RDS::DBSecurityGroup 1
AWS::S3::Bucket 3
AWS::SSM::AssociationCompliance 107
AWS::SSM::ManagedInstanceInventory 108
Example 2: This example returns the resource type `AwsEC2SecurityGroup` named 'default' for the given aggregator filtered with region us-east-1.

Get-CFGAggregateDiscoveredResourceList -ConfigurationAggregatorName raju - ResourceType ([Amazon.ConfigService.ResourceType]::AWSEC2SecurityGroup) - Filters_Region us-east-1 -Filters_ResourceName default

Output:

<table>
<thead>
<tr>
<th>ResourceId</th>
<th>sg-01234bd5dbfa67c89</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResourceName</td>
<td>default</td>
</tr>
<tr>
<td>ResourceType</td>
<td>AWS::EC2::SecurityGroup</td>
</tr>
<tr>
<td>SourceAccountId</td>
<td>123456789102</td>
</tr>
<tr>
<td>SourceRegion</td>
<td>us-east-1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ResourceId</th>
<th>sg-0123a4ebbf56789be</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResourceName</td>
<td>default</td>
</tr>
<tr>
<td>ResourceType</td>
<td>AWS::EC2::SecurityGroup</td>
</tr>
<tr>
<td>SourceAccountId</td>
<td>123456789102</td>
</tr>
<tr>
<td>SourceRegion</td>
<td>us-east-1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ResourceId</th>
<th>sg-4fc1d234</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResourceName</td>
<td>default</td>
</tr>
<tr>
<td>ResourceType</td>
<td>AWS::EC2::SecurityGroup</td>
</tr>
<tr>
<td>SourceAccountId</td>
<td>123456789102</td>
</tr>
<tr>
<td>SourceRegion</td>
<td>us-east-1</td>
</tr>
</tbody>
</table>

- For API details, see [ListAggregateDiscoveredResources](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

**Get-CFGAggregateResourceConfig**

The following code example shows how to use Get-CFGAggregateResourceConfig.
Tools for PowerShell

Example 1: This example returns the Configuration Item for the given resource aggregated and expands Configuration.

```powershell
(Get-CFGAggregateResourceConfig -ResourceIdentifier_SourceRegion us-east-1 -ResourceIdentifier_SourceAccountId 123456789012 -ResourceIdentifier_ResourceId sg-4fc1d234 -ResourceIdentifier_ResourceType ([Amazon.ConfigService.ResourceType]::AWSEC2SecurityGroup) -ConfigurationAggregatorName raju).Configuration | ConvertFrom-Json
```

Output:

```json
{"description":"default VPC security group","groupName":"default","ipPermissions": [{"ipProtocol":"-1","ipv6Ranges":[]},"prefixListIds":[]},"userIdGroupPairs": [{"groupId":"sg-4fc1d234"},"userId":"123456789012"}],"ipv4Ranges":[]},"prefixListIds":[]},"ipv4Ranges": [{"cidrIp":"54.240.197.224/29"},"description":"office subnet"],}]

```

• For API details, see [GetAggregateResourceconfig-service](https://docs.aws.amazon.com/config/latest/developerguide/getaggregateresourceconfig-service.html) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-CFGAggregateResourceConfigBatch

The following code example shows how to use Get-CFGAggregateResourceConfigBatch.

Tools for PowerShell

Example 1: This example fetches current configuration item for resource (identified) present in the given aggregator.

```powershell
$resIdentifier=[Amazon.ConfigService.Model.AggregateResourceIdentifier]@{
    ResourceId = "i-01e3cb4df567e8aa"
    ResourceName = "arn:aws:ec2:eu-west-1:123456789012:instance/i-01e3cb4df567e8aa"
```
ResourceType = [Amazon.ConfigService.ResourceType]::AWSEC2Instance
SourceAccountId = "123456789012"
SourceRegion = "eu-west-1"

Get-CFGAggregateResourceConfigBatch -ResourceIdentifier $resIdentifier -
ConfigurationAggregatorName raju

Output:

BaseConfigurationItems UnprocessedResourceIdentifiers
---------------------- ------------------------------
{}                     {arn:aws:ec2:eu-west-1:123456789012:instance/
i-012e3cb4df567e8aa}

- For API details, see `BatchGetAggregateResourceconfig-service` in `AWS Tools for PowerShell Cmdlet Reference`.

Get-CFGAggregationAuthorizationList

The following code example shows how to use Get-CFGAggregationAuthorizationList.

**Tools for PowerShell**

**Example 1: This example retrieves authorizations granted to aggregators.**

Get-CFGAggregationAuthorizationList

Output:

AggregationAuthorizationArn
AuthorizedAccountId AuthorizedAwsRegion CreationTime
------------------------------------------
------------------------------------------
arn:aws:config-service:eu-west-1:123456789012:aggregation-
authorization/123456789012/eu-west-1 123456789012 eu-west-1

- For API details, see `DescribeAggregationAuthorizations` in `AWS Tools for PowerShell Cmdlet Reference`.

AWS Config
Get-CFGComplianceByConfigRule

The following code example shows how to use Get-CFGComplianceByConfigRule.

Tools for PowerShell

Example 1: This example retrieves compliances details for the rule ebs-optimized-instance, for which there is no current evaluation results for the rule, hence it returns INSUFFICIENT_DATA

(Get-CFGComplianceByConfigRule -ConfigRuleName ebs-optimized-instance).Compliance

Output:

<table>
<thead>
<tr>
<th>ComplianceContributorCount</th>
<th>ComplianceType</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>INSUFFICIENT_DATA</td>
</tr>
</tbody>
</table>

Example 2: This example returns the number of non-compliant resources for the rule ALB_HTTP_TO_HTTPS_REDIRECTION_CHECK.

(Get-CFGComplianceByConfigRule -ConfigRuleName ALB_HTTP_TO_HTTPS_REDIRECTION_CHECK -ComplianceType NON_COMPLIANT).Compliance.ComplianceContributorCount

Output:

<table>
<thead>
<tr>
<th>CapExceeded</th>
<th>CappedCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>2</td>
</tr>
</tbody>
</table>

• For API details, see [DescribeComplianceByConfigRule](#) in AWS Tools for PowerShell Cmdlet Reference.

Get-CFGComplianceByResource

The following code example shows how to use Get-CFGComplianceByResource.

Tools for PowerShell

Example 1: This example checks the AWS::SSM::ManagedInstanceInventory resource type for 'COMPLIANT' compliance type.
Get-CFGComplianceByResource -ComplianceType COMPLIANT -ResourceType AWS::SSM::ManagedInstanceInventory

Output:

<table>
<thead>
<tr>
<th>Compliance</th>
<th>ResourceId</th>
<th>ResourceType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon.ConfigService.Model.Compliance</td>
<td>i-0123bcf4b567890e3</td>
<td>AWS::SSM::ManagedInstanceInventory</td>
</tr>
<tr>
<td>Amazon.ConfigService.Model.Compliance</td>
<td>i-0a1234f6f5d6b78f7</td>
<td>AWS::SSM::ManagedInstanceInventory</td>
</tr>
</tbody>
</table>

• For API details, see DescribeComplianceByResource in AWS Tools for PowerShell Cmdlet Reference.

Get-CFGComplianceDetailsByConfigRule

The following code example shows how to use Get-CFGComplianceDetailsByConfigRule.

Tools for PowerShell

Example 1: This example obtains the evaluation results for the rule access-keys-rotated and returns the output grouped by compliance-type

Get-CFGComplianceDetailsByConfigRule -ConfigRuleName access-keys-rotated | Group-Object ComplianceType

Output:

```
Count Name                      Group
----- ----                      ----- 
```

Example 2: This example queries compliance details for the rule access-keys-rotated for COMPLIANT resources.
Get-CFGComplianceDetailsByConfigRule -ConfigRuleName access-keys-rotated -ComplianceType COMPLIANT | ForEach-Object {$_ .EvaluationResultIdentifier.EvaluationResultQualifier}

Output:

<table>
<thead>
<tr>
<th>ConfigRuleName</th>
<th>ResourceId</th>
<th>ResourceType</th>
</tr>
</thead>
<tbody>
<tr>
<td>access-keys-rotated</td>
<td>BCAB1CDJ2LITAPVEW3JAH</td>
<td>AWS::IAM::User</td>
</tr>
<tr>
<td>access-keys-rotated</td>
<td>BCAB1CDJ2LITL3EHREM4Q</td>
<td>AWS::IAM::User</td>
</tr>
</tbody>
</table>

- For API details, see [GetComplianceDetailsByConfigRule](#) in **AWS Tools for PowerShell Cmdlet Reference**.

### Get-CFGComplianceDetailsByResource

The following code example shows how to use `Get-CFGComplianceDetailsByResource`.

**Tools for PowerShell**

**Example 1: This example evaulation results for the given resource.**

```powershell
Get-CFGComplianceDetailsByResource -ResourceId ABCD5STJ4EFGHIVEW6JAH -ResourceType 'AWS::IAM::User'
```

Output:

<table>
<thead>
<tr>
<th>Annotation</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ComplianceType</td>
<td>COMPLIANT</td>
</tr>
<tr>
<td>ConfigRuleInvokedTime</td>
<td>8/25/2019 11:34:56 PM</td>
</tr>
<tr>
<td>EvaluationResultIdentifier</td>
<td>Amazon.ConfigService.Model.EvaluationResultIdentifier</td>
</tr>
<tr>
<td>ResultRecordedTime</td>
<td>8/25/2019 11:34:56 PM</td>
</tr>
<tr>
<td>ResultToken</td>
<td></td>
</tr>
</tbody>
</table>

- For API details, see [GetComplianceDetailsByResource](#) in **AWS Tools for PowerShell Cmdlet Reference**.

### Get-CFGComplianceSummaryByConfigRule

The following code example shows how to use `Get-CFGComplianceSummaryByConfigRule`.

---

AWS Config  287
Tools for PowerShell

Example 1: This sample returns the number of Config rules that are non-compliant.

Get-CFGComplianceSummaryByConfigRule -Select ComplianceSummary.NonCompliantResourceCount

Output:

<table>
<thead>
<tr>
<th>CapExceeded</th>
<th>CappedCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>9</td>
</tr>
</tbody>
</table>

• For API details, see GetComplianceSummaryByConfigRule in AWS Tools for PowerShell Cmdlet Reference.

Get-CFGComplianceSummaryByResourceType

The following code example shows how to use Get-CFGComplianceSummaryByResourceType.

Example 1: This sample returns the number of resources that are compliant or noncompliant and converts the output to json.

Get-CFGComplianceSummaryByResourceType -Select ComplianceSummariesByResourceType.ComplianceSummary | ConvertTo-Json

{  
  "ComplianceSummaryTimestamp": "2019-12-14T06:14:49.778Z",  
  "CompliantResourceCount": {  
    "CapExceeded": false,  
    "CappedCount": 2  
  },  
  "NonCompliantResourceCount": {  
    "CapExceeded": true,  
    "CappedCount": 100  
  }
}

• For API details, see GetComplianceSummaryByResourceType in AWS Tools for PowerShell Cmdlet Reference.
Get-CFGConfigRule

The following code example shows how to use Get-CFGConfigRule.

Tools for PowerShell

Example 1: This sample lists config rules for the account, with selected properties.

Get-CFGConfigRule | Select-Object ConfigRuleName, ConfigRuleId, ConfigRuleArn, ConfigRuleState

Output:

<table>
<thead>
<tr>
<th>ConfigRuleName</th>
<th>ConfigRuleId</th>
<th>ConfigRuleArn</th>
<th>ConfigRuleState</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALB_REDIRECTION_CHECK</td>
<td>config-rule-12iyn3</td>
<td>arn:aws:config-service:eu-west-1:123456789012:config-rule/config-rule-12iyn3</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>access-keys-rotated</td>
<td>config-rule-aospfr</td>
<td>arn:aws:config-service:eu-west-1:123456789012:config-rule/config-rule-aospfr</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>autoscaling-group-elb-healthcheck-required</td>
<td>config-rule-cn1f2x</td>
<td>arn:aws:config-service:eu-west-1:123456789012:config-rule/config-rule-cn1f2x</td>
<td>ACTIVE</td>
</tr>
</tbody>
</table>

• For API details, see DescribeConfigRules in AWS Tools for PowerShell Cmdlet Reference.

Get-CFGConfigRuleEvaluationStatus

The following code example shows how to use Get-CFGConfigRuleEvaluationStatus.

Tools for PowerShell

Example 1: This sample returns the status information for the given config rules.

Get-CFGConfigRuleEvaluationStatus -ConfigRuleName root-account-mfa-enabled, vpc-flow-logs-enabled

Output:

<table>
<thead>
<tr>
<th>ConfigRuleArn</th>
<th>ConfigRuleId</th>
</tr>
</thead>
<tbody>
<tr>
<td>config-rule-kvqlwk</td>
<td>config-rule-kvqlwk</td>
</tr>
<tr>
<td>ConfigRuleName</td>
<td>root-account-mfa-enabled</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>FirstActivatedTime</td>
<td>8/27/2019 8:05:17 AM</td>
</tr>
<tr>
<td>FirstEvaluationStarted</td>
<td>True</td>
</tr>
<tr>
<td>LastErrorCode</td>
<td></td>
</tr>
<tr>
<td>LastErrorMessage</td>
<td></td>
</tr>
<tr>
<td>LastFailedEvaluationTime</td>
<td>1/1/0001 12:00:00 AM</td>
</tr>
<tr>
<td>LastFailedInvocationTime</td>
<td>1/1/0001 12:00:00 AM</td>
</tr>
<tr>
<td>LastSuccessfulEvaluationTime</td>
<td>12/13/2019 8:12:03 AM</td>
</tr>
<tr>
<td>LastSuccessfulInvocationTime</td>
<td>12/13/2019 8:12:03 AM</td>
</tr>
</tbody>
</table>

| ConfigRuleArn               | arn:aws:config:eu-west-1:123456789012:config-rule/config-rule-z1s23b |
| ConfigRuleId                | config-rule-z1s23b        |
| ConfigRuleName              | vpc-flow-logs-enabled     |
| FirstActivatedTime          | 8/14/2019 6:23:44 AM      |
| FirstEvaluationStarted      | True                     |
| LastErrorCode               |                          |
| LastErrorMessage            |                          |
| LastFailedEvaluationTime    | 1/1/0001 12:00:00 AM     |
| LastFailedInvocationTime    | 1/1/0001 12:00:00 AM     |
| LastSuccessfulEvaluationTime| 12/13/2019 7:12:01 AM   |
| LastSuccessfulInvocationTime| 12/13/2019 7:12:01 AM   |

- For API details, see [DescribeConfigRuleEvaluationStatus](#) in *AWS Tools for PowerShell Cmdlet Reference*.

---

**Get-CFGConfigurationAggregatorList**

The following code example shows how to use Get-CFGConfigurationAggregatorList.

**Tools for PowerShell**

**Example 1: This sample returns all the aggregators for the region/account.**

```
Get-CFGConfigurationAggregatorList
```

**Output:**

```
AccountAggregationSources :
 {Amazon.ConfigService.Model.AccountAggregationSource}
ConfigurationAggregatorArn : arn:aws:config-service:eu-west-1:123456789012:config-aggregator/config-aggregator-xabca1me
```
For API details, see [DescribeConfigurationAggregators](https://docs.aws.amazon.com/config-service/latest/developerguide/configuration-aggregator-configuration.html) in AWS Tools for PowerShell Cmdlet Reference.

### Get-CFGConfigurationAggregatorSourcesStatus

The following code example shows how to use Get-CFGConfigurationAggregatorSourcesStatus.

**Tools for PowerShell**

**Example 1:** This sample displays requested fields for the sources in the given aggregator.

```powershell
Get-CFGConfigurationAggregatorSourcesStatus -ConfigurationAggregatorName raju | select SourceType, LastUpdateStatus, LastUpdateTime, SourceId
```

**Output:**

<table>
<thead>
<tr>
<th>SourceType</th>
<th>LastUpdateStatus</th>
<th>LastUpdateTime</th>
<th>SourceId</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORGANIZATION</td>
<td>SUCCEEDED</td>
<td>12/31/2019 7:45:06 AM</td>
<td>Organization</td>
</tr>
<tr>
<td>ACCOUNT</td>
<td>SUCCEEDED</td>
<td>12/31/2019 7:09:38 AM</td>
<td>612641234567</td>
</tr>
<tr>
<td>ACCOUNT</td>
<td>SUCCEEDED</td>
<td>12/31/2019 7:12:53 AM</td>
<td>933301234567</td>
</tr>
<tr>
<td>ACCOUNT</td>
<td>SUCCEEDED</td>
<td>12/31/2019 7:18:10 AM</td>
<td>933301234567</td>
</tr>
<tr>
<td>ACCOUNT</td>
<td>SUCCEEDED</td>
<td>12/31/2019 7:25:17 AM</td>
<td>933301234567</td>
</tr>
<tr>
<td>ACCOUNT</td>
<td>SUCCEEDED</td>
<td>12/31/2019 7:25:49 AM</td>
<td>612641234567</td>
</tr>
<tr>
<td>ACCOUNT</td>
<td>SUCCEEDED</td>
<td>12/31/2019 7:26:11 AM</td>
<td>612641234567</td>
</tr>
</tbody>
</table>
For API details, see DescribeConfigurationAggregatorSourcesStatus in AWS Tools for PowerShell Cmdlet Reference.

Get-CFGConfigurationRecorder

The following code example shows how to use Get-CFGConfigurationRecorder.

Tools for PowerShell

Example 1: This example returns the details of configuration recorders.

```powershell
Get-CFGConfigurationRecorder | Format-List
```

Output:

```
Name           : default
RecordingGroup : Amazon.ConfigService.Model.RecordingGroup
RoleARN        : arn:aws:iam::123456789012:role/aws-service-role/config.amazonaws.com/AWSServiceRoleForConfig
```

For API details, see DescribeConfigurationRecorders in AWS Tools for PowerShell Cmdlet Reference.

Get-CFGConfigurationRecorderStatus

The following code example shows how to use Get-CFGConfigurationRecorderStatus.

Tools for PowerShell

Example 1: This sample returns status of the configuration recorders.

```powershell
Get-CFGConfigurationRecorderStatus
```

Output:

```
LastErrorCode       :
LastErrorMessage    :
LastStatus          : Success
LastStatusChangeTime: 12/31/2019 6:14:12 AM
LastStopTime        : 10/11/2019 10:13:46 AM
```
Name : default
Recording : True

- For API details, see [DescribeConfigurationRecorderStatus](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

### Get-CFGConformancePack

The following code example shows how to use `Get-CFGConformancePack`.

#### Tools for PowerShell

**Example 1:** This sample lists all conformance packs.

```powershell
Get-CFGConformancePack
```

**Output:**

```plaintext
ConformancePackId              : conformance-pack-p0acabcde
ConformancePackInputParameters : {}
ConformancePackName            : dono
CreatedBy                      :
DeliveryS3Bucket               : kt-ps-examples
DeliveryS3KeyPrefix            :
LastUpdateRequestedTime        : 12/31/2019 8:45:31 AM
```

- For API details, see [DescribeConformancePacks](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

### Get-CFGDeliveryChannel

The following code example shows how to use `Get-CFGDeliveryChannel`.

#### Tools for PowerShell

**Example 1:** This example retrieves the delivery channel for the region and displays details.

```powershell
Get-CFGDeliveryChannel -Region eu-west-1 | Select-Object Name, S3BucketName, S3KeyPrefix, @{N="DeliveryFrequency";E={$_.ConfigSnapshotDeliveryProperties.DeliveryFrequency}}
```
Output:

<table>
<thead>
<tr>
<th>Name</th>
<th>S3BucketName</th>
<th>S3KeyPrefix</th>
<th>DeliveryFrequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>config-bucket-NA</td>
<td>my</td>
<td>TwentyFour_Hours</td>
</tr>
</tbody>
</table>

- For API details, see [DescribeDeliveryChannels](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-CFGResourceTag**

The following code example shows how to use `Get-CFGResourceTag`.

**Tools for PowerShell**

**Example 1: This example lists associated tags for the given resource**

```
Get-CFGResourceTag -ResourceArn $rules[0].ConfigRuleArn
```

**Output:**

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
<td>-----</td>
</tr>
<tr>
<td>Version</td>
<td>1.3</td>
</tr>
</tbody>
</table>

- For API details, see [ListTagsForResource](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-CFGConformancePack**

The following code example shows how to use `Remove-CFGConformancePack`.

**Tools for PowerShell**

**Example 1: This sample removes the given conformance pack, along with all the rules, remediation actions and evaluation results for the pack.**

```
Remove-CFGConformancePack -ConformancePackName dono
```

**Output:**

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-CFGConformancePack (DeleteConformancePack)" on target "dono".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): Y

- For API details, see [DeleteConformancePack](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

**Write-CFGConformancePack**

The following code example shows how to use `Write-CFGConformancePack`.

**Tools for PowerShell**

**Example 1:** This sample creates conformance pack, fetching template from the given yaml file.

```powershell
Write-CFGConformancePack -ConformancePackName dono -DeliveryS3Bucket kt-ps-examples -TemplateBody (Get-Content C:\windows\temp\template.yaml -Raw)
```

- For API details, see [PutConformancePack](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

**Write-CFGDeliveryChannel**

The following code example shows how to use `Write-CFGDeliveryChannel`.

**Tools for PowerShell**

**Example 1:** This example changes the deliveryFrequency property of an existing delivery channel.

```powershell
Write-CFGDeliveryChannel -ConfigSnapshotDeliveryProperties_DeliveryFrequency TwentyFour_Hours -DeliveryChannelName default -DeliveryChannel_S3BucketName config-bucket-NA -DeliveryChannel_S3KeyPrefix my
```

- For API details, see [PutDeliveryChannel](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

**Device Farm examples using Tools for PowerShell**

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with Device Farm.
Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

- Actions

Actions

New-DFUpload

The following code example shows how to use New-DFUpload.

Tools for PowerShell

Example 1: This example creates an AWS Device Farm upload for an Android app. You can get the project ARN from the output of New-DFProject or Get-DFProjectList. Use the signed URL in the New-DFUpload output to upload a file to Device Farm.

```
New-DFUpload -ContentType "application/octet-stream" -ProjectArn "arn:aws:devicefarm:us-west-2:123456789012:project:EXAMPLEa-7ec1-4741-9c1f-d3e04EXAMPLE" -Name "app.apk" -Type ANDROID_APP
```

- For API details, see [CreateUpload](#) in AWS Tools for PowerShell Cmdlet Reference.

AWS Directory Service examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with AWS Directory Service.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.
**Scenarios** are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

**Topics**

- **Actions**

**Actions**

**Add-DSIpRoute**

The following code example shows how to use `Add-DSIpRoute`.

**Tools for PowerShell**

**Example 1: This command removes the Resource Tag assigned to the specified Directory-id**

```
Add-DSIpRoute -DirectoryId d-123456ijkl -IpRoute @{CidrIp ="203.0.113.5/32"} - UpdateSecurityGroupForDirectoryController $true
```

- For API details, see [AddIpRoutes](https://docs.aws.amazon.com/tools/latest/dwpsg/Networking_with_PowerShell.html) in *AWS Tools for PowerShell Cmdlet Reference*.

**Add-DSResourceTag**

The following code example shows how to use `Add-DSResourceTag`.

**Tools for PowerShell**

**Example 1: This command adds the Resource Tag to the specified Directory-id**

```
Add-DSResourceTag -ResourceId d-123456ijkl -Tag @{Key="myTag"; Value="mytgValue"}
```

- For API details, see [AddTagsToResource](https://docs.aws.amazon.com/tools/latest/dwpsg/Working_with_PowerShell.html) in *AWS Tools for PowerShell Cmdlet Reference*.

**Approve-DSTrust**

The following code example shows how to use `Approve-DSTrust`.
Example 1: This example calls the AWS Directory Service VerifyTrust API operation for specified TrustId.

```
Approve-DSTrust -TrustId t-9067157123
```

- For API details, see [VerifyTrust](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Confirm-DSSharedDirectory

The following code example shows how to use Confirm-DSSharedDirectory.

Tools for PowerShell

Example 1: This example accepts a directory sharing request sent from the directory owner AWS account.

```
Confirm-DSSharedDirectory -SharedDirectoryId d-9067012345
```

Output:

```
CreatedDateTime : 12/30/2019 4:20:27 AM
LastUpdatedDateTime : 12/30/2019 4:21:40 AM
OwnerAccountId : 123456781234
OwnerDirectoryId : d-123456ijkl
SharedAccountId : 123456784321
SharedDirectoryId : d-9067012345
ShareMethod : 
ShareNotes : This is test sharing
ShareStatus : Sharing
```

- For API details, see [AcceptSharedDirectory](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Connect-DSDirectory

The following code example shows how to use Connect-DSDirectory.

Tools for PowerShell

Example 1: This example creates an AD Connector to connect to an on-premises directory.
Connect-DSDirectory -Name contoso.com -ConnectSettings_CustomerUserName Administrator -Password $Password -ConnectSettings_CustomerDnsIp 172.31.36.96 -ShortName CONTOSO -Size Small -ConnectSettings_VpcId vpc-123459da -ConnectSettings_SubnetId subnet-1234ccaa, subnet-5678ffbb

- For API details, see [ConnectDirectory](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Deny-DSSharedDirectory

The following code example shows how to use Deny-DSSharedDirectory.

**Tools for PowerShell**

**Example 1:** This example rejects a directory sharing request that was sent from the directory owner account.

Deny-DSSharedDirectory -SharedDirectoryId d-9067012345

**Output:**

d-9067012345

- For API details, see [RejectSharedDirectory](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Disable-DSDirectoryShare

The following code example shows how to use Disable-DSDirectoryShare.

**Tools for PowerShell**

**Example 1:** This example stops the directory sharing between the directory owner and consumer account.

Disable-DSDirectoryShare -DirectoryId d-123456ijkl -UnshareTarget_Id 123456784321 -UnshareTarget_Type ACCOUNT

**Output:**

d-9067012345
• For API details, see UnshareDirectory in AWS Tools for PowerShell Cmdlet Reference.

**Disable-DSDLAPS**

The following code example shows how to use Disable-DSDLAPS.

**Tools for PowerShell**

*Example 1: This example deactivates LDAP secure calls for the specified directory.*

```
Disable-DSDLAPS -DirectoryId d-123456ijkl -Type Client
```

• For API details, see DisableLDAPS in AWS Tools for PowerShell Cmdlet Reference.

**Disable-DSRadius**

The following code example shows how to use Disable-DSRadius.

**Tools for PowerShell**

*Example 1: This example disables RADIUS server configured for an AD Connector or Microsoft AD directory.*

```
Disable-DSRadius -DirectoryId d-123456ijkl
```

• For API details, see DisableRadius in AWS Tools for PowerShell Cmdlet Reference.

**Disable-DSSso**

The following code example shows how to use Disable-DSSso.

**Tools for PowerShell**

*Example 1: This example disables single sign-on for a directory.*

```
Disable-DSSso -DirectoryId d-123456ijkl
```

• For API details, see DisableSso in AWS Tools for PowerShell Cmdlet Reference.
Enable-DSDirectoryShare

The following code example shows how to use Enable-DSDirectoryShare.

**Tools for PowerShell**

**Example 1:** This example shares a specified directory in your AWS account with another AWS Account using Handshake method.

```powershell
Enable-DSDirectoryShare -DirectoryId d-123456ijkl -ShareTarget_Id 123456784321 -ShareMethod HANDSHAKE -ShareTarget_Type ACCOUNT
```

**Output:**

d-9067012345


Enable-DSLDAPS

The following code example shows how to use Enable-DSLDAPS.

**Tools for PowerShell**

**Example 1:** This example activates the switch for the specific directory to always use LDAP secure calls.

```powershell
Enable-DSLDAPS -DirectoryId d-123456ijkl -Type Client
```


Enable-DSRadius

The following code example shows how to use Enable-DSRadius.

**Tools for PowerShell**

**Example 1:** This example enables multi-factor authentication (MFA) with the provided RADIUS server configuration for an AD Connector or Microsoft AD directory.
Enable-DSRadius -DirectoryId d-123456ijkl
-RadiusSettings_AuthenticationProtocol PAP
-RadiusSettings_DisplayLabel Radius
-RadiusSettings_RadiusPort 1812
-RadiusSettings_RadiusRetry 4
-RadiusSettings_RadiusServer 10.4.185.113
-RadiusSettings_RadiusTimeout 50
-RadiusSettings_SharedSecret wJalrXUtnFEMI

- For API details, see EnableRadius in AWS Tools for PowerShell Cmdlet Reference.

Enable-DSSso

The following code example shows how to use Enable-DSSso.

Tools for PowerShell

Example 1: This example enables single sign-on for a directory.

Enable-DSSso -DirectoryId d-123456ijkl

- For API details, see EnableSso in AWS Tools for PowerShell Cmdlet Reference.

Get-DSCertificate

The following code example shows how to use Get-DSCertificate.

Tools for PowerShell

Example 1: This example displays information about the certificate registered for a secured LDAP connection.

Get-DSCertificate -DirectoryId d-123456ijkl -CertificateId c-906731e34f

Output:

CertificateId : c-906731e34f
CommonName    : contoso-EC2AMAZ-CTGG2NM-CA
ExpiryDateTime: 4/15/2025 6:34:15 PM
• For API details, see DescribeCertificate in AWS Tools for PowerShell Cmdlet Reference.

Get-DSCertificateList

The following code example shows how to use Get-DSCertificateList.

Tools for PowerShell

Example 1: This example lists all the certificates registered for a secured LDAP connection for specified directory.

Get-DSCertificateList -DirectoryId d-123456ijkl

Output:

<table>
<thead>
<tr>
<th>CertificateId</th>
<th>CommonName</th>
<th>ExpiryDateTime</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>c-906731e34f</td>
<td>contoso-EC2AMAZ-CTGG2NM-CA</td>
<td>4/15/2025 6:34:15 PM</td>
<td>Registered</td>
</tr>
</tbody>
</table>

• For API details, see ListCertificates in AWS Tools for PowerShell Cmdlet Reference.

Get-DSConditionalForwarder

The following code example shows how to use Get-DSConditionalForwarder.

Tools for PowerShell

Example 1: This command gets all configured Conditional Forwarders of given Directory-id.

Get-DSConditionalForwarder -DirectoryId d-123456ijkl

Output:

<table>
<thead>
<tr>
<th>DnsIpAddr</th>
<th>RemoteDomainName</th>
<th>ReplicationScope</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
- For API details, see [DescribeConditionalForwarders](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-DSDirectory**

The following code example shows how to use Get-DSDirectory.

**Tools for PowerShell**

**Example 1:** This command Obtains information about the directories that belong to this account.

```
Get-DSDirectory | Select-Object DirectoryId, Name, DnsIpAddrs, Type
```

**Output:**

<table>
<thead>
<tr>
<th>DirectoryId</th>
<th>Name</th>
<th>DnsIpAddrs</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>d-123456abcd</td>
<td>abcd.example.com</td>
<td>{172.31.14.189, 172.31.13.145}</td>
<td>SimpleAD</td>
</tr>
<tr>
<td>d-123456efgh</td>
<td>wifi.example.com</td>
<td>{172.31.16.108, 172.31.10.56}</td>
<td>ADConnector</td>
</tr>
<tr>
<td>d-123456ijkl</td>
<td>lan2.example.com</td>
<td>{172.31.10.56, 172.31.16.108}</td>
<td>MicrosoftAD</td>
</tr>
</tbody>
</table>

- For API details, see [DescribeDirectories](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-DSDirectoryLimit**

The following code example shows how to use Get-DSDirectoryLimit.

**Tools for PowerShell**

**Example 1:** This example displays the directory limit information for the us-east-1 region.

```
Get-DSDirectoryLimit -Region us-east-1
```

**Output:**

CloudOnlyDirectoriesCurrentCount : 1
CloudOnlyDirectoriesLimit : 10
CloudOnlyDirectoriesLimitReached : False
CloudOnlyMicrosoftADCurrentCount : 1
CloudOnlyMicrosoftADLimit : 20
CloudOnlyMicrosoftADLimitReached : False
ConnectedDirectoriesCurrentCount : 1
ConnectedDirectoriesLimit : 10

- For API details, see [GetDirectoryLimits](aws-getdirectorylimits) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-DSDomainControllerList**

The following code example shows how to use Get-DSDomainControllerList.

**Tools for PowerShell**

**Example 1: This command gets the detailed list of Domain Controllers launched for mentioned directory-id**

```
Get-DSDomainControllerList -DirectoryId d-123456ijkl
```

**Output:**

```
AvailabilityZone          : us-east-1b
DirectoryId               : d-123456ijkl
DnsIpAddr                 : 172.31.16.108
DomainControllerId        : dc-1234567aa6
Status                    : Active
StatusLastUpdatedDateTime : 4/24/2019 1:37:54 PM
StatusReason              :
SubnetId                  : subnet-1234kkaa
VpcId                     : vpc-123459d

AvailabilityZone          : us-east-1d
DirectoryId               : d-123456ijkl
DnsIpAddr                 : 172.31.10.56
DomainControllerId        : dc-1234567aa7
Status                    : Active
StatusLastUpdatedDateTime : 4/4/2019 5:14:31 AM
StatusReason              :
```
SubnetId                  : subnet-5678ffbb
VpcId                     : vpc-123459d

- For API details, see [DescribeDomainControllers](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-DSEventTopic

The following code example shows how to use `Get-DSEventTopic`.

#### Tools for PowerShell

**Example 1:** This command shows information of configured SNS Topic for notification while directory status changes.

```powershell
Get-DSEventTopic -DirectoryId d-123456ijkl
```

**Output:**

- CreatedDateTime : 12/13/2019 11:15:32 AM
- DirectoryId     : d-123456ijkl
- Status          : Registered
- TopicName       : snstopicname

- For API details, see [DescribeEventTopics](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-DSIpRouteList

The following code example shows how to use `Get-DSIpRouteList`.

#### Tools for PowerShell

**Example 1:** This command gets the public IP address blocks configured in Directory IP Routing

```powershell
Get-DSIpRouteList -DirectoryId d-123456ijkl
```

**Output:**

- AddedDateTime       : 12/13/2019 12:27:22 PM
CidrIp              : 203.0.113.5/32
Description         : Public IP of On-Prem DNS Server
DirectoryId         : d-123456ijkl
IpRouteStatusMsg    : Added
IpRouteStatusReason :

- For API details, see [ListIpRoutes](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-DSLDAPSSetting

The following code example shows how to use Get-DSLDAPSSetting.

**Tools for PowerShell**

**Example 1:** This example describes the status of LDAP security for the specified directory.

```
Get-DSLDAPSSetting -DirectoryId d-123456ijkl
```

**Output:**

```
<table>
<thead>
<tr>
<th>LastUpdatedDateTime</th>
<th>LDAPSStatus</th>
<th>LDAPSStatusReason</th>
</tr>
</thead>
<tbody>
<tr>
<td>4/15/2020 6:51:03 PM</td>
<td>Enabled</td>
<td>LDAPS is enabled successfully.</td>
</tr>
</tbody>
</table>
```

- For API details, see [DescribeLDAPSSettings](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-DSLogSubscriptionList

The following code example shows how to use Get-DSLogSubscriptionList.

**Tools for PowerShell**

**Example 1:** This command gets the log subscriptions information of specified directory-id

```
Get-DSLogSubscriptionList -DirectoryId d-123456ijkl
```

**Output:**

```
DirectoryId  LogGroupName  SubscriptionCreatedDateTime
-------------  ----------------  ---------------------------------
```
• For API details, see [ListLogSubscriptions](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-DSResourceTag**

The following code example shows how to use `Get-DSResourceTag`.

**Tools for PowerShell**

**Example 1:** This command gets all the Tags of specified Directory.

```
Get-DSResourceTag -ResourceId d-123456ijkl
```

**Output:**

```
Key   Value
---   ----- 
myTag myTagValue
```

• For API details, see [ListTagsForResource](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-DSSchemaExtension**

The following code example shows how to use `Get-DSSchemaExtension`.

**Tools for PowerShell**

**Example 1:** This example lists all schema extensions applied to a Microsoft AD Directory.

```
Get-DSSchemaExtension -DirectoryId d-123456ijkl
```

**Output:**

```
Description                      : ManagedADSchemaExtension
DirectoryId                      : d-123456ijkl
```
For API details, see [ListSchemaExtensions](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-DSSharedDirectory

The following code example shows how to use `Get-DSSharedDirectory`.

**Tools for PowerShell**

**Example 1:** This example gets the shared directories of your AWS Account

```powershell
Get-DSSharedDirectory -OwnerDirectoryId d-123456ijkl -SharedDirectoryId d-9067012345
```

**Output:**

```powershell
CreatedDateTime     : 12/30/2019 4:34:37 AM
LastUpdatedDateTime : 12/30/2019 4:35:22 AM
OwnerAccountId      : 123456781234
OwnerDirectoryId    : d-123456ijkl
SharedAccountId     : 123456784321
SharedDirectoryId   : d-9067012345
ShareMethod         : HANDSHAKE
ShareNotes          : This is a test Sharing
ShareStatus         : Shared
```

For API details, see [DescribeSharedDirectories](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-DSSnapshot

The following code example shows how to use `Get-DSSnapshot`.

**Tools for PowerShell**

**Example 1:** This command gets information about the specified directory snapshots that belong to this account.
Get-DSSnapshot -DirectoryId d-123456ijkl

Output:

DirectoryId : d-123456ijkl
Name        :
SnapshotId  : s-9064bd1234
StartTime   : 12/13/2019 6:33:01 PM
Status      : Completed
Type        : Auto

DirectoryId : d-123456ijkl
Name        :
SnapshotId  : s-9064bb4321
StartTime   : 12/9/2019 9:48:11 PM
Status      : Completed
Type        : Auto

- For API details, see DescribeSnapshots in AWS Tools for PowerShell Cmdlet Reference.

Get-DSSnapshotLimit

The following code example shows how to use Get-DSSnapshotLimit.

Tools for PowerShell

Example 1: This command gets the manual snapshot limits for a specified directory.

Get-DSSnapshotLimit -DirectoryId d-123456ijkl

Output:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>False</td>
</tr>
</tbody>
</table>

- For API details, see GetSnapshotLimits in AWS Tools for PowerShell Cmdlet Reference.

Get-DSTrust

The following code example shows how to use Get-DSTrust.
Tools for PowerShell

Example 1: This command gets the information of trust relationships created for specified directory-id.

Get-DSTrust -DirectoryId d-123456abcd

Output:

<table>
<thead>
<tr>
<th>CreatedDateTime</th>
<th>7/5/2019 4:55:42 AM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DirectoryId</td>
<td>d-123456abcd</td>
</tr>
<tr>
<td>LastUpdatedDateTime</td>
<td>7/5/2019 4:56:04 AM</td>
</tr>
<tr>
<td>RemoteDomainName</td>
<td>contoso.com</td>
</tr>
<tr>
<td>SelectiveAuth</td>
<td>Disabled</td>
</tr>
<tr>
<td>StateLastUpdatedDateTime</td>
<td>7/5/2019 4:56:04 AM</td>
</tr>
<tr>
<td>TrustDirection</td>
<td>One-Way: Incoming</td>
</tr>
<tr>
<td>TrustId</td>
<td>t-9067157123</td>
</tr>
<tr>
<td>TrustState</td>
<td>Created</td>
</tr>
<tr>
<td>TrustStateReason</td>
<td></td>
</tr>
<tr>
<td>TrustType</td>
<td>Forest</td>
</tr>
</tbody>
</table>

- For API details, see [DescribeTrusts](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

New-DSAlias

The following code example shows how to use New-DSAlias.

Example 1: This command creates an alias for a directory and assigns the alias to the specified directory-id.

New-DSAlias -DirectoryId d-123456ijkl -Alias MyOrgName

Output:

<table>
<thead>
<tr>
<th>Alias</th>
<th>DirectoryId</th>
</tr>
</thead>
<tbody>
<tr>
<td>myorgname</td>
<td>d-123456ijkl</td>
</tr>
</tbody>
</table>
For API details, see CreateAlias in AWS Tools for PowerShell Cmdlet Reference.

New-DSComputer

The following code example shows how to use New-DSComputer.

Tools for PowerShell

Example 1: This example creates a new Active Directory computer object.

```powershell
Get-DSCredential
New-DSComputer -DirectoryId d-123456ijkl -ComputerName ADMemberServer -Password $Password
```

Output:

<table>
<thead>
<tr>
<th>ComputerAttributes</th>
<th>ComputerId</th>
</tr>
</thead>
<tbody>
<tr>
<td>ComputerName</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>[WindowsSamName, DistinguishedName]</td>
<td>S-1-5-21-1191241402-978882507-2717148213-1662</td>
</tr>
<tr>
<td>ADMemberServer</td>
<td></td>
</tr>
</tbody>
</table>

For API details, see CreateComputer in AWS Tools for PowerShell Cmdlet Reference.

New-DSConditionalForwarder

The following code example shows how to use New-DSConditionalForwarder.

Tools for PowerShell

Example 1: This example creates a Conditional forwarder in specified AWS Directory-id.

```powershell
New-DSConditionalForwarder -DirectoryId d-123456ijkl -DnsIpAddr 172.31.36.96,172.31.10.56 -RemoteDomainName contoso.com
```

For API details, see CreateConditionalForwarder in AWS Tools for PowerShell Cmdlet Reference.

New-DSDirectory

The following code example shows how to use New-DSDirectory.
Tools for PowerShell

Example 1: This example create a new Simple AD directory.

```
New-DSDirectory -Name corp.example.com -Password $Password -Size Small -
VpcSettings_VpcId vpc-123459d -VpcSettings_SubnetIds subnet-1234kkaa,subnet-5678ffbb
```

• For API details, see [CreateDirectory](#) in *AWS Tools for PowerShell Cmdlet Reference*.

New-DSLogSubscription

The following code example shows how to use New-DSLogSubscription.

Tools for PowerShell

Example 1: This example creates a subscription to forward real-time Directory Service domain controller security logs to the specified Amazon CloudWatch log group in your AWS account.

```
New-DSLogSubscription -DirectoryId d-123456ijkl -LogGroupName /aws/directoryservice/
d-123456ijkl-lan2.example.com
```

• For API details, see [CreateLogSubscription](#) in *AWS Tools for PowerShell Cmdlet Reference*.

New-DSMicrosoftAD

The following code example shows how to use New-DSMicrosoftAD.

Tools for PowerShell

Example 1: This example creates new Microsoft AD Directory in AWS Cloud.

```
New-DSMicrosoftAD -Name corp.example.com -Password $Password -edition Standard -
VpcSettings_VpcId vpc-123459d -VpcSettings_SubnetIds subnet-1234kkaa,subnet-5678ffbb
```

• For API details, see [CreateMicrosoftAD](#) in *AWS Tools for PowerShell Cmdlet Reference*.

New-DSSnapshot

The following code example shows how to use New-DSSnapshot.
Examples:

### Tools for PowerShell

#### Example 1: This example creates a directory snapshot

```powershell
New-DSSnapshot -DirectoryId d-123456ijkl
```

- For API details, see `CreateSnapshot` in *AWS Tools for PowerShell Cmdlet Reference*.

#### New-DSTrust

The following code example shows how to use `New-DSTrust`.

```powershell
New-DSTrust -DirectoryId d-123456ijkl -RemoteDomainName contoso.com -TrustDirection Two-Way -TrustType Forest -TrustPassword $Password -ConditionalForwarderIpAddr 172.31.36.96
```

**Output:**

`t-9067157123`

- For API details, see `CreateTrust` in *AWS Tools for PowerShell Cmdlet Reference*.

#### Register-DSCertificate

The following code example shows how to use `Register-DSCertificate`.

```powershell
Register-DSCertificate -DirectoryId d-123456ijkl -CertificateData $Certificate
```

**Output:**

`AWS Directory Service`
Register-DSEventTopic

The following code example shows how to use Register-DSEventTopic.

Tools for PowerShell

Example 1: This example associate a directory as a publisher with an SNS topic.

```
Register-DSEventTopic -DirectoryId d-123456ijkl -TopicName snstopicname
```

For API details, see RegisterEventTopic in AWS Tools for PowerShell Cmdlet Reference.

Remove-DSConditionalForwarder

The following code example shows how to use Remove-DSConditionalForwarder.

Tools for PowerShell

Example 1: This example removes the conditional forwarder that has been set up for your AWS Directory.

```
Remove-DSConditionalForwarder -DirectoryId d-123456ijkl -RemoteDomainName contoso.com
```

For API details, see DeleteConditionalForwarder in AWS Tools for PowerShell Cmdlet Reference.

Remove-DSDirectory

The following code example shows how to use Remove-DSDirectory.

Tools for PowerShell

Example 1: This example deletes an AWS Directory service directory (Simple AD/Microsoft AD/AD Connector)
Remove-DSDirectory -DirectoryId d-123456ijkl

- For API details, see [DeleteDirectory](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Remove-DSIpRoute

The following code example shows how to use Remove-DSIpRoute.

**Tools for PowerShell**

**Example 1:** This command removes the specified IP from Configured IP routes of Directory-id.

```powershell
Remove-DSIpRoute -DirectoryId d-123456ijkl -CidrIp 203.0.113.5/32
```

- For API details, see [RemoveIpRoutes](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Remove-DSLogSubscription

The following code example shows how to use Remove-DSLogSubscription.

**Tools for PowerShell**

**Example 1:** This command removes the Log Subscription of specified Directory-id

```powershell
Remove-DSLogSubscription -DirectoryId d-123456ijkl
```

- For API details, see [DeleteLogSubscription](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Remove-DSResourceTag

The following code example shows how to use Remove-DSResourceTag.

**Tools for PowerShell**

**Example 1:** This command removes the Resource Tag assigned to the specified Directory-id

```powershell
Remove-DSResourceTag -ResourceId d-123456ijkl -TagKey myTag
```
Remove-DSSnapshot

The following code example shows how to use Remove-DSSnapshot.

Tools for PowerShell

Example 1: This example removes the manually created snapshot.

```powershell
Remove-DSSnapshot -SnapshotId s-9068b488kc
```

- For API details, see [DeleteSnapshot](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Remove-DSTrust

The following code example shows how to use Remove-DSTrust.

Tools for PowerShell

Example 1: This example removes the existing trust relationship between your AWS Managed AD Directory and an external domain.

```powershell
Get-DSTrust -DirectoryId d-123456ijkl -Select Trusts.TrustId | Remove-DSTrust
```

Output:

```
t-9067157123
```

- For API details, see [DeleteTrust](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Reset-DSUserPassword

The following code example shows how to use Reset-DSUserPassword.

Tools for PowerShell

Example 1: This example resets the password of Active Directory user named ADUser in AWS Managed microsoft AD or Simple AD Directory
Reset-DSUserPassword -UserName ADuser -DirectoryId d-123456ijkl -NewPassword $Password

- For API details, see ResetUserPassword in AWS Tools for PowerShell Cmdlet Reference.

Restore-DSFromSnapshot

The following code example shows how to use Restore-DSFromSnapshot.

Tools for PowerShell

Example 1: This example restores a directory using an existing directory snapshot.

Restore-DSFromSnapshot -SnapshotId s-9068b488kc

- For API details, see RestoreFromSnapshot in AWS Tools for PowerShell Cmdlet Reference.

Set-DSDomainControllerCount

The following code example shows how to use Set-DSDomainControllerCount.

Tools for PowerShell

Example 1: This example sets the number of domain controller to 3 for specified directory-id.

Set-DSDomainControllerCount -DirectoryId d-123456ijkl -DesiredNumber 3

- For API details, see UpdateNumberOfDomainControllers in AWS Tools for PowerShell Cmdlet Reference.

Start-DSSchemaExtension

The following code example shows how to use Start-DSSchemaExtension.

Tools for PowerShell

Example 1: This Example Applies a schema extension to a Microsoft AD directory.

$ldif = Get-Content D:\Users\Username\Downloads\ExtendedSchema.ldf -Raw
Start-DSSchemaExtension -DirectoryId d-123456ijkl -CreateSnapshotBeforeSchemaExtension $true -Description ManagedADSchemaExtension -LdifContent $ldif

**Output:**

e-9067306643

- For API details, see [StartSchemaExtension](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Stop-DSSchemaExtension**

The following code example shows how to use Stop-DSSchemaExtension.

**Tools for PowerShell**

**Example 1:** This example cancels an in-progress schema extension to a Microsoft AD directory.

Stop-DSSchemaExtension -DirectoryId d-123456ijkl -SchemaExtensionId e-9067306643

- For API details, see [CancelSchemaExtension](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Unregister-DSCertificate**

The following code example shows how to use Unregister-DSCertificate.

**Tools for PowerShell**

**Example 1:** This example deletes from the system the certificate that was registered for a secured LDAP connection.

Unregister-DSCertificate -DirectoryId d-123456ijkl -CertificateId c-906731e34f

- For API details, see [DeregisterCertificate](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Unregister-DSEventTopic**

The following code example shows how to use Unregister-DSEventTopic.
Tools for PowerShell

Example 1: This example removes the specified directory as a publisher to the specified SNS topic.

```powershell
Unregister-DSEventTopic -DirectoryId d-123456ijkl -TopicName snstopicname
```

- For API details, see [DeregisterEventTopic](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Update-DSCConditionalForwarder

The following code example shows how to use Update-DSCConditionalForwarder.

Tools for PowerShell

Example 1: This example updates a conditional forwarder that has been set up for your AWS directory.

```powershell
Update-DSCConditionalForwarder -DirectoryId d-123456ijkl -DnsIpAddr 172.31.36.96,172.31.16.108 -RemoteDomainName contoso.com
```

- For API details, see [UpdateConditionalForwarder](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Update-DSRadius

The following code example shows how to use Update-DSRadius.

Tools for PowerShell

Example 1: This example updates RADIUS server information for an AD Connector or Microsoft AD directory.

```powershell
Update-DSRadius -DirectoryId d-123456ijkl -RadiusSettings_RadiusRetry 3
```

- For API details, see [UpdateRadius](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Update-DSTrust

The following code example shows how to use Update-DSTrust.
Tools for PowerShell

Example 1: This example updates the SelectiveAuth parameter of specified trust-id from Disabled to Enabled.

Update-DSTrust -TrustId t-9067157123 -SelectiveAuth Enabled

Output:

<table>
<thead>
<tr>
<th>RequestId</th>
<th>TrustId</th>
</tr>
</thead>
<tbody>
<tr>
<td>138864a7-c9a8-4ad1-a828-eae479e85b45</td>
<td>t-9067157123</td>
</tr>
</tbody>
</table>

• For API details, see UpdateTrust in AWS Tools for PowerShell Cmdlet Reference.

AWS DMS examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with AWS DMS.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

• Actions

Actions

New-DMSReplicationTask

The following code example shows how to use New-DMSReplicationTask.
Tools for PowerShell

Example 1: This example creates a new AWS Database Migration Service replication task that uses CdcStartTime instead of CdcStartPosition. The MigrationType is set to "full-load-and-cdc", meaning the target table must be empty. The new task is tagged with a tag that has a key of Stage and a key value of Test. For more information about the values used by this cmdlet, see Creating a Task (https://docs.aws.amazon.com/dms/latest/userguide/CHAP_Tasks.Creating.html) in the AWS Database Migration Service User Guide.

```powershell
  -CdcStartTime "2019-08-08T12:12:12"
  -CdcStopPosition "server_time:2019-08-09T12:12:12"
  -MigrationType "full-load-and-cdc"
  -ReplicationTaskIdentifier "task1"
  -ReplicationTaskSetting ""
  -SourceEndpointArn "arn:aws:dms:us-east-1:123456789012:endpoint:EXAMPLEW5UANC7Y3P4EEXAMPLE"
  -TableMapping "file://home/testuser/table-mappings.json"
  -Tag @{"Key"="Stage";"Value"="Test"}
  -TargetEndpointArn "arn:aws:dms:us-east-1:123456789012:endpoint:EXAMPLEJZASXWHTWCLNEXAMPLE"
```

- For API details, see CreateReplicationTask in AWS Tools for PowerShell Cmdlet Reference.

DynamoDB examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with DynamoDB.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics
Actions

Add-DDBIndexSchema

The following code example shows how to use Add-DDBIndexSchema.

Tools for PowerShell

Example 1: Creates an empty TableSchema object and adds a new local secondary index definition to it before writing the TableSchema object to the pipeline.

```
$schema | Add-DDBIndexSchema -IndexName "LastPostIndex" -RangeKeyName "LastPostDateTime" -RangeKeyDataType "S" -ProjectionType "keys_only"
$schema = New-DDBTableSchema
```

Output:

```
AttributeSchema                             KeySchema
LocalSecondaryIndexSchema                   -------
-----------                             -------
{LastPostDateTime}                          {}
{LastPostIndex}
```

Example 2: Adds a new local secondary index definition to the supplied TableSchema object before writing the TableSchema object back to the pipeline. The TableSchema object can also be supplied using the -Schema parameter.

```
New-DDBTableSchema | Add-DDBIndexSchema -IndexName "LastPostIndex" -RangeKeyName "LastPostDateTime" -RangeKeyDataType "S" -ProjectionType "keys_only"
```

Output:

```
AttributeSchema                             KeySchema
LocalSecondaryIndexSchema                   -------
-----------                             -------
```

DynamoDB

323
For API details, see Add-DDBIndexSchema in AWS Tools for PowerShell Cmdlet Reference.

Add-DDBKeySchema

The following code example shows how to use Add-DDBKeySchema.

Tools for PowerShell

Example 1: Creates an empty TableSchema object and adds key and attribute definition entries to it using the specified key data before writing the TableSchema object to the pipeline. The key type is declared to be 'HASH' by default; use the -KeyType parameter with a value of 'RANGE' to declare a range key.

```powershell
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
```

Output:

```
AttributeSchema                             KeySchema
------------------------                   ---------
{ForumName}                                 {ForumName}
{ }
```

Example 2: Adds new key and attribute definition entries to the supplied TableSchema object before writing the TableSchema object to the pipeline. The key type is declared to be 'HASH' by default; use the -KeyType parameter with a value of 'RANGE' to declare a range key. The TableSchema object can also be supplied using the -Schema parameter.

```powershell
New-DDBTableSchema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
```

Output:

```
AttributeSchema                             KeySchema
------------------------                   ---------
LocalSecondaryIndexSchema                   
{ForumName}                                 {ForumName}
{ }
```
ConvertFrom-DDBItem

The following code example shows how to use ConvertFrom-DDBItem.

**Example 1:** ConvertFrom-DDBItem is used to convert the result of Get-DDBItem from a hashtable of DynamoDB AttributeValues to a hashtable of common types like string and double.

```powershell
@{
    SongTitle = 'Somewhere Down The Road'
    Artist    = 'No One You Know'
} | ConvertTo-DDBItem
Get-DDBItem -TableName 'Music' -Key $key | ConvertFrom-DDBItem
```

**Output:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genre</td>
<td>Country</td>
</tr>
<tr>
<td>Artist</td>
<td>No One You Know</td>
</tr>
<tr>
<td>Price</td>
<td>1.94</td>
</tr>
<tr>
<td>CriticRating</td>
<td>9</td>
</tr>
<tr>
<td>SongTitle</td>
<td>Somewhere Down The Road</td>
</tr>
<tr>
<td>AlbumTitle</td>
<td>Somewhat Famous</td>
</tr>
</tbody>
</table>

ConvertTo-DDBItem

The following code example shows how to use ConvertTo-DDBItem.

**Example 1:** ConvertTo-DDBItem is used to convert the result of Get-DDBItem from a hashtable of DynamoDB AttributeValues to a hashtable of common types like string and double.
Example 1: An example for converting a hashtable into a dictionary of DynamoDB attribute values.

```powershell
@{
    SongTitle = 'Somewhere Down The Road'
    Artist    = 'No One You Know'
} | ConvertTo-DDBItem

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SongTitle</td>
<td>Amazon.DynamoDBv2.Model.AttributeValue</td>
</tr>
<tr>
<td>Artist</td>
<td>Amazon.DynamoDBv2.Model.AttributeValue</td>
</tr>
</tbody>
</table>
```

Example 2: An example for converting a hashtable into a dictionary of DynamoDB attribute values.

```powershell
@{
    MyMap        = @{
        MyString = 'my string'
    }
    MyStringSet  = [System.Collections.Generic.HashSet[String]]@('my', 'string')
    MyNumericSet = [System.Collections.Generic.HashSet[Int]]@1, 2, 3
        (System.IO.MemoryStream::new([Text.Encoding]:.UTF8.GetBytes('my'))),
        (System.IO.MemoryStream::new([Text.Encoding]:.UTF8.GetBytes('string')))
    )
    MyList1      = @('my', 'string')
    MyList2      = [System.Collections.Generic.List[Int]]@1, 2
    MyList3      = [System.Collections.ArrayList]@(1, 2, $true)
} | ConvertTo-DDBItem

Output:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyStringSet</td>
<td>Amazon.DynamoDBv2.Model.AttributeValue</td>
</tr>
<tr>
<td>MyList1</td>
<td>Amazon.DynamoDBv2.Model.AttributeValue</td>
</tr>
<tr>
<td>MyNumericSet</td>
<td>Amazon.DynamoDBv2.Model.AttributeValue</td>
</tr>
<tr>
<td>MyList2</td>
<td>Amazon.DynamoDBv2.Model.AttributeValue</td>
</tr>
<tr>
<td>MyBinarySet</td>
<td>Amazon.DynamoDBv2.Model.AttributeValue</td>
</tr>
</tbody>
</table>
For API details, see [ConvertTo-DDBItem](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-DDBBatchItem**

The following code example shows how to use Get-DDBBatchItem.

**Tools for PowerShell**

**Example 1: Gets the item with the SongTitle "Somewhere Down The Road" from the DynamoDB tables 'Music' and 'Songs'.**

```powershell
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$keysAndAttributes = New-Object Amazon.DynamoDBv2.Model.KeysAndAttributes
$list.Add($key)
$keysAndAttributes.Keys = $list

$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.KeysAndAttributes]$keysAndAttributes
    'Songs' = [Amazon.DynamoDBv2.Model.KeysAndAttributes]$keysAndAttributes
}

$batchItems = Get-DDBBatchItem -RequestItem $requestItem
$batchItems.GetEnumerator() | ForEach-Object {$PSItem.Value} | ConvertFrom-DDBItem
```

**Output:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Artist</td>
<td>No One You Know</td>
</tr>
<tr>
<td>SongTitle</td>
<td>Somewhere Down The Road</td>
</tr>
<tr>
<td>AlbumTitle</td>
<td>Somewhat Famous</td>
</tr>
<tr>
<td>CriticRating</td>
<td>10</td>
</tr>
</tbody>
</table>
For API details, see [BatchGetItem](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-DDBItem**

The following code example shows how to use Get-DDBItem.

**Tools for PowerShell**

**Example 1: Returns the DynamoDB item with the partition key SongTitle and the sort key Artist.**

```powershell
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

Get-DDBItem -TableName 'Music' -Key $key | ConvertFrom-DDBItem
```

**Output:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genre</td>
<td>Country</td>
</tr>
<tr>
<td>SongTitle</td>
<td>Somewhere Down The Road</td>
</tr>
<tr>
<td>AlbumTitle</td>
<td>Somewhat Famous</td>
</tr>
<tr>
<td>CriticRating</td>
<td>9</td>
</tr>
<tr>
<td>Genre</td>
<td>Country</td>
</tr>
<tr>
<td>Price</td>
<td>1.94</td>
</tr>
<tr>
<td>Artist</td>
<td>No One You Know</td>
</tr>
<tr>
<td>AlbumTitle</td>
<td>Somewhat Famous</td>
</tr>
</tbody>
</table>

For API details, see [GetItem](#) in *AWS Tools for PowerShell Cmdlet Reference*.  

DynamoDB
Get-DDBTable

The following code example shows how to use Get-DDBTable.

Tools for PowerShell

Example 1: Returns details of the specified table.

Get-DDBTable -TableName "myTable"

- For API details, see DescribeTable in AWS Tools for PowerShell Cmdlet Reference.

Get-DDBTableList

The following code example shows how to use Get-DDBTableList.

Tools for PowerShell

Example 1: Returns details of all tables, automatically iterating until the service indicates no further tables exist.

Get-DDBTableList

Example 2: Manually iterates for details of all tables, returning up to 10 tables per call until the service indicates no further tables exist.

$nextToken = $null
do {
    Get-DDBTableList -ExclusiveStartTableName $nextToken -Limit 10
    $nextToken = $AWSHistory.LastServiceResponse.LastEvaluatedTableName
} while ($nextToken -ne $null)

- For API details, see ListTables in AWS Tools for PowerShell Cmdlet Reference.

Invoke-DDBQuery

The following code example shows how to use Invoke-DDBQuery.
Tools for PowerShell

**Example 1: Invokes a query that returns DynamoDB items with the specified SongTitle and Artist.**

```powershell
$invokeDDBQuery = @{
    TableName = 'Music'
    KeyConditionExpression = ' SongTitle = :SongTitle and Artist = :Artist'
    ExpressionAttributeValues = @{
        ':SongTitle' = 'Somewhere Down The Road'
        ':Artist' = 'No One You Know'
    } | ConvertTo-DDBItem
}
Invoke-DDBQuery @invokeDDBQuery | ConvertFrom-DDBItem
```

**Output:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genre</td>
<td>Country</td>
</tr>
<tr>
<td>Artist</td>
<td>No One You Know</td>
</tr>
<tr>
<td>Price</td>
<td>1.94</td>
</tr>
<tr>
<td>CriticRating</td>
<td>9</td>
</tr>
<tr>
<td>SongTitle</td>
<td>Somewhere Down The Road</td>
</tr>
<tr>
<td>AlbumTitle</td>
<td>Somewhat Famous</td>
</tr>
</tbody>
</table>

- For API details, see [Query](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Invoke-DDBScan**

The following code example shows how to use Invoke-DDBScan.

Tools for PowerShell

**Example 1: Returns all items in the Music table.**

```powershell
Invoke-DDBScan -TableName 'Music' | ConvertFrom-DDBItem
```

**Output:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example 2: Returns items in the Music table with a CriticRating greater than or equal to nine.

```powershell
$scanFilter = @{
    CriticRating = [Amazon.DynamoDBv2.Model.Condition]@{
        AttributeValueList = @(@{N = '9'})
        ComparisonOperator = 'GE'
    }
}
Invoke-DDBScan -TableName 'Music' -ScanFilter $scanFilter | ConvertFrom-DDBItem
```

Output:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genre</td>
<td>Country</td>
</tr>
<tr>
<td>Artist</td>
<td>No One You Know</td>
</tr>
<tr>
<td>Price</td>
<td>1.94</td>
</tr>
<tr>
<td>CriticRating</td>
<td>9</td>
</tr>
<tr>
<td>SongTitle</td>
<td>Somewhere Down The Road</td>
</tr>
<tr>
<td>AlbumTitle</td>
<td>Somewhat Famous</td>
</tr>
<tr>
<td>Genre</td>
<td>Country</td>
</tr>
<tr>
<td>Artist</td>
<td>No One You Know</td>
</tr>
<tr>
<td>Price</td>
<td>1.98</td>
</tr>
<tr>
<td>CriticRating</td>
<td>8.4</td>
</tr>
<tr>
<td>SongTitle</td>
<td>My Dog Spot</td>
</tr>
<tr>
<td>AlbumTitle</td>
<td>Hey Now</td>
</tr>
</tbody>
</table>

- For API details, see [Scan](https://aws.amazon.com/documentation/dynamodb/reference/scan/) in AWS Tools for PowerShell Cmdlet Reference.

**New-DDBTable**

The following code example shows how to use New-DDBTable.
Tools for PowerShell

Example 1: This example creates a table named Thread that has a primary key consisting of 'ForumName' (key type hash) and 'Subject' (key type range). The schema used to construct the table can be piped into each cmdlet as shown or specified using the -Schema parameter.

```powershell
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyType RANGE -KeyDataType "S"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Output:

```
AttributeDefinitions   : {ForumName, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {}
```

Example 2: This example creates a table named Thread that has a primary key consisting of 'ForumName' (key type hash) and 'Subject' (key type range). A local secondary index is also defined. The key of the local secondary index will be set automatically from the primary hash key on the table (ForumName). The schema used to construct the table can be piped into each cmdlet as shown or specified using the -Schema parameter.

```powershell
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S"
$schema | Add-DDBIndexSchema -IndexName "LastPostIndex" -RangeKeyName "LastPostDateTime" -RangeKeyDataType "S" -ProjectionType "keys_only"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Output:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
```
KeySchema                     : {ForumName, Subject}
TableStatus                   : CREATING
CreationDateTime              : 10/28/2013 4:39:49 PM
ProvisionedThroughput         : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes                : 0
ItemCount                     : 0
LocalSecondaryIndexes         : {LastPostIndex}

**Example 3:** This example shows how to use a single pipeline to create a table named `Thread` that has a primary key consisting of 'ForumName' (key type hash) and 'Subject' (key type range) and a local secondary index. The Add-DDBKeySchema and Add-DDBIndexSchema create a new TableSchema object for you if one is not supplied from the pipeline or the -Schema parameter.

```
New-DDBTableSchema |  
Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S" |  
Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S" |  
Add-DDBIndexSchema -IndexName "LastPostIndex" `  
   -RangeKeyName "LastPostDateTime" `  
   -RangeKeyDataType "S" `  
   -ProjectionType "keys_only" |  
New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

**Output:**

```
AttributeDefinitions : {ForumName, LastPostDateTime, Subject}
TableName            : Thread
KeySchema             : {ForumName, Subject}
TableStatus           : CREATING
CreationDateTime      : 10/28/2013 4:39:49 PM
ProvisionedThroughput : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes        : 0
ItemCount             : 0
LocalSecondaryIndexes : {LastPostIndex}
```

- For API details, see [CreateTable](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**New-DDBTableSchema**

The following code example shows how to use New-DDBTableSchema.
Tools for PowerShell

Example 1: Creates an empty TableSchema object ready to accept key and index definitions for use in creating a new Amazon DynamoDB table. The returned object can be piped into the Add-DDBKeySchema, Add-DDBIndexSchema and New-DDBTable cmdlets or passed to them using the -Schema parameter on each cmdlet.

New-DDBTableSchema

Output:

<table>
<thead>
<tr>
<th>AttributeSchema</th>
<th>KeySchema</th>
</tr>
</thead>
<tbody>
<tr>
<td>LocalSecondaryIndexSchema</td>
<td>{}</td>
</tr>
<tr>
<td>{}</td>
<td>{}</td>
</tr>
</tbody>
</table>

- For API details, see [New-DDBTableSchema](#) in AWS Tools for PowerShell Cmdlet Reference.

Remove-DDBItem

The following code example shows how to use Remove-DDBItem.

Tools for PowerShell

Example 1: Removes the DynamoDB item that matches the provided key.

```powershell
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem
Remove-DDBItem -TableName 'Music' -Key $key -Confirm:$false
```

- For API details, see [DeleteItem](#) in AWS Tools for PowerShell Cmdlet Reference.

Remove-DDBTable

The following code example shows how to use Remove-DDBTable.
Tools for PowerShell

Example 1: Deletes the specified table. You are prompted for confirmation before the operation proceeds.

Remove-DDBTable -TableName "myTable"

Example 2: Deletes the specified table. You are not prompted for confirmation before the operation proceeds.

Remove-DDBTable -TableName "myTable" -Force

• For API details, see [DeleteTable](#) in *AWS Tools for PowerShell Cmdlet Reference.*

Set-DDBBatchItem

The following code example shows how to use Set-DDBBatchItem.

Tools for PowerShell

Example 1: Creates a new item, or replaces an existing item with a new item in the DynamoDB tables Music and Songs.

```powershell
$item = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
    AlbumTitle = 'Somewhat Famous'
    Price = 1.94
    Genre = 'Country'
    CriticRating = 10.0
} | ConvertTo-DDBItem

$writeRequest = New-Object Amazon.DynamoDBv2.Model.WriteRequest

Output:

$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
    'Songs' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
}
Set-DDBBatchItem -RequestItem $requestItem

- For API details, see [BatchWriteItem](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Set-DDBItem**

The following code example shows how to use Set-DDBItem.

**Tools for PowerShell**

**Example 1: Creates a new item, or replaces an existing item with a new item.**

```powershell
$item = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
  AlbumTitle = 'Somewhat Famous'
  Price = 1.94
  Genre = 'Country'
  CriticRating = 9.0
}
Set-DDBItem -TableName 'Music' -Item $item
```

- For API details, see [PutItem](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Update-DDBItem**

The following code example shows how to use Update-DDBItem.

**Tools for PowerShell**

**Example 1: Sets the genre attribute to 'Rap' on the DynamoDB item with the partition key SongTitle and the sort key Artist.**

```powershell
$key = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
}
Set-DDBItem -TableName 'Music' -Key $key
```

```powershell
$updateDdbItem = @{
  TableName = 'Music'
  Key = $key
}
```
UpdateExpression = 'set Genre = :val1'
ExpressionAttributeValue = (@{
    ':val1' = ([Amazon.DynamoDBv2.Model.AttributeValue]'Rap')
})
}
Update-DDBItem @updateDdbItem

Output:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genre</td>
<td>Rap</td>
</tr>
</tbody>
</table>

- For API details, see UpdateItem in AWS Tools for PowerShell Cmdlet Reference.

**Update-DDBTable**

The following code example shows how to use Update-DDBTable.

**Tools for PowerShell**

**Example 1: Updates the provisioned throughput for the given table.**

Update-DDBTable -TableName "myTable" -ReadCapacity 10 -WriteCapacity 5

- For API details, see UpdateTable in AWS Tools for PowerShell Cmdlet Reference.

**Amazon EC2 examples using Tools for PowerShell**

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with Amazon EC2.

*Actions* are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

*Scenarios* are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.
Topics

- Actions

Actions

Add-EC2CapacityReservation

The following code example shows how to use Add-EC2CapacityReservation.

Tools for PowerShell

**Example 1: This example creates a new Capacity Reservation with the specified attributes**

```powershell
Add-EC2CapacityReservation -InstanceType m4.xlarge -InstanceCount 2 -AvailabilityZone eu-west-1b -EbsOptimized True -InstancePlatform Windows
```

**Output:**

```
AvailabilityZone       : eu-west-1b
AvailableInstanceCount : 2
CapacityReservationId  : cr-0c1f2345db6f7cdba
CreateDate             : 3/28/2019 9:29:41 AM
EbsOptimized           : True
EndDate                : 1/1/0001 12:00:00 AM
EndDateType            : unlimited
EphemeralStorage       : False
InstanceMatchCriteria  : open
InstancePlatform       : Windows
InstanceType           : m4.xlarge
State                  : active
Tags                   : {}
Tenancy                : default
TotalInstanceCount     : 2
```

- For API details, see [CreateCapacityReservation](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Add-EC2InternetGateway

The following code example shows how to use Add-EC2InternetGateway.
Tools for PowerShell

**Example 1:** This example attaches the specified Internet gateway to the specified VPC.

```powershell
Add-EC2InternetGateway -InternetGatewayId igw-1a2b3c4d -VpcId vpc-12345678
```

**Example 2:** This example creates a VPC and an Internet gateway, and then attaches the Internet gateway to the VPC.

```powershell
$vpc = New-EC2Vpc -CidrBlock 10.0.0.0/16
New-EC2InternetGateway | Add-EC2InternetGateway -VpcId $vpc.VpcId
```

- For API details, see [AttachInternetGateway](#) in *AWS Tools for PowerShell Cmdlet Reference*.

---

**Add-EC2NetworkInterface**

The following code example shows how to use `Add-EC2NetworkInterface`.

**Tools for PowerShell**

**Example 1:** This example attaches the specified network interface to the specified instance.

```powershell
Add-EC2NetworkInterface -NetworkInterfaceId eni-12345678 -InstanceId i-1a2b3c4d -DeviceIndex 1
```

**Output:**

```powershell
eni-attach-1a2b3c4d
```

- For API details, see [AttachNetworkInterface](#) in *AWS Tools for PowerShell Cmdlet Reference*.

---

**Add-EC2Volume**

The following code example shows how to use `Add-EC2Volume`.

**Tools for PowerShell**

**Example 1:** This example attaches the specified volume to the specified instance and exposes it with the specified device name.
Add-EC2Volume -VolumeId vol-12345678 -InstanceId i-1a2b3c4d -Device /dev/sdh

Output:

<table>
<thead>
<tr>
<th>AttachTime</th>
<th>12/22/2015 1:53:58 AM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeleteOnTermination</td>
<td>False</td>
</tr>
<tr>
<td>Device</td>
<td>/dev/sdh</td>
</tr>
<tr>
<td>InstanceId</td>
<td>i-1a2b3c4d</td>
</tr>
<tr>
<td>State</td>
<td>attaching</td>
</tr>
<tr>
<td>VolumeId</td>
<td>vol-12345678</td>
</tr>
</tbody>
</table>

- For API details, see [AttachVolume](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Add-EC2VpnGateway**

The following code example shows how to use `Add-EC2VpnGateway`.

**Tools for PowerShell**

**Example 1:** This example attaches the specified virtual private gateway to the specified VPC.

```
Add-EC2VpnGateway -VpnGatewayId vgw-1a2b3c4d -VpcId vpc-12345678
```

Output:

<table>
<thead>
<tr>
<th>State</th>
<th>VpcId</th>
</tr>
</thead>
<tbody>
<tr>
<td>attaching</td>
<td>vpc-12345678</td>
</tr>
</tbody>
</table>

- For API details, see [AttachVpnGateway](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Approve-EC2VpcPeeringConnection**

The following code example shows how to use `Approve-EC2VpcPeeringConnection`.

**Tools for PowerShell**

**Example 1:** This example approves the requested VpcPeeringConnectionId pcx-1dfad234b56ff78be
Approve-EC2VpcPeeringConnection -VpcPeeringConnectionId pcx-1dfad234b56ff78be

**Output:**

AccepterVpcInfo : Amazon.EC2.Model.VpcPeeringConnectionVpcInfo
ExpirationTime   : 1/1/0001 12:00:00 AM
RequesterVpcInfo : Amazon.EC2.Model.VpcPeeringConnectionVpcInfo
Status           : Amazon.EC2.Model.VpcPeeringConnectionStateReason
Tags             : {}
VpcPeeringConnectionId : pcx-1dfad234b56ff78be

- For API details, see [AcceptVpcPeeringConnection](#) in *AWS Tools for PowerShell Cmdlet Reference*.

---

**Confirm-EC2ProductInstance**

The following code example shows how to use `Confirm-EC2ProductInstance`.

**Tools for PowerShell**

**Example 1:** This example determines whether the specified product code is associated with the specified instance.

```
Confirm-EC2ProductInstance -ProductCode 774F4FF8 -InstanceId i-12345678
```

- For API details, see [ConfirmProductInstance](#) in *AWS Tools for PowerShell Cmdlet Reference*.

---

**Copy-EC2Image**

The following code example shows how to use `Copy-EC2Image`.

**Tools for PowerShell**

**Example 1:** This example copies the specified AMI in the 'EU (Ireland)' region to the 'US West (Oregon)' region. If -Region is not specified, the current default region is used as the destination region.

```
Copy-EC2Image -SourceRegion eu-west-1 -SourceImageId ami-12345678 -Region us-west-2 -Name "Copy of ami-12345678"
```
Output:

ami-87654321

- For API details, see CopyImage in AWS Tools for PowerShell Cmdlet Reference.

Copy-EC2Snapshot

The following code example shows how to use Copy-EC2Snapshot.

Tools for PowerShell

Example 1: This example copies the specified snapshot from the EU (Ireland) region to the US West (Oregon) region.

Copy-EC2Snapshot -SourceRegion eu-west-1 -SourceSnapshotId snap-12345678 -Region us-west-2

Example 2: If you set a default region and omit the Region parameter, the default destination region is the default region.

Set-DefaultAWSRegion us-west-2
Copy-EC2Snapshot -SourceRegion eu-west-1 -SourceSnapshotId snap-12345678

- For API details, see CopySnapshot in AWS Tools for PowerShell Cmdlet Reference.

Deny-EC2VpcPeeringConnection

The following code example shows how to use Deny-EC2VpcPeeringConnection.

Tools for PowerShell

Example 1: The above example denies the request for VpcPeering request id pcx-01a2b3ce45fe67eb8

Deny-EC2VpcPeeringConnection -VpcPeeringConnectionId pcx-01a2b3ce45fe67eb8

- For API details, see RejectVpcPeeringConnection in AWS Tools for PowerShell Cmdlet Reference.
Disable-EC2VgwRoutePropagation

The following code example shows how to use Disable-EC2VgwRoutePropagation.

Tools for PowerShell

Example 1: This example disables the VGW from automatically propagating routes to the specified routing table.

```
Disable-EC2VgwRoutePropagation -RouteTableId rtb-12345678 -GatewayId vgw-1a2b3c4d
```

- For API details, see DisableVgwRoutePropagation in AWS Tools for PowerShell Cmdlet Reference.

Disable-EC2VpcClassicLink

The following code example shows how to use Disable-EC2VpcClassicLink.

Tools for PowerShell

Example 1: This example disables EC2VpcClassicLink for the vpc-01e23c4a5d6db78e9. It returns either True or False

```
Disable-EC2VpcClassicLink -VpcId vpc-01e23c4a5d6db78e9
```

- For API details, see DisableVpcClassicLink in AWS Tools for PowerShell Cmdlet Reference.

Disable-EC2VpcClassicLinkDnsSupport

The following code example shows how to use Disable-EC2VpcClassicLinkDnsSupport.

Tools for PowerShell

Example 1: This example disables ClassicLink DNS support for the vpc-0b12d3456a7e8910d

```
Disable-EC2VpcClassicLinkDnsSupport -VpcId vpc-0b12d3456a7e8910d
```

- For API details, see DisableVpcClassicLinkDnsSupport in AWS Tools for PowerShell Cmdlet Reference.
Dismount-EC2InternetGateway

The following code example shows how to use Dismount-EC2InternetGateway.

Tools for PowerShell

**Example 1:** This example detaches the specified Internet gateway from the specified VPC.

Dismount-EC2InternetGateway -InternetGatewayId igw-1a2b3c4d -VpcId vpc-12345678

- For API details, see [DetachInternetGateway](...) in *AWS Tools for PowerShell Cmdlet Reference*.

Dismount-EC2NetworkInterface

The following code example shows how to use Dismount-EC2NetworkInterface.

Tools for PowerShell

**Example 1:** This example removes the specified attachment between a network interface and an instance.

Dismount-EC2NetworkInterface -AttachmentId eni-attach-1a2b3c4d -Force

- For API details, see [DetachNetworkInterface](...) in *AWS Tools for PowerShell Cmdlet Reference*.

Dismount-EC2Volume

The following code example shows how to use Dismount-EC2Volume.

Tools for PowerShell

**Example 1:** This example detaches the specified volume.

Dismount-EC2Volume -VolumeId vol-12345678

**Output:**

<table>
<thead>
<tr>
<th>AttachTime</th>
<th>12/22/2015 1:53:58 AM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeleteOnTermination</td>
<td>False</td>
</tr>
<tr>
<td>Device</td>
<td>/dev/sdh</td>
</tr>
</tbody>
</table>
Example 2: You can also specify the instance ID and device name to ensure that you are detaching the correct volume.

Dismount-EC2Volume -VolumeId vol-12345678 -InstanceId i-1a2b3c4d -Device /dev/sdh

• For API details, see DetachVolume in AWS Tools for PowerShell Cmdlet Reference.

Dismount-EC2VpnGateway

The following code example shows how to use Dismount-EC2VpnGateway.

Tools for PowerShell

Example 1: This example detaches the specified virtual private gateway from the specified VPC.

Dismount-EC2VpnGateway -VpnGatewayId vgw-1a2b3c4d -VpcId vpc-12345678

• For API details, see DetachVpnGateway in AWS Tools for PowerShell Cmdlet Reference.

Edit-EC2CapacityReservation

The following code example shows how to use Edit-EC2CapacityReservation.

Tools for PowerShell

Example 1: This example modifies the CapacityReservationId cr-0c1f2345db6f7cdba by changing the instance count to 1

Edit-EC2CapacityReservation -CapacityReservationId cr-0c1f2345db6f7cdba -InstanceCount 1

Output:

True
Edit-EC2Host

The following code example shows how to use Edit-EC2Host.

Tools for PowerShell

Example 1: This example modifies the AutoPlacement settings to off for the dedicated host h-01e23f4cd567890f3

```
Edit-EC2Host -HostId h-03e09f8cd681609f3 -AutoPlacement off
```

Output:

```
Successful            Unsuccessful
----------            ------------
{h-01e23f4cd567890f3} {}
```

• For API details, see ModifyHosts in AWS Tools for PowerShell Cmdlet Reference.

Edit-EC2IdFormat

The following code example shows how to use Edit-EC2IdFormat.

Tools for PowerShell

Example 1: This example enables the longer ID format for the specified resource type.

```
Edit-EC2IdFormat -Resource instance -UseLongId $true
```

Example 2: This example disables the longer ID format for the specified resource type.

```
Edit-EC2IdFormat -Resource instance -UseLongId $false
```

• For API details, see ModifyIdFormat in AWS Tools for PowerShell Cmdlet Reference.

Edit-EC2ImageAttribute

The following code example shows how to use Edit-EC2ImageAttribute.
Example 1: This example updates the description for the specified AMI.

```
Edit-EC2ImageAttribute -ImageId ami-12345678 -Description "New description"
```

Example 2: This example makes the AMI public (for example, so any AWS account can use it).

```
Edit-EC2ImageAttribute -ImageId ami-12345678 -Attribute launchPermission -
OperationType add -UserGroup all
```

Example 3: This example makes the AMI private (for example, so that only you as the owner can use it).

```
Edit-EC2ImageAttribute -ImageId ami-12345678 -Attribute launchPermission -
OperationType remove -UserGroup all
```

Example 4: This example grants launch permission to the specified AWS account.

```
Edit-EC2ImageAttribute -ImageId ami-12345678 -Attribute launchPermission -
OperationType add -UserId 111122223333
```

Example 5: This example removes launch permission from the specified AWS account.

```
Edit-EC2ImageAttribute -ImageId ami-12345678 -Attribute launchPermission -
OperationType remove -UserId 111122223333
```

• For API details, see ModifyImageAttribute in AWS Tools for PowerShell Cmdlet Reference.

```
Edit-EC2InstanceAttribute
```

The following code example shows how to use Edit-EC2InstanceAttribute.

Tools for PowerShell

Example 1: This example modifies the instance type of the specified instance.

```
Edit-EC2InstanceAttribute -InstanceId i-12345678 -InstanceType m3.medium
```

Amazon EC2
Example 2: This example enables enhanced networking for the specified instance, by specifying "simple" as the value of the single root I/O virtualization (SR-IOV) network support parameter, -SriovNetSupport.

```
Edit-EC2InstanceAttribute -InstanceId i-12345678 -SriovNetSupport "simple"
```

Example 3: This example modifies the security groups for the specified instance. The instance must be in a VPC. You must specify the ID of each security group, not the name.

```
Edit-EC2InstanceAttribute -InstanceId i-12345678 -Group @("sg-12345678", "sg-45678901")
```

Example 4: This example enables EBS I/O optimization for the specified instance. This feature isn't available with all instance types. Additional usage charges apply when using an EBS-optimized instance.

```
Edit-EC2InstanceAttribute -InstanceId i-12345678 -EbsOptimized $true
```

Example 5: This example enables source/destination checking for the specified instance. For a NAT instance to perform NAT, the value must be 'false'.

```
Edit-EC2InstanceAttribute -InstanceId i-12345678 -SourceDestCheck $true
```

Example 6: This example disables termination for the specified instance.

```
Edit-EC2InstanceAttribute -InstanceId i-12345678 -DisableApiTermination $true
```

Example 7: This example changes the specified instance so that it terminates when shutdown is initiated from the instance.

```
Edit-EC2InstanceAttribute -InstanceId i-12345678 -InstanceInitiatedShutdownBehavior terminate
```

- For API details, see [ModifyInstanceAttribute](aws-docroot-sdkcontrols/ра/En/Services/AWSDocSet/165/awstoolsforcps/powershell/cmdlets/AWSToolsForPowerShell.CmdletReference.xml#ModifyInstanceAttribute) in *AWS Tools for PowerShell Cmdlet Reference*.

**Edit-EC2InstanceCreditSpecification**

The following code example shows how to use `Edit-EC2InstanceCreditSpecification`.
Tools for PowerShell

Example 1: This enables T2 unlimited credits for instance i-01234567890abcdef.

```powershell
$Credit = New-Object -TypeName Amazon.EC2.Model.InstanceCreditSpecificationRequest
$Credit.InstanceId = "i-01234567890abcdef"
$Credit.CpuCredits = "unlimited"
Edit-EC2InstanceCreditSpecification -InstanceCreditSpecification $Credit
```

- For API details, see [ModifyInstanceCreditSpecification](https://docs.aws.amazon.com/powershell/latest/reference/ModifyInstanceCreditSpecification) in *AWS Tools for PowerShell Cmdlet Reference*.

**Edit-EC2NetworkInterfaceAttribute**

The following code example shows how to use `Edit-EC2NetworkInterfaceAttribute`.

Tools for PowerShell

Example 1: This example modifies the specified network interface so that the specified attachment is deleted on termination.

```powershell
Edit-EC2NetworkInterfaceAttribute -NetworkInterfaceId eni-1a2b3c4d -Attachment_AttachmentId eni-attach-1a2b3c4d -Attachment_DeleteOnTermination $true
```

Example 2: This example modifies the description of the specified network interface.

```powershell
Edit-EC2NetworkInterfaceAttribute -NetworkInterfaceId eni-1a2b3c4d -Description "my description"
```

Example 3: This example modifies the security group for the specified network interface.

```powershell
Edit-EC2NetworkInterfaceAttribute -NetworkInterfaceId eni-1a2b3c4d -Groups sg-1a2b3c4d
```

Example 4: This example disables source/destination checking for the specified network interface.

```powershell
Edit-EC2NetworkInterfaceAttribute -NetworkInterfaceId eni-1a2b3c4d -SourceDestCheck $false
```
• For API details, see ModifyNetworkInterfaceAttribute in AWS Tools for PowerShell Cmdlet Reference.

Edit-EC2ReservedInstance

The following code example shows how to use Edit-EC2ReservedInstance.

Tools for PowerShell

Example 1: This example modifies the Availability Zone, instance count, and platform for the specified Reserved instances.

```powershell
$config = New-Object Amazon.EC2.Model.ReservedInstancesConfiguration
$config.AvailabilityZone = "us-west-2a"
$config.InstanceCount = 1
$config.Platform = "EC2-VPC"

Edit-EC2ReservedInstance
-ReservedInstancesId @("FE32132D-70D5-4795-B400-AE435EXAMPLE", "0CC556F3-7AB8-4C00-B0E5-98666EXAMPLE")
-TargetConfiguration $config
```

• For API details, see ModifyReservedInstances in AWS Tools for PowerShell Cmdlet Reference.

Edit-EC2SnapshotAttribute

The following code example shows how to use Edit-EC2SnapshotAttribute.

Tools for PowerShell

Example 1: This example makes the specified snapshot public by setting its CreateVolumePermission attribute.

```powershell
Edit-EC2SnapshotAttribute -SnapshotId snap-12345678 -Attribute CreateVolumePermission -OperationType Add -GroupName all
```

• For API details, see ModifySnapshotAttribute in AWS Tools for PowerShell Cmdlet Reference.

Edit-EC2SpotFleetRequest

The following code example shows how to use Edit-EC2SpotFleetRequest.
Tools for PowerShell

Example 1: This example updates the target capacity of the specified Spot fleet request.

```
Edit-EC2SpotFleetRequest -SpotFleetRequestId sfr-73fbd2ce-aa30-494c-8788-1cee4EXAMPLE -TargetCapacity 10
```

Output:

```
True
```

- For API details, see [ModifySpotFleetRequest](https://aws-tools-for-powershell-reference.md) in *AWS Tools for PowerShell Cmdlet Reference*.

**Edit-EC2SubnetAttribute**

The following code example shows how to use `Edit-EC2SubnetAttribute`.

Tools for PowerShell

Example 1: This example enables public IP addressing for the specified subnet.

```
Edit-EC2SubnetAttribute -SubnetId subnet-1a2b3c4d -MapPublicIpOnLaunch $true
```

Example 2: This example disables public IP addressing for the specified subnet.

```
Edit-EC2SubnetAttribute -SubnetId subnet-1a2b3c4d -MapPublicIpOnLaunch $false
```

- For API details, see [ModifySubnetAttribute](https://aws-tools-for-powershell-reference.md) in *AWS Tools for PowerShell Cmdlet Reference*.

**Edit-EC2VolumeAttribute**

The following code example shows how to use `Edit-EC2VolumeAttribute`.

Tools for PowerShell

Example 1: This example modifies the specified attribute of the specified volume. I/O operations for the volume are automatically resumed after being suspended due to potentially inconsistent data.

```
Edit-EC2VolumeAttribute -VolumeId vol-12345678 -AutoEnableIO $true
```
Edit-EC2VpcAttribute

The following code example shows how to use Edit-EC2VpcAttribute.

Tools for PowerShell

Example 1: This example enables support for DNS hostnames for the specified VPC.

```
Edit-EC2VpcAttribute -VpcId vpc-12345678 -EnableDnsHostnames $true
```

Example 2: This example disables support for DNS hostnames for the specified VPC.

```
Edit-EC2VpcAttribute -VpcId vpc-12345678 -EnableDnsHostnames $false
```

Example 3: This example enables support for DNS resolution for the specified VPC.

```
Edit-EC2VpcAttribute -VpcId vpc-12345678 -EnableDnsSupport $true
```

Example 4: This example disables support for DNS resolution for the specified VPC.

```
Edit-EC2VpcAttribute -VpcId vpc-12345678 -EnableDnsSupport $false
```

Enable-EC2VgwRoutePropagation

The following code example shows how to use Enable-EC2VgwRoutePropagation.

Tools for PowerShell

Example 1: This example enables the specified VGW to propagate routes automatically to the specified routing table.

```
Enable-EC2VgwRoutePropagation -RouteTableId rtb-12345678 -GatewayId vgw-1a2b3c4d
```

• For API details, see ModifyVolumeAttribute in AWS Tools for PowerShell Cmdlet Reference.

• For API details, see ModifyVpcAttribute in AWS Tools for PowerShell Cmdlet Reference.
Enable-EC2VolumeIO

The following code example shows how to use Enable-EC2VolumeIO.

Tools for PowerShell

   Example 1: This example enables I/O operations for the specified volume, if I/O operations were disabled.

   Enable-EC2VolumeIO -VolumeId vol-12345678

   For API details, see EnableVolumeIo in AWS Tools for PowerShell Cmdlet Reference.

Enable-EC2VpcClassicLink

The following code example shows how to use Enable-EC2VpcClassicLink.

Tools for PowerShell

   Example 1: This example enables VPC vpc-0123456b789b0d12f for ClassicLink

   Enable-EC2VpcClassicLink -VpcId vpc-0123456b789b0d12f

   Output:

   True

   For API details, see EnableVpcClassicLink in AWS Tools for PowerShell Cmdlet Reference.

Enable-EC2VpcClassicLinkDnsSupport

The following code example shows how to use Enable-EC2VpcClassicLinkDnsSupport.

Tools for PowerShell

   Example 1: This example enables vpc-0b12d3456a7e8910d to support DNS hostname resolution for ClassicLink

   Enable-EC2VpcClassicLinkDnsSupport -VpcId vpc-0b12d3456a7e8910d -Region eu-west-1
For API details, see `EnableVpcClassicLinkDnsSupport` in `AWS Tools for PowerShell Cmdlet Reference`.

### Get-EC2AccountAttribute

The following code example shows how to use Get-EC2AccountAttribute.

#### Tools for PowerShell

**Example 1:** This example describes whether you can launch instances into EC2-Classics and EC2-VPC in the region, or only into EC2-VPC.

```powershell
(Get-EC2AccountAttribute -AttributeName supported-platforms).AttributeValues
```

**Output:**

```
AttributeValue
-------------
EC2
VPC
```

**Example 2:** This example describes your default VPC, or 'none' if you do not have a default VPC in the region.

```powershell
(Get-EC2AccountAttribute -AttributeName default-vpc).AttributeValues
```

**Output:**

```
AttributeValue
-------------
vpc-12345678
```

**Example 3:** This example describes the maximum number of On-Demand instances that you can run.

```powershell
(Get-EC2AccountAttribute -AttributeName max-instances).AttributeValues
```

**Output:**
For API details, see `DescribeAccountAttributes` in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-EC2Address

The following code example shows how to use `Get-EC2Address`.

#### Tools for PowerShell

**Example 1:** This example describes the specified Elastic IP address for instances in EC2-Classic.

```powershell
Get-EC2Address -AllocationId eipalloc-12345678
```

**Output:**

```
AllocationId            : eipalloc-12345678
AssociationId           : eipassoc-12345678
Domain                  : vpc
InstanceId              : i-87654321
NetworkInterfaceId      : eni-12345678
NetworkInterfaceOwnerId : 12345678
PrivateIpAddress        : 10.0.2.172
PublicIp                : 198.51.100.2
```

**Example 2:** This example describes your Elastic IP addresses for instances in a VPC. This syntax requires PowerShell version 3 or later.

```powershell
Get-EC2Address -Filter @{'Name'='domain';'Values'='vpc'}
```

**Example 3:** This example describes the specified Elastic IP address for instances in EC2-Classic.

```powershell
Get-EC2Address -PublicIp 203.0.113.17
```

**Output:**
Example 4: This example describes your Elastic IP addresses for instances in EC2-Classic. This syntax requires PowerShell version 3 or later.

Get-EC2Address -Filter @{ Name="domain"; Values="standard" }
Example 8: This example fetches list of EC2 IP addresses matching tag key 'Category' with value 'Prod'

Get-EC2Address -Filter @{Name="tag:Category";Values="Prod"}

Output:

<table>
<thead>
<tr>
<th>AllocationId</th>
<th>eipalloc-0123f456f81a01b58</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssociationId</td>
<td>eipassoc-0d1b23a456d103810</td>
</tr>
<tr>
<td>CustomerOwnedIp</td>
<td></td>
</tr>
<tr>
<td>CustomerOwnedIpv4Pool</td>
<td></td>
</tr>
<tr>
<td>Domain</td>
<td>vpc</td>
</tr>
<tr>
<td>InstanceId</td>
<td>i-0123cb4df567e1aa</td>
</tr>
<tr>
<td>NetworkBorderGroup</td>
<td>eu-west-1</td>
</tr>
<tr>
<td>NetworkInterfaceId</td>
<td>eni-0123f41d5a60d5f40</td>
</tr>
<tr>
<td>NetworkInterfaceOwnerId</td>
<td>123456789012</td>
</tr>
<tr>
<td>PrivateIpAddress</td>
<td>192.168.1.84</td>
</tr>
<tr>
<td>PublicIp</td>
<td>34.250.81.29</td>
</tr>
<tr>
<td>PublicIpv4Pool</td>
<td>amazon</td>
</tr>
<tr>
<td>Tags</td>
<td>[Category, Name]</td>
</tr>
</tbody>
</table>

- For API details, see [DescribeAddresses](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-EC2AvailabilityZone**

The following code example shows how to use Get-EC2AvailabilityZone.
Example 1: This example describes the Availability Zones for the current region that are available to you.

```
Get-EC2AvailabilityZone
```

**Output:**

<table>
<thead>
<tr>
<th>Messages</th>
<th>RegionName</th>
<th>State</th>
<th>ZoneName</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>us-west-2</td>
<td>available</td>
<td>us-west-2a</td>
</tr>
<tr>
<td></td>
<td>us-west-2</td>
<td>available</td>
<td>us-west-2b</td>
</tr>
<tr>
<td></td>
<td>us-west-2</td>
<td>available</td>
<td>us-west-2c</td>
</tr>
</tbody>
</table>

Example 2: This example describes any Availability Zones that are in an impaired state. The syntax used by this example requires PowerShell version 3 or higher.

```
Get-EC2AvailabilityZone -Filter @{ Name="state";Values="impaired" }
```

Example 3: With PowerShell version 2, you must use `New-Object` to create the filter.

```powershell
$filter = New-Object Amazon.EC2.Model.Filter
$filter.Name = "state"
$filter.Values = "impaired"

Get-EC2AvailabilityZone -Filter $filter
```


---

## Get-EC2BundleTask

The following code example shows how to use `Get-EC2BundleTask`.

**Example 1: This example describes the specified bundle task.**

```
Get-EC2BundleTask -BundleId bun-12345678
```
Example 2: This example describes the bundle tasks whose state is either 'complete' or 'failed'.

```powershell
$filter = New-Object Amazon.EC2.Model.Filter
$filter.Name = "state"
$filter.Values = @( "complete", "failed" )

Get-EC2BundleTask -Filter $filter
```

- For API details, see [DescribeBundleTasks](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

Get-EC2CapacityReservation

The following code example shows how to use Get-EC2CapacityReservation.

Tools for PowerShell

Example 1: This example describes one or more of your Capacity Reservations for the region

```powershell
Get-EC2CapacityReservation -Region eu-west-1
```

Output:

```plaintext
AvailabilityZone       : eu-west-1b
AvailableInstanceCount : 2
CapacityReservationId  : cr-0c1f2345db6f7c6ba
CreateDate             : 3/28/2019 9:29:41 AM
EbsOptimized           : True
EndDate                : 1/1/0001 12:00:00 AM
EndDateType            : unlimited
EphemeralStorage       : False
InstanceMatchCriteria  : open
InstancePlatform       : Windows
InstanceType           : m4.xlarge
State                  : active
Tags                   : {}
Tenancy                : default
TotalInstanceCount     : 2
```

- For API details, see [DescribeCapacityReservations](#) in [AWS Tools for PowerShell Cmdlet Reference](#).
Get-EC2ConsoleOutput

The following code example shows how to use Get-EC2ConsoleOutput.

Tools for PowerShell

Example 1: This example gets the console output for the specified Linux instance. The console output is encoded.

```powershell
Get-EC2ConsoleOutput -InstanceId i-0e19abcd47c123456
```

Output:

```
InstanceId          Output
----------          ------
i-0e194d3c47c123637 WyAgICAwLjAwMDAwMF0gQ29tbW...bGU9dHR5UzAgc2Vs
```

Example 2: This example stores the encoded console output in a variable and then decodes it.

```powershell
$Output_encoded = (Get-EC2ConsoleOutput -InstanceId i-0e19abcd47c123456).Output
[System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($Output_encoded))
```

- For API details, see `GetConsoleOutput` in AWS Tools for PowerShell Cmdlet Reference.

Get-EC2CustomerGateway

The following code example shows how to use Get-EC2CustomerGateway.

Tools for PowerShell

Example 1: This example describes the specified customer gateway.

```powershell
Get-EC2CustomerGateway -CustomerGatewayId cgw-1a2b3c4d
```

Output:

```
BgpAsn : 65534
CustomerGatewayId : cgw-1a2b3c4d
IpAddress : 203.0.113.12
State : available
```
Example 2: This example describes any customer gateway whose state is either pending or available.

```powershell
$filter = New-Object Amazon.EC2.Model.Filter
$filter.Name = "state"
$filter.Values = @("pending", "available")

Get-EC2CustomerGateway -Filter $filter
```

Example 3: This example describes all your customer gateways.

```
Get-EC2CustomerGateway
```

- For API details, see [DescribeCustomerGateways](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-EC2DhcpOption

The following code example shows how to use `Get-EC2DhcpOption`.

**Tools for PowerShell**

**Example 1: This example lists your DHCP options sets.**

```
Get-EC2DhcpOption
```

**Output:**

```
DhcpConfigurations          DhcpOptionsId    Tag
------------------            -------------    ---
{domain-name, domain-name-servers} dopt-1a2b3c4d    {}
{domain-name, domain-name-servers} dopt-2a3b4c5d    {}
{domain-name-servers}         dopt-3a4b5c6d    {}
```

**Example 2: This example gets configuration details for the specified DHCP options set.**

```
(Get-EC2DhcpOption -DhcpOptionsId dopt-1a2b3c4d).DhcpConfigurations
```

**Output:**
For API details, see [DescribeDhcpOptions](#) in *AWS Tools for PowerShell Cmdlet Reference.*

### Get-EC2FlowLog

The following code example shows how to use `Get-EC2FlowLog`.

**Tools for PowerShell**

**Example 1:** This example describes one or more flow logs with log destination type 's3'

```powershell
Get-EC2FlowLog -Filter @{Name="log-destination-type";Values="s3"}
```

**Output:**

```
CreationTime             : 2/25/2019 9:07:36 PM
DeliverLogsErrorMessage : 
DeliverLogsPermissionArn:
DeliverLogsStatus        : SUCCESS
FlowLogId                : fl-01b2e3d45f67f8901
FlowLogStatus            : ACTIVE
LogDestination           : arn:aws:s3:::my-bucket-dd-tata
LogDestinationType       : s3
LogGroupName             :
ResourceId               : eni-01d2dda3456b7e890
TrafficType              : ALL
```

For API details, see [DescribeFlowLogs](#) in *AWS Tools for PowerShell Cmdlet Reference.*

### Get-EC2Host

The following code example shows how to use `Get-EC2Host`.

**Tools for PowerShell**

**Example 1:** This example returns the EC2 host details
Get-EC2Host

Output:

```
AutoPlacement     : off
AvailabilityZone  : eu-west-1b
ClientToken       :
HostId            : h-01e23f4cd567890f1
HostProperties    : Amazon.EC2.Model.HostProperties
HostReservationId :
Instances         : {}
ReleaseTime       : 1/1/0001 12:00:00 AM
State             : available
Tags              : {}
```

Example 2: This example queries the AvailableInstanceCapacity for the host h-01e23f4cd567899f1

```
Get-EC2Host -HostId h-01e23f4cd567899f1 | Select-Object -ExpandProperty
  AvailableCapacity | Select-Object -expand AvailableInstanceCapacity
```

Output:

```
AvailableCapacity InstanceType TotalCapacity
----------------- ------------ ------------
11                m4.xlarge    11
```

- For API details, see [DescribeHosts](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-EC2HostReservationOffering

The following code example shows how to use Get-EC2HostReservationOffering.

Tools for PowerShell

Example 1: This example describes the Dedicated Host reservations that are available to purchase for the given filter 'instance-family' where PaymentOption is 'NoUpfront'
Get-EC2HostReservationOffering -Filter @{Name="instance-family";Values="m4"} | Where-Object PaymentOption -eq NoUpfront

**Output:**

<table>
<thead>
<tr>
<th>CurrencyCode</th>
<th>Duration</th>
<th>HourlyPrice</th>
<th>InstanceFamily</th>
<th>OfferingId</th>
<th>PaymentOption</th>
<th>UpfrontPrice</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>94608000</td>
<td>1.307</td>
<td>m4</td>
<td>hro-0c1f234567890d9ab</td>
<td>NoUpfront</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>31536000</td>
<td>1.830</td>
<td>m4</td>
<td>hro-04ad12aaaf34b5a67</td>
<td>NoUpfront</td>
<td>0.000</td>
</tr>
</tbody>
</table>

- For API details, see [DescribeHostReservationOfferings](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

**Get-EC2HostReservationPurchasePreview**

The following code example shows how to use Get-EC2HostReservationPurchasePreview.

**Tools for PowerShell**

**Example 1:** This example previews a reservation purchase with configurations that match those of your Dedicated Host h-01e23f4cd567890f1

Get-EC2HostReservationPurchasePreview -OfferingId hro-0c1f23456789d0ab -HostIdSet h-01e23f4cd567890f1

**Output:**

<table>
<thead>
<tr>
<th>CurrencyCode</th>
<th>Purchase</th>
<th>TotalHourlyPrice</th>
<th>TotalUpfrontPrice</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
For API details, see [GetHostReservationPurchasePreview](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

**Get-EC2IdFormat**

The following code example shows how to use `Get-EC2IdFormat`.

**Tools for PowerShell**

**Example 1:** This example describes the ID format for the specified resource type.

```
Get-EC2IdFormat -Resource instance
```

**Output:**

<table>
<thead>
<tr>
<th>Resource</th>
<th>UseLongIds</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance</td>
<td>False</td>
</tr>
</tbody>
</table>

**Example 2:** This example describes the ID formats for all resource types that support longer IDs.

```
Get-EC2IdFormat
```

**Output:**

<table>
<thead>
<tr>
<th>Resource</th>
<th>UseLongIds</th>
</tr>
</thead>
<tbody>
<tr>
<td>reservation</td>
<td>False</td>
</tr>
<tr>
<td>instance</td>
<td>False</td>
</tr>
</tbody>
</table>

For API details, see [DescribeIdFormat](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

**Get-EC2IdentityIdFormat**

The following code example shows how to use `Get-EC2IdentityIdFormat`.

---

Amazon EC2
Example 1: This example returns the ID format for the resource 'image' for the role given

```
Get-EC2IdentityIdFormat -PrincipalArn arn:aws:iam::123456789511:role/JDBC -Resource image
```

Output:

<table>
<thead>
<tr>
<th>Deadline</th>
<th>Resource</th>
<th>UseLongIds</th>
</tr>
</thead>
<tbody>
<tr>
<td>8/2/2018 11:30:00 PM</td>
<td>image</td>
<td>True</td>
</tr>
</tbody>
</table>

- For API details, see [DescribeIdentityIdFormat](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

Get-EC2Image

The following code example shows how to use Get-EC2Image.

**Example 1: This example describes the specified AMI.**

```
Get-EC2Image -ImageId ami-12345678
```

Output:

```
Architecture        : x86_64
BlockDeviceMappings : {/dev/xvda}
CreationDate        : 2014-10-20T00:56:28.000Z
Description         : My image
Hypervisor          : xen
ImageId             : ami-12345678
ImageLocation       : 123456789012/my-image
ImageOwnerAlias     : 
ImageType           : machine
KernelId            : 
Name                : my-image
OwnerId             : 123456789012
Platform            : 
ProductCodes        : {}
```
<table>
<thead>
<tr>
<th>Public</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>RamdiskId</td>
<td></td>
</tr>
<tr>
<td>RootDeviceName</td>
<td>/dev/xvda</td>
</tr>
<tr>
<td>RootDeviceType</td>
<td>ebs</td>
</tr>
<tr>
<td>SriovNetSupport</td>
<td>simple</td>
</tr>
<tr>
<td>State</td>
<td>available</td>
</tr>
<tr>
<td>StateReason</td>
<td></td>
</tr>
<tr>
<td>Tags</td>
<td>{Name}</td>
</tr>
<tr>
<td>VirtualizationType</td>
<td>hvm</td>
</tr>
</tbody>
</table>

**Example 2: This example describes the AMIs that you own.**

```powershell
Get-EC2Image -owner self
```

**Example 3: This example describes the public AMIs that run Microsoft Windows Server.**

```powershell
Get-EC2Image -Filter @{ Name="platform"; Values="windows" }`n
```

**Example 4: This example describes all public AMIs in the 'us-west-2' region.**

```powershell
Get-EC2Image -Region us-west-2
```

- For API details, see [DescribeImages](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-EC2ImageAttribute**

The following code example shows how to use Get-EC2ImageAttribute.

**Tools for PowerShell**

**Example 1: This example gets the description for the specified AMI.**

```powershell
Get-EC2ImageAttribute -ImageId ami-12345678 -Attribute description
```

**Output:**

```powershell
BlockDeviceMappings : {}
Description         : My image description
ImageId             : ami-12345678
KernelId            : 
```
Example 2: This example gets the launch permissions for the specified AMI.

Get-EC2ImageAttribute -ImageId ami-12345678 -Attribute launchPermission

Output:

- BlockDeviceMappings : {}
- Description : 
- ImageId : ami-12345678
- KernelId : 
- LaunchPermissions : {all}
- ProductCodes : {}
- RamdiskId : 
- SriovNetSupport : 

Example 3: This example tests whether enhanced networking is enabled.

Get-EC2ImageAttribute -ImageId ami-12345678 -Attribute sriovNetSupport

Output:

- BlockDeviceMappings : {}
- Description : 
- ImageId : ami-12345678
- KernelId : 
- LaunchPermissions : {}
- ProductCodes : {}
- RamdiskId : 
- SriovNetSupport : simple

- For API details, see DescribeImageAttribute in AWS Tools for PowerShell Cmdlet Reference.

Get-EC2ImageByName

The following code example shows how to use Get-EC2ImageByName.
Example 1: This example describes the complete set of filter names that are currently supported.

Get-EC2ImageByName

Output:

- WINDOWS_2016_BASE
- WINDOWS_2016_NANO
- WINDOWS_2016_CORE
- WINDOWS_2016_CONTAINER
- WINDOWS_2016_SQL_SERVER_ENTERPRISE_2016
- WINDOWS_2016_SQL_SERVER_STANDARD_2016
- WINDOWS_2016_SQL_SERVER_WEB_2016
- WINDOWS_2016_SQL_SERVER_EXPRESS_2016
- WINDOWS_2012R2_BASE
- WINDOWS_2012R2_CORE
- WINDOWS_2012R2_SQL_SERVER_EXPRESS_2016
- WINDOWS_2012R2_SQL_SERVER_STANDARD_2016
- WINDOWS_2012R2_SQL_SERVER_WEB_2016
- 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
### Example 2: This example describes the specified AMI.
Using this command to locate an AMI is helpful because AWS releases new Windows AMIs with the latest updates each month. You can specify the 'ImageId' to New-EC2Instance to launch an instance using the current AMI for the specified filter.

```
Get-EC2ImageByName -Names WINDOWS_2016_BASE
```

#### Output:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86_64</td>
</tr>
<tr>
<td>BlockDeviceMappings</td>
<td>/dev/sda1, xvdca, xvdcb, xvdcc...</td>
</tr>
<tr>
<td>CreationDate</td>
<td>yyyy.mm.ddThh:mm:ss.000Z</td>
</tr>
<tr>
<td>Description</td>
<td>Microsoft Windows Server 2016 with Desktop Experience Locale English AMI provided by Amazon</td>
</tr>
<tr>
<td>Hypervisor</td>
<td>xen</td>
</tr>
<tr>
<td>ImageId</td>
<td>ami-xxxxxxxxx</td>
</tr>
<tr>
<td>ImageLocation</td>
<td>amazon/Windows_Server-2016-English-Full-Base-yyyy.mm.dd</td>
</tr>
<tr>
<td>ImageOwnerAlias</td>
<td>amazon</td>
</tr>
<tr>
<td>ImageType</td>
<td>machine</td>
</tr>
<tr>
<td>KernelId</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Windows_Server-2016-English-Full-Base-yyyy.mm.dd</td>
</tr>
<tr>
<td>OwnerId</td>
<td>801119661308</td>
</tr>
<tr>
<td>Platform</td>
<td>Windows</td>
</tr>
<tr>
<td>ProductCodes</td>
<td>{}</td>
</tr>
<tr>
<td>Public</td>
<td>True</td>
</tr>
<tr>
<td>RamdiskId</td>
<td></td>
</tr>
<tr>
<td>RootDeviceName</td>
<td>/dev/sda1</td>
</tr>
<tr>
<td>RootDeviceType</td>
<td>ebs</td>
</tr>
<tr>
<td>SriovNetSupport</td>
<td>simple</td>
</tr>
<tr>
<td>State</td>
<td>available</td>
</tr>
<tr>
<td>StateReason</td>
<td></td>
</tr>
<tr>
<td>Tags</td>
<td>{}</td>
</tr>
<tr>
<td>VirtualizationType</td>
<td>hvm</td>
</tr>
</tbody>
</table>

- For API details, see [Get-EC2ImageByName](https://docs.aws.amazon.com/powershell/latest/userguide/aws-tools-reference-getec2imagebyname.html) in *AWS Tools for PowerShell Cmdlet Reference*.  


Get-EC2ImportImageTask

The following code example shows how to use Get-EC2ImportImageTask.

Tools for PowerShell

Example 1: This example describes the specified image import task.

Get-EC2ImportImageTask -ImportTaskId import-ami-hgfedcba

Output:

Architecture : x86_64
Description    : Windows Image 2
Hypervisor     :
ImageId        : ami-1a2b3c4d
ImportTaskId   : import-ami-hgfedcba
LicenseType    : AWS
Platform       : Windows
Progress       :
SnapshotDetails : {/dev/sda1}
Status         : completed
StatusMessage  :

Example 2: This example describes all your image import tasks.

Get-EC2ImportImageTask

Output:

Architecture   :
Description    : Windows Image 1
Hypervisor     :
ImageId        :
ImportTaskId   : import-ami-abcddefgh
LicenseType    : AWS
Platform       : Windows
Progress       :
SnapshotDetails : {}
Status         : deleted
StatusMessage  : User initiated task cancelation
For API details, see [DescribeImportImageTasks](https://docs.aws.amazon.com/powershell/latest/ug/commands-reference.html) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-EC2ImportSnapshotTask**

The following code example shows how to use `Get-EC2ImportSnapshotTask`.

### Tools for PowerShell

**Example 1:** This example describes the specified snapshot import task.

```powershell
Get-EC2ImportSnapshotTask -ImportTaskId import-snap-abcdefgh
```

**Output:**

<table>
<thead>
<tr>
<th>Description</th>
<th>ImportTaskId</th>
<th>SnapshotTaskDetail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk Image Import 1</td>
<td>import-snap-abcdefgh</td>
<td>Amazon.EC2.Model.SnapshotTaskDetail</td>
</tr>
</tbody>
</table>

**Example 2:** This example describes all your snapshot import tasks.

```powershell
Get-EC2ImportSnapshotTask
```

**Output:**

<table>
<thead>
<tr>
<th>Description</th>
<th>ImportTaskId</th>
<th>SnapshotTaskDetail</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Disk Image Import 1     import-snap-abcdefgh
Amazon.EC2.Model.SnapshotTaskDetail
Disk Image Import 2     import-snap-hgfedcba
Amazon.EC2.Model.SnapshotTaskDetail

For API details, see [DescribeImportSnapshotTasks](https://docs.aws.amazon.com/powershell/latest/reference/ps/aws-ec2-commands.html) in **AWS Tools for PowerShell Cmdlet Reference**.

### Get-EC2Instance

The following code example shows how to use Get-EC2Instance.

**Tools for PowerShell**

**Example 1: This example describes the specified instance.**

```
(Get-EC2Instance -InstanceId i-12345678).Instances
```

**Output:**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AmiLaunchIndex</td>
<td>0</td>
</tr>
<tr>
<td>Architecture</td>
<td>x86_64</td>
</tr>
<tr>
<td>BlockDeviceMappings</td>
<td>/dev/sda1</td>
</tr>
<tr>
<td>ClientToken</td>
<td>TleEy1448154045270</td>
</tr>
<tr>
<td>EbsOptimized</td>
<td>False</td>
</tr>
<tr>
<td>Hypervisor</td>
<td>xen</td>
</tr>
<tr>
<td>IamInstanceProfile</td>
<td>Amazon.EC2.Model.IamInstanceProfile</td>
</tr>
<tr>
<td>ImageId</td>
<td>ami-12345678</td>
</tr>
<tr>
<td>InstanceId</td>
<td>i-12345678</td>
</tr>
<tr>
<td>InstanceLifecycle</td>
<td></td>
</tr>
<tr>
<td>InstanceType</td>
<td>t2.micro</td>
</tr>
<tr>
<td>KernelId</td>
<td></td>
</tr>
<tr>
<td>KeyName</td>
<td>my-key-pair</td>
</tr>
<tr>
<td>LaunchTime</td>
<td>12/4/2015 4:44:40 PM</td>
</tr>
<tr>
<td>Monitoring</td>
<td>Amazon.EC2.Model.Monitoring</td>
</tr>
<tr>
<td>NetworkInterfaces</td>
<td>ip-10-0-2-172.us-west-2.compute.internal</td>
</tr>
<tr>
<td>Placement</td>
<td>Amazon.EC2.Model.Placement</td>
</tr>
<tr>
<td>Platform</td>
<td>Windows</td>
</tr>
<tr>
<td>PrivateDnsName</td>
<td>ip-10-0-2-172.us-west-2.compute.internal</td>
</tr>
<tr>
<td>PrivateIpAddress</td>
<td>10.0.2.172</td>
</tr>
</tbody>
</table>
Example 2: This example describes all your instances in the current region, grouped by reservation. To see the instance details expand the Instances collection within each reservation object.

Get-EC2Instance

Output:

GroupNames : {}
Groups : {}
Instances : {}
OwnerId : 123456789012
RequesterId : 226008221399
ReservationId : r-c5df370c

GroupNames : {}
Groups : {}
Instances : {}
OwnerId : 123456789012
RequesterId : 854251627541
ReservationId : r-63e65bab
Example 3: This example illustrates using a filter to query for EC2 instances in a specific subnet of a VPC.

```
(Get-EC2Instance -Filter @{Name="vpc-id";Values="vpc-1a2bc34d"}, @{Name="subnet-id";Values="subnet-1a2b3c4d"}).Instances
```

Output:

<table>
<thead>
<tr>
<th>InstanceId</th>
<th>InstanceType</th>
<th>Platform</th>
<th>PrivateIpAddress</th>
<th>PublicIpAddress</th>
</tr>
</thead>
<tbody>
<tr>
<td>i-01af...82cf180e19</td>
<td>t2.medium</td>
<td>Windows</td>
<td>10.0.0.98</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>subnet-1a2b3c4d</td>
<td>vpc-1a2b3c4d</td>
</tr>
<tr>
<td>i-0374...7e9d5b0c45</td>
<td>t2.xlarge</td>
<td>Windows</td>
<td>10.0.0.53</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>subnet-1a2b3c4d</td>
<td>vpc-1a2b3c4d</td>
</tr>
</tbody>
</table>

- For API details, see [DescribeInstances](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-EC2InstanceAttribute

The following code example shows how to use Get-EC2InstanceAttribute.

**Tools for PowerShell**

Example 1: This example describes the instance type of the specified instance.

```
Get-EC2InstanceAttribute -InstanceId i-12345678 -Attribute instanceType
```

Output:

```
InstanceId       : i-12345678
InstanceType     : t2.micro
```

Example 2: This example describes whether enhanced networking is enabled for the specified instance.

```
Get-EC2InstanceAttribute -InstanceId i-12345678 -Attribute sriovNetSupport
```

Output:
SriovNetSupport : simple

Example 3: This example describes the security groups for the specified instance.

(Get-EC2InstanceAttribute -InstanceId i-12345678 -Attribute groupSet).Groups

Output:

GroupId
-------
sg-12345678
sg-45678901

Example 4: This example describes whether EBS optimization is enabled for the specified instance.

Get-EC2InstanceAttribute -InstanceId i-12345678 -Attribute ebsOptimized

Output:

EbsOptimized : False

Example 5: This example describes the 'disableApiTermination' attribute of the specified instance.

Get-EC2InstanceAttribute -InstanceId i-12345678 -Attribute disableApiTermination

Output:

DisableApiTermination : False

Example 6: This example describes the 'instanceInitiatedShutdownBehavior' attribute of the specified instance.

Get-EC2InstanceAttribute -InstanceId i-12345678 -Attribute instanceInitiatedShutdownBehavior
Output:

InstanceInitiatedShutdownBehavior : stop

- For API details, see [DescribeInstanceAttribute](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-EC2InstanceMetadata**

The following code example shows how to use Get-EC2InstanceMetadata.

**Tools for PowerShell**

**Example 1: Lists the available categories of instance metadata that can be queried.**

```
Get-EC2InstanceMetadata -ListCategory
```

**Output:**

AmiId
LaunchIndex
ManifestPath
AncestorAmiId
BlockDeviceMapping
InstanceId
InstanceType
LocalHostname
LocalIpv4
KernelId
AvailabilityZone
ProductCode
PublicHostname
PublicIpv4
PublicKey
RamdiskId
Region
ReservationId
SecurityGroup
UserData
InstanceMonitoring
IdentityDocument
IdentitySignature
Example 2: Returns the id of the Amazon Machine Image (AMI) that was used to launch the instance.

Get-EC2InstanceMetadata -Category AmiId

Output:

ami-b2e756ca

Example 3: This example queries the JSON-formatted identity document for the instance.

Get-EC2InstanceMetadata -Category IdentityDocument

{  
  "availabilityZone" : "us-west-2a",
  "devpayProductCodes" : null,
  "marketplaceProductCodes" : null,
  "version" : "2017-09-30",
  "instanceId" : "i-01ed50f7e2607f09e",
  "billingProducts" : [ "bp-6ba54002" ],
  "instanceType" : "t2.small",
  "pendingTime" : "2018-03-07T16:26:04Z",
  "imageId" : "ami-b2e756ca",
  "privateIp" : "10.0.0.171",
  "accountId" : "111122223333",
  "architecture" : "x86_64",
  "kernelId" : null,
  "ramdiskId" : null,
  "region" : "us-west-2"
}

Example 4: This example uses a path query to obtain the network interface macs for the instance.

Get-EC2InstanceMetadata -Path "/network/interfaces/macs"

Output:

02:80:7f:ef:4c:e0/
Example 5: If there is an IAM role associated with the instance, returns information about the last time the instance profile was updated, including the instance's LastUpdated date, InstanceProfileArn, and InstanceProfileId.

Get-EC2InstanceMetadata -Path "/iam/info"

Output:

```
{
    "Code" : "Success",
    "LastUpdated" : "2018-03-08T03:38:40Z",
    "InstanceProfileArn" : "arn:aws:iam::111122223333:instance-profile/MyLaunchRole_Profile",
    "InstanceProfileId" : "AIPAI4...WVK2RW"
}
```

- For API details, see Get-EC2InstanceMetadata in AWS Tools for PowerShell Cmdlet Reference.

Get-EC2InstanceStatus

The following code example shows how to use Get-EC2InstanceStatus.

Tools for PowerShell

Example 1: This example describes the status of the specified instance.

Get-EC2InstanceStatus -InstanceId i-12345678

Output:

```
AvailabilityZone : us-west-2a
Events : {}
InstanceId : i-12345678
InstanceState : Amazon.EC2.Model.InstanceState
Status : Amazon.EC2.Model.InstanceStatusSummary
SystemStatus : Amazon.EC2.Model.InstanceStatusSummary
```

$status = Get-EC2InstanceStatus -InstanceId i-12345678
$status.InstanceState
Output:

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>running</td>
</tr>
</tbody>
</table>

```powershell
$status.Status
```

Output:

<table>
<thead>
<tr>
<th>Details</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>{reachability}</td>
<td>ok</td>
</tr>
</tbody>
</table>

```powershell
$status.SystemStatus
```

Output:

<table>
<thead>
<tr>
<th>Details</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>{reachability}</td>
<td>ok</td>
</tr>
</tbody>
</table>

- For API details, see [DescribeInstanceStatus](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-EC2InternetGateway**

The following code example shows how to use `Get-EC2InternetGateway`.

**Tools for PowerShell**

**Example 1: This example describes the specified Internet gateway.**

```powershell
Get-EC2InternetGateway -InternetGatewayId igw-1a2b3c4d
```

Output:

<table>
<thead>
<tr>
<th>Attachments</th>
<th>InternetGatewayId</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>{vpc-1a2b3c4d}</td>
<td>igw-1a2b3c4d</td>
<td>{}</td>
</tr>
</tbody>
</table>
Example 2: This example describes all your Internet gateways.

Get-EC2InternetGateway

Output:

<table>
<thead>
<tr>
<th>Attachments</th>
<th>InternetGatewayId</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>{vpc-1a2b3c4d}</td>
<td>igw-1a2b3c4d</td>
<td>{}</td>
</tr>
<tr>
<td>{}</td>
<td>igw-2a3b4c5d</td>
<td>{}</td>
</tr>
</tbody>
</table>

• For API details, see [DescribeInternetGateways](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-EC2KeyPair

The following code example shows how to use Get-EC2KeyPair.

Tools for PowerShell

Example 1: This example describes the specified key pair.

Get-EC2KeyPair -KeyName my-key-pair

Output:

<table>
<thead>
<tr>
<th>KeyFingerprint</th>
<th>KeyName</th>
</tr>
</thead>
</table>

Example 2: This example describes all your key pairs.

Get-EC2KeyPair

• For API details, see [DescribeKeyPairs](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-EC2NetworkAcl

The following code example shows how to use Get-EC2NetworkAcl.
Tools for PowerShell

Example 1: This example describes the specified network ACL.

Get-EC2NetworkAcl -NetworkAclId acl-12345678

Output:

Associations : {aclassoc-1a2b3c4d}
IsDefault    : False
NetworkAclId : acl-12345678
Tags         : {Name}
VpcId        : vpc-12345678

Example 2: This example describes the rules for the specified network ACL.

(Get-EC2NetworkAcl -NetworkAclId acl-12345678).Entries

Output:

CidrBlock    : 0.0.0.0/0
Egress       : True
IcmpTypeCode :
PortRange    :
Protocol     : -1
RuleAction   : deny
RuleNumber   : 32767

CidrBlock    : 0.0.0.0/0
Egress       : False
IcmpTypeCode :
PortRange    :
Protocol     : -1
RuleAction   : deny
RuleNumber   : 32767

Example 3: This example describes all your network ACLs.

Get-EC2NetworkAcl

For API details, see DescribeNetworkAcls in AWS Tools for PowerShell Cmdlet Reference.
Get-EC2NetworkInterface

The following code example shows how to use Get-EC2NetworkInterface.

Tools for PowerShell

Example 1: This example describes the specified network interface.

Get-EC2NetworkInterface -NetworkInterfaceId eni-12345678

Output:

Association : 
Attachment : Amazon.EC2.Model.NetworkInterfaceAttachment
AvailabilityZone : us-west-2c
Description : 
Groups : {my-security-group}
MacAddress : 0a:e9:a6:19:4c:7f
NetworkInterfaceId : eni-12345678
OwnerId : 123456789012
PrivateDnsName : ip-10-0-0-107.us-west-2.compute.internal
PrivateIpAddress : 10.0.0.107
PrivateIpAddresses : {ip-10-0-0-107.us-west-2.compute.internal}
RequesterId : 
RequesterManaged : False
SourceDestCheck : True
Status : in-use
SubnetId : subnet-1a2b3c4d
TagSet : {}
VpcId : vpc-12345678

Example 2: This example describes all your network interfaces.

Get-EC2NetworkInterface

- For API details, see DescribeNetworkInterfaces in AWS Tools for PowerShell Cmdlet Reference.

Get-EC2NetworkInterfaceAttribute

The following code example shows how to use Get-EC2NetworkInterfaceAttribute.
Tools for PowerShell

Example 1: This example describes the specified network interface.

```
Get-EC2NetworkInterfaceAttribute -NetworkInterfaceId eni-12345678 -Attribute Attachment
```

Output:
```
Attachment : Amazon.EC2.Model.NetworkInterfaceAttachment
```

Example 2: This example describes the specified network interface.

```
Get-EC2NetworkInterfaceAttribute -NetworkInterfaceId eni-12345678 -Attribute Description
```

Output:
```
Description : My description
```

Example 3: This example describes the specified network interface.

```
Get-EC2NetworkInterfaceAttribute -NetworkInterfaceId eni-12345678 -Attribute GroupSet
```

Output:
```
Groups : {my-security-group}
```

Example 4: This example describes the specified network interface.

```
Get-EC2NetworkInterfaceAttribute -NetworkInterfaceId eni-12345678 -Attribute SourceDestCheck
```

Output:
```
SourceDestCheck : True
```
For API details, see \texttt{DescribeNetworkInterfaceAttribute} in \textit{AWS Tools for PowerShell Cmdlet Reference}.

\textbf{Get-EC2PasswordData}

The following code example shows how to use \texttt{Get-EC2PasswordData}.

\textbf{Tools for PowerShell}

\textbf{Example 1:} This example decrypts the password that Amazon EC2 assigned to the Administrator account for the specified Windows instance. As a pem file was specified, the setting of the -Decrypt switch is automatically assumed.

\begin{verbatim}
Get-EC2PasswordData -InstanceId i-12345678 -PemFile C:\path\my-key-pair.pem
\end{verbatim}

\textbf{Output:}

\begin{verbatim}
mYZ(PA9?C)Q
\end{verbatim}

\textbf{Example 2: (Windows PowerShell only)} Inspects the instance to determine the name of the keypair used to launch the instance and then attempts to find the corresponding keypair data in the configuration store of the AWS Toolkit for Visual Studio. If the keypair data is found the password is decrypted.

\begin{verbatim}
Get-EC2PasswordData -InstanceId i-12345678 -Decrypt
\end{verbatim}

\textbf{Output:}

\begin{verbatim}
mYZ(PA9?C)Q
\end{verbatim}

\textbf{Example 3: Returns the encrypted password data for the instance.}

\begin{verbatim}
Get-EC2PasswordData -InstanceId i-12345678
\end{verbatim}

\textbf{Output:}

\begin{verbatim}
iVz3BAK/WAXV.....dqt8WeMA==
\end{verbatim}
• For API details, see GetPasswordData in AWS Tools for PowerShell Cmdlet Reference.

Get-EC2PlacementGroup

The following code example shows how to use Get-EC2PlacementGroup.

Tools for PowerShell

Example 1: This example describes the specified placement group.

Get-EC2PlacementGroup -GroupName my-placement-group

Output:

<table>
<thead>
<tr>
<th>GroupName</th>
<th>State</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>my-placement-group</td>
<td>available</td>
<td>cluster</td>
</tr>
</tbody>
</table>

• For API details, see DescribePlacementGroups in AWS Tools for PowerShell Cmdlet Reference.

Get-EC2PrefixList

The following code example shows how to use Get-EC2PrefixList.

Tools for PowerShell

Example 1: This example fetches the available AWS services in a prefix list format for the region

Get-EC2PrefixList

Output:

<table>
<thead>
<tr>
<th>Cidrs</th>
<th>PrefixListId</th>
<th>PrefixListName</th>
</tr>
</thead>
<tbody>
<tr>
<td>{52.94.5.0/24, 52.119.240.0/21, 52.94.24.0/23}</td>
<td>pl-6fa54006</td>
<td>com.amazonaws.eu-west-1.dynamodb</td>
</tr>
<tr>
<td>{52.218.0.0/17, 54.231.128.0/19}</td>
<td>pl-6da54004</td>
<td>com.amazonaws.eu-west-1.s3</td>
</tr>
</tbody>
</table>
• For API details, see [DescribePrefixLists](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-EC2Region**

The following code example shows how to use Get-EC2Region.

**Tools for PowerShell**

**Example 1: This example describes the regions that are available to you.**

```powershell
Get-EC2Region
```

**Output:**

<table>
<thead>
<tr>
<th>Endpoint</th>
<th>RegionName</th>
</tr>
</thead>
<tbody>
<tr>
<td>ec2.eu-west-1.amazonaws.com</td>
<td>eu-west-1</td>
</tr>
<tr>
<td>ec2.ap-southeast-1.amazonaws.com</td>
<td>ap-southeast-1</td>
</tr>
<tr>
<td>ec2.ap-southeast-2.amazonaws.com</td>
<td>ap-southeast-2</td>
</tr>
<tr>
<td>ec2.eu-central-1.amazonaws.com</td>
<td>eu-central-1</td>
</tr>
<tr>
<td>ec2.ap-northeast-1.amazonaws.com</td>
<td>ap-northeast-1</td>
</tr>
<tr>
<td>ec2.us-east-1.amazonaws.com</td>
<td>us-east-1</td>
</tr>
<tr>
<td>ec2.sa-east-1.amazonaws.com</td>
<td>sa-east-1</td>
</tr>
<tr>
<td>ec2.us-west-1.amazonaws.com</td>
<td>us-west-1</td>
</tr>
<tr>
<td>ec2.us-west-2.amazonaws.com</td>
<td>us-west-2</td>
</tr>
</tbody>
</table>

• For API details, see [DescribeRegions](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-EC2RouteTable**

The following code example shows how to use Get-EC2RouteTable.

**Tools for PowerShell**

**Example 1: This example describes all your route tables.**

```powershell
Get-EC2RouteTable
```

**Output:**

```
DestinationCidrBlock : 10.0.0.0/16
```
Example 2: This example returns details for the specified route table.

Get-EC2RouteTable -RouteTableId rtb-1a2b3c4d

Example 3: This example describes the route tables for the specified VPC.

Get-EC2RouteTable -Filter @{ Name="vpc-id"; Values="vpc-1a2b3c4d" }

Output:

Associations : {rtbassoc-12345678}
PropagatingVgw : {}
Routes : {},
RouteTableId : rtb-1a2b3c4d
Tags : {}
VpcId : vpc-1a2b3c4d

- For API details, see DescribeRouteTables in AWS Tools for PowerShell Cmdlet Reference.

Get-EC2ScheduledInstance

The following code example shows how to use Get-EC2ScheduledInstance.
Tools for PowerShell

Example 1: This example describes the specified Scheduled Instance.

```powershell
Get-EC2ScheduledInstance -ScheduledInstanceId sci-1234-1234-1234-1234-123456789012
```

Output:

```
AvailabilityZone : us-west-2b
CreateDate       : 1/25/2016 1:43:38 PM
HourlyPrice      : 0.095
InstanceCount    : 1
InstanceType     : c4.large
NetworkPlatform  : EC2-VPC
NextSlotStartTime: 1/31/2016 1:00:00 AM
Platform         : Linux/UNIX
PreviousSlotEndTime:
Recurrence       : Amazon.EC2.Model.ScheduledInstanceRecurrence
ScheduledInstanceId: sci-1234-1234-1234-1234-123456789012
SlotDurationInHours: 32
TermEndDate      : 1/31/2017 1:00:00 AM
TermStartDate    : 1/31/2016 1:00:00 AM
TotalScheduledInstanceHours: 1696
```

Example 2: This example describes all your Scheduled Instances.

```powershell
Get-EC2ScheduledInstance
```

- For API details, see [DescribeScheduledInstances](https://docs.aws.amazon.com/tools/latest/dw/latest/reference/pgcmdlet.html#DescribeScheduledInstances) in AWS Tools for PowerShell Cmdlet Reference.

Get-EC2ScheduledInstanceAvailability

The following code example shows how to use Get-EC2ScheduledInstanceAvailability.

Tools for PowerShell

Example 1: This example describes a schedule that occurs every week on Sunday, starting on the specified date.

```powershell
Get-EC2ScheduledInstanceAvailability -Recurrence_Frequency Weekly -Recurrence_Interval 1 -Recurrence_OccurrenceDay 1 -
```
FirstSlotStartTimeRange_EarliestTime 2016-01-31T00:00:00Z -
FirstSlotStartTimeRange_LatestTime 2016-01-31T04:00:00Z

Output:

AvailabilityZone            : us-west-2b
AvailableInstanceCount      : 20
FirstSlotStartTime          : 1/31/2016 8:00:00 AM
HourlyPrice                 : 0.095
InstanceType                : c4.large
MaxTermDurationInDays       : 366
MinTermDurationInDays       : 366
NetworkPlatform             : EC2-VPC
Platform                    : Linux/UNIX
PurchaseToken               : eyJ2IjoiMSIsInMiOjEsImMi0i...
Recurrence                  : Amazon.EC2.Model.ScheduledInstanceRecurrence
SlotDurationInHours         : 23
TotalScheduledInstanceHours : 1219

Example 2: To narrow the results, you can add filters for criteria such as operating system, network, and instance type.

-Filter @{ Name="platform";Values="Linux/UNIX" },@{ Name="network-platform";Values="EC2-VPC" },@{ Name="instance-type";Values="c4.large" }

- For API details, see DescribeScheduledInstanceAvailability in AWS Tools for PowerShell Cmdlet Reference.

Get-EC2SecurityGroup

The following code example shows how to use Get-EC2SecurityGroup.

Tools for PowerShell

Example 1: This example describes the specified security group for a VPC. When working with security groups belonging to a VPC you must use the security group ID (-GroupId parameter), not name (-GroupName parameter), to reference the group.

Get-EC2SecurityGroup -GroupId sg-12345678
Output:

<table>
<thead>
<tr>
<th>Description</th>
<th>: default VPC security group</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupId</td>
<td>: sg-12345678</td>
</tr>
<tr>
<td>GroupName</td>
<td>: default</td>
</tr>
<tr>
<td>IpPermissions</td>
<td>: {Amazon.EC2.Model.IpPermission}</td>
</tr>
<tr>
<td>IpPermissionsEgress</td>
<td>: {Amazon.EC2.Model.IpPermission}</td>
</tr>
<tr>
<td>OwnerId</td>
<td>: 123456789012</td>
</tr>
<tr>
<td>Tags</td>
<td>: {}</td>
</tr>
<tr>
<td>VpcId</td>
<td>: vpc-12345678</td>
</tr>
</tbody>
</table>

Example 2: This example describes the specified security group for EC2-Classic. When working with security groups for EC2-Classic you may use either the group name (-GroupName parameter) or group ID (-GroupId parameter) to reference the security group.

```
Get-EC2SecurityGroup -GroupName my-security-group
```

Output:

<table>
<thead>
<tr>
<th>Description</th>
<th>: my security group</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupId</td>
<td>: sg-45678901</td>
</tr>
<tr>
<td>GroupName</td>
<td>: my-security-group</td>
</tr>
<tr>
<td>IpPermissions</td>
<td>: {Amazon.EC2.Model.IpPermission, Amazon.EC2.Model.IpPermission}</td>
</tr>
<tr>
<td>IpPermissionsEgress</td>
<td>: {}</td>
</tr>
<tr>
<td>OwnerId</td>
<td>: 123456789012</td>
</tr>
<tr>
<td>Tags</td>
<td>: {}</td>
</tr>
<tr>
<td>VpcId</td>
<td>:</td>
</tr>
</tbody>
</table>

Example 3: This example retrieves all the security groups for the vpc-0fc1ff23456b789eb

```
Get-EC2SecurityGroup -Filter @{Name="vpc-id";Values="vpc-0fc1ff23456b789eb"}
```

- For API details, see [DescribeSecurityGroups](#) in AWS Tools for PowerShell Cmdlet Reference.

**Get-EC2Snapshot**

The following code example shows how to use Get-EC2Snapshot.

**Tools for PowerShell**

Example 1: This example describes the specified snapshot.
Get-EC2Snapshot -SnapshotId snap-12345678

**Output:**

```powershell
data encryptionKeyId : Description : Created by CreateImage(i-1a2b3c4d) for ami-12345678 from vol-12345678
encrypted : False
KmsKeyId :
OwnerAlias :
OwnerId : 123456789012
Progress : 100%
SnapshotId : snap-12345678
StartTime : 10/23/2014 6:01:28 AM
State : completed
StateMessage :
Tags :
VolumeId : vol-12345678
VolumeSize : 8
```

**Example 2:** This example describes the snapshots that have a 'Name' tag.

```powershell
```

**Example 3:** This example describes the snapshots that have a 'Name' tag with the value 'TestValue'.

```powershell
Get-EC2Snapshot | ? { $.Tags.Count -gt 0 -and $.Tags.Key -eq "Name" -and
$.Tags.Value -eq "TestValue" }
```

**Example 4:** This example describes all your snapshots.

```powershell
Get-EC2Snapshot -Owner self
```

- For API details, see [DescribeSnapshots](https://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_DescribeSnapshots.html) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-EC2SnapshotAttribute

The following code example shows how to use Get-EC2SnapshotAttribute.
Tools for PowerShell

Example 1: This example describes the specified attribute of the specified snapshot.

```
Get-EC2SnapshotAttribute -SnapshotId snap-12345678 -Attribute ProductCodes
```

Output:

<table>
<thead>
<tr>
<th>CreateVolumePermissions</th>
<th>ProductCodes</th>
<th>SnapshotId</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>{}</td>
<td>snap-12345678</td>
</tr>
</tbody>
</table>

Example 2: This example describes the specified attribute of the specified snapshot.

```
(Get-EC2SnapshotAttribute -SnapshotId snap-12345678 -Attribute CreateVolumePermission).CreateVolumePermissions
```

Output:

<table>
<thead>
<tr>
<th>Group</th>
<th>UserId</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td></td>
</tr>
</tbody>
</table>

- For API details, see [DescribeSnapshotAttribute](https://docs.aws.amazon.com/AWSEC2/latest/userguide/using-ec2snapshots.html) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-EC2SpotDatafeedSubscription

The following code example shows how to use Get-EC2SpotDatafeedSubscription.

Tools for PowerShell

Example 1: This example describes your Spot instance data feed.

```
Get-EC2SpotDatafeedSubscription
```

Output:

<table>
<thead>
<tr>
<th>Bucket</th>
<th>my-s3-bucket</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault</td>
<td></td>
</tr>
<tr>
<td>OwnerId</td>
<td>123456789012</td>
</tr>
<tr>
<td>Prefix</td>
<td>spotdata</td>
</tr>
</tbody>
</table>
For API details, see `DescribeSpotDatafeedSubscription` in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-EC2SpotFleetInstance

The following code example shows how to use `Get-EC2SpotFleetInstance`.

**Tools for PowerShell**

**Example 1:** This example describes the instances associated with the specified Spot fleet request.

```powershell
Get-EC2SpotFleetInstance -SpotFleetRequestId sfr-73fbd2ce-aa30-494c-8788-1cee4EXAMPLE
```

**Output:**

<table>
<thead>
<tr>
<th>InstanceId</th>
<th>InstanceType</th>
<th>SpotInstanceRequestID</th>
</tr>
</thead>
<tbody>
<tr>
<td>i-f089262a</td>
<td>c3.large</td>
<td>sir-12345678</td>
</tr>
<tr>
<td>i-7e8b24a4</td>
<td>c3.large</td>
<td>sir-87654321</td>
</tr>
</tbody>
</table>

For API details, see `DescribeSpotFleetInstances` in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-EC2SpotFleetRequest

The following code example shows how to use `Get-EC2SpotFleetRequest`.

**Tools for PowerShell**

**Example 1:** This example describes the specified Spot fleet request.

```powershell
Get-EC2SpotFleetRequest -SpotFleetRequestId sfr-73fbd2ce-aa30-494c-8788-1cee4EXAMPLE | format-list
```

**Output:**

```
ConfigData            : Amazon.EC2.Model.SpotFleetRequestConfigData
```
Example 2: This example describes all your Spot fleet requests.

Get-EC2SpotFleetRequest

- For API details, see [DescribeSpotFleetRequests](https://docs.aws.amazon.com/tools/latest/dwpsg/SpotFleetRequests.html) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-EC2SpotFleetRequestHistory

The following code example shows how to use Get-EC2SpotFleetRequestHistory.

Tools for PowerShell

Example 1: This example describes the history of the specified Spot fleet request.

Get-EC2SpotFleetRequestHistory -SpotFleetRequestId sfr-73fbd2ce-aa30-494c-8788-1cee4EXAMPLE -StartTime 2015-12-26T00:00:00Z

Output:

<table>
<thead>
<tr>
<th>EventInformation</th>
<th>EventType</th>
<th>Timestamp</th>
</tr>
</thead>
</table>
- For API details, see [DescribeSpotFleetRequestHistory](#) in **AWS Tools for PowerShell Cmdlet Reference**.

**Get-EC2SpotInstanceRequest**

The following code example shows how to use Get-EC2SpotInstanceRequest.

**Tools for PowerShell**

**Example 1:** This example describes the specified Spot instance request.

Get-EC2SpotInstanceRequest -SpotInstanceRequestId sir-12345678

**Output:**

- ActualBlockHourlyPrice : 
- AvailabilityZoneGroup : 
- BlockDurationMinutes : 0
- CreateTime : 4/8/2015 2:51:33 PM
- Fault : 
- InstanceId : i-12345678
- LaunchedAvailabilityZone : us-west-2b
- LaunchGroup : 
- ProductDescription : Linux/UNIX
- SpotInstanceRequestId : sir-12345678
- SpotPrice : 0.020000
- State : active
- Status : Amazon.EC2.Model.SpotInstanceStatus
- Tags : {Name}
- Type : one-time

**Example 2:** This example describes all your Spot instance requests.

Get-EC2SpotInstanceRequest

- For API details, see [DescribeSpotInstanceRequests](#) in **AWS Tools for PowerShell Cmdlet Reference**.
Get-EC2SpotPriceHistory

The following code example shows how to use Get-EC2SpotPriceHistory.

Tools for PowerShell

Example 1: This example gets the last 10 entries in the Spot price history for the specified instance type and Availability Zone. Note that the value specified for the -AvailabilityZone parameter must be valid for the region value supplied to either the cmdlet's -Region parameter (not shown in the example) or set as default in the shell. This example command assumes a default region of 'us-west-2' has been set in the environment.

```
Get-EC2SpotPriceHistory -InstanceType c3.large -AvailabilityZone us-west-2a -MaxResult 10
```

Output:

```
AvailabilityZone   : us-west-2a
InstanceType       : c3.large
Price              : 0.017300
ProductDescription : Linux/UNIX (Amazon VPC)
Timestamp          : 12/25/2015 7:39:49 AM

AvailabilityZone   : us-west-2a
InstanceType       : c3.large
Price              : 0.017200
ProductDescription : Linux/UNIX (Amazon VPC)
Timestamp          : 12/25/2015 7:38:29 AM

AvailabilityZone   : us-west-2a
InstanceType       : c3.large
Price              : 0.017300
ProductDescription : Linux/UNIX (Amazon VPC)
Timestamp          : 12/25/2015 6:57:13 AM
...
```

- For API details, see [DescribeSpotPriceHistory](#) in AWS Tools for PowerShell Cmdlet Reference.

Get-EC2Subnet

The following code example shows how to use Get-EC2Subnet.
Example 1: This example describes the specified subnet.

Get-EC2Subnet -SubnetId subnet-1a2b3c4d

Output:

<table>
<thead>
<tr>
<th>AvailabilityZone</th>
<th>: us-west-2c</th>
</tr>
</thead>
<tbody>
<tr>
<td>AvailableIpAddressCount</td>
<td>: 251</td>
</tr>
<tr>
<td>CidrBlock</td>
<td>: 10.0.0.0/24</td>
</tr>
<tr>
<td>DefaultForAz</td>
<td>: False</td>
</tr>
<tr>
<td>MapPublicIpOnLaunch</td>
<td>: False</td>
</tr>
<tr>
<td>State</td>
<td>: available</td>
</tr>
<tr>
<td>SubnetId</td>
<td>: subnet-1a2b3c4d</td>
</tr>
<tr>
<td>Tags</td>
<td>: {}</td>
</tr>
<tr>
<td>VpcId</td>
<td>: vpc-12345678</td>
</tr>
</tbody>
</table>

Example 2: This example describes all your subnets.

Get-EC2Subnet

- For API details, see [DescribeSubnets](#) in AWS Tools for PowerShell Cmdlet Reference.

Get-EC2Tag

The following code example shows how to use Get-EC2Tag.

Example 1: This example fetches the tags for resource-type 'image'

Get-EC2Tag -Filter @{Name="resource-type";Values="image"}

Output:

<table>
<thead>
<tr>
<th>Key</th>
<th>ResourceId</th>
<th>ResourceType</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>ami-0a123b4cccb567a8ea</td>
<td>image</td>
<td>Win7-Imported</td>
</tr>
</tbody>
</table>
auto-delete ami-0a123b4cc567a8ea image never

Example 2: This example fetches all the tags for all the resources and groups them by resource type

Get-EC2Tag | Group-Object resourcetype

Output:

<table>
<thead>
<tr>
<th>Count</th>
<th>Name</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>subnet</td>
<td>{Amazon.EC2.Model.TagDescription, Amazon.EC2.Model.TagDescription, Amazon.EC2.Model.TagDescription,...}</td>
</tr>
<tr>
<td>53</td>
<td>instance</td>
<td>{Amazon.EC2.Model.TagDescription, Amazon.EC2.Model.TagDescription, Amazon.EC2.Model.TagDescription,...}</td>
</tr>
<tr>
<td>3</td>
<td>route-table</td>
<td>{Amazon.EC2.Model.TagDescription, Amazon.EC2.Model.TagDescription, Amazon.EC2.Model.TagDescription,...}</td>
</tr>
<tr>
<td>5</td>
<td>security-group</td>
<td>{Amazon.EC2.Model.TagDescription, Amazon.EC2.Model.TagDescription, Amazon.EC2.Model.TagDescription,...}</td>
</tr>
<tr>
<td>30</td>
<td>volume</td>
<td>{Amazon.EC2.Model.TagDescription, Amazon.EC2.Model.TagDescription, Amazon.EC2.Model.TagDescription,...}</td>
</tr>
<tr>
<td>1</td>
<td>internet-gateway</td>
<td>{Amazon.EC2.Model.TagDescription}</td>
</tr>
<tr>
<td>3</td>
<td>network-interface</td>
<td>{Amazon.EC2.Model.TagDescription, Amazon.EC2.Model.TagDescription,...}</td>
</tr>
<tr>
<td>4</td>
<td>elastic-ip</td>
<td>{Amazon.EC2.Model.TagDescription, Amazon.EC2.Model.TagDescription,...}</td>
</tr>
<tr>
<td>1</td>
<td>dhcp-options</td>
<td>{Amazon.EC2.Model.TagDescription}</td>
</tr>
<tr>
<td>2</td>
<td>image</td>
<td>{Amazon.EC2.Model.TagDescription}</td>
</tr>
<tr>
<td>3</td>
<td>vpc</td>
<td>{Amazon.EC2.Model.TagDescription, Amazon.EC2.Model.TagDescription,...}</td>
</tr>
</tbody>
</table>

Example 3: This example displays all the resources with tag 'auto-delete' with value 'no' for the given region

Get-EC2Tag -Region eu-west-1 -Filter @{Name="tag:auto-delete";Values="no"}
Output:

<table>
<thead>
<tr>
<th>Key</th>
<th>ResourceId</th>
<th>ResourceType</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto-delete</td>
<td>i-0f1bce23d5dd678b</td>
<td>instance</td>
<td>no</td>
</tr>
<tr>
<td>auto-delete</td>
<td>vol-01d234aa5678901a2</td>
<td>volume</td>
<td>no</td>
</tr>
<tr>
<td>auto-delete</td>
<td>vol-01234bfb5def6f7b8</td>
<td>volume</td>
<td>no</td>
</tr>
<tr>
<td>auto-delete</td>
<td>vol-01ccb23f4c5e67890</td>
<td>volume</td>
<td>no</td>
</tr>
</tbody>
</table>

Example 4: This example obtains all the resources with tag 'auto-delete' with 'no' value and further filters in the next pipe to parse only 'instance' resource types and eventually creates 'ThisInstance' tag for each instance resources with value being the instance id itself.

Get-EC2Tag -Region eu-west-1 -Filter @{Name="tag:auto-delete";Values="no"} | Where-Object ResourceType -eq "instance" | ForEach-Object {New-EC2Tag -ResourceId $_.ResourceId -Tag @{Key="ThisInstance";Value=$_.ResourceId}}

Example 5: This example fetches tags for all the instance resources as well as 'Name' keys and displays them in a table format.

Get-EC2Tag -Filter @{Name="resource-type";Values="instance"},@{Name="key";Values="Name"} | Select-Object ResourceId, @{Name="Name-Tag";Expression={$PSItem.Value}} | Format-Table -AutoSize

Output:

<table>
<thead>
<tr>
<th>ResourceId</th>
<th>Name-Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>i-012e3cb4df567e1aa</td>
<td>jump1</td>
</tr>
<tr>
<td>i-01c23a45d6fc7a89f</td>
<td>repro-3</td>
</tr>
</tbody>
</table>

- For API details, see [DescribeTags](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

**Get-EC2Volume**

The following code example shows how to use Get-EC2Volume.

**Tools for PowerShell**

Example 1: This example describes the specified EBS volume.
Get-EC2Volume -VolumeId vol-12345678

**Output:**

<table>
<thead>
<tr>
<th>Attachments</th>
<th>: {}</th>
</tr>
</thead>
<tbody>
<tr>
<td>AvailabilityZone</td>
<td>: us-west-2c</td>
</tr>
<tr>
<td>CreateTime</td>
<td>: 7/17/2015 4:35:19 PM</td>
</tr>
<tr>
<td>Encrypted</td>
<td>: False</td>
</tr>
<tr>
<td>Iops</td>
<td>: 90</td>
</tr>
<tr>
<td>KmsKeyId</td>
<td>:</td>
</tr>
<tr>
<td>Size</td>
<td>: 30</td>
</tr>
<tr>
<td>SnapshotId</td>
<td>: snap-12345678</td>
</tr>
<tr>
<td>State</td>
<td>: in-use</td>
</tr>
<tr>
<td>Tags</td>
<td>: {}</td>
</tr>
<tr>
<td>VolumeId</td>
<td>: vol-12345678</td>
</tr>
<tr>
<td>VolumeType</td>
<td>: standard</td>
</tr>
</tbody>
</table>

**Example 2:** This example describes your EBS volumes that have the status 'available'.

Get-EC2Volume -Filter @{ Name="status"; Values="available" }

**Output:**

<table>
<thead>
<tr>
<th>Attachments</th>
<th>: {}</th>
</tr>
</thead>
<tbody>
<tr>
<td>AvailabilityZone</td>
<td>: us-west-2c</td>
</tr>
<tr>
<td>CreateTime</td>
<td>: 12/21/2015 2:31:29 PM</td>
</tr>
<tr>
<td>Encrypted</td>
<td>: False</td>
</tr>
<tr>
<td>Iops</td>
<td>: 60</td>
</tr>
<tr>
<td>KmsKeyId</td>
<td>:</td>
</tr>
<tr>
<td>Size</td>
<td>: 20</td>
</tr>
<tr>
<td>SnapshotId</td>
<td>: snap-12345678</td>
</tr>
<tr>
<td>State</td>
<td>: available</td>
</tr>
<tr>
<td>Tags</td>
<td>: {}</td>
</tr>
<tr>
<td>VolumeId</td>
<td>: vol-12345678</td>
</tr>
<tr>
<td>VolumeType</td>
<td>: gp2</td>
</tr>
</tbody>
</table>

**Example 3:** This example describes all your EBS volumes.

Get-EC2Volume
For API details, see [DescribeVolumes](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-EC2VolumeAttribute**

The following code example shows how to use Get-EC2VolumeAttribute.

**Tools for PowerShell**

**Example 1:** This example describes the specified attribute of the specified volume.

```powershell
Get-EC2VolumeAttribute -VolumeId vol-12345678 -Attribute AutoEnableIO
```

**Output:**

<table>
<thead>
<tr>
<th>AutoEnableIO</th>
<th>ProductCodes</th>
<th>VolumeId</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>{}</td>
<td>vol-12345678</td>
</tr>
</tbody>
</table>

For API details, see [DescribeVolumeAttribute](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-EC2VolumeStatus**

The following code example shows how to use Get-EC2VolumeStatus.

**Tools for PowerShell**

**Example 1:** This example describes the status of the specified volume.

```powershell
Get-EC2VolumeStatus -VolumeId vol-12345678
```

**Output:**

```
Actions : {}
AvailabilityZone : us-west-2a
Events : {}
VolumeId : vol-12345678
VolumeStatus : Amazon.EC2.Model.VolumeStatusInfo
```

```
(Get-EC2VolumeStatus -VolumeId vol-12345678).VolumeStatus
```
Output:

<table>
<thead>
<tr>
<th>Details</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>{io-enabled, io-performance}</td>
<td>ok</td>
</tr>
</tbody>
</table>

(Get-EC2VolumeStatus -VolumeId vol-12345678).VolumeStatus.Details

Output:

<table>
<thead>
<tr>
<th>Name</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>io-enabled</td>
<td>passed</td>
</tr>
<tr>
<td>io-performance</td>
<td>not-applicable</td>
</tr>
</tbody>
</table>

- For API details, see [DescribeVolumeStatus](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-EC2Vpc

The following code example shows how to use Get-EC2Vpc.

### Tools for PowerShell

#### Example 1: This example describes the specified VPC.

Get-EC2Vpc -VpcId vpc-12345678

Output:

- CidrBlock : 10.0.0.0/16
- DhcpOptionsId : dopt-1a2b3c4d
- InstanceTenancy : default
- IsDefault : False
- State : available
- Tags : {Name}
- VpcId : vpc-12345678

#### Example 2: This example describes the default VPC (there can be only one per region). If your account supports EC2-Classic in this region, there is no default VPC.
Get-EC2Vpc -Filter @{Name="isDefault"; Values="true"}

Output:

CidrBlock       : 172.31.0.0/16
DhcpOptionsId   : dopt-12345678
InstanceTenancy : default
IsDefault       : True
State           : available
Tags            : {}
VpcId           : vpc-45678901

Example 3: This example describes the VPCs that match the specified filter (that is, have a CIDR that matches the value '10.0.0.0/16' and are in the state 'available').

Get-EC2Vpc -Filter @{Name="cidr"; Values="10.0.0.0/16"},@{Name="state";Values="available"}

Example 4: This example describes all your VPCs.

Get-EC2Vpc

- For API details, see DescribeVpcs in AWS Tools for PowerShell Cmdlet Reference.

Get-EC2VpcAttribute

The following code example shows how to use Get-EC2VpcAttribute.

Tools for PowerShell

Example 1: This example describes the 'enableDnsSupport' attribute.

Get-EC2VpcAttribute -VpcId vpc-12345678 -Attribute enableDnsSupport

Output:

EnableDnsSupport
---------------
Example 2: This example describes the 'enableDnsHostnames' attribute.

Get-EC2VpcAttribute -VpcId vpc-12345678 -Attribute enableDnsHostnames

Output:

EnableDnsHostnames
------------------
True

- For API details, see DescribeVpcAttribute in AWS Tools for PowerShell Cmdlet Reference.

Get-EC2VpcClassicLink

The following code example shows how to use Get-EC2VpcClassicLink.

Example 1: Above example returns all the VPCs with their ClassicLinkEnabled state for the region

Get-EC2VpcClassicLink -Region eu-west-1

Output:

<table>
<thead>
<tr>
<th>ClassicLinkEnabled</th>
<th>Tags</th>
<th>VpcId</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>{Name}</td>
<td>vpc-0fc1ff23f45b678eb</td>
</tr>
<tr>
<td>False</td>
<td>{}</td>
<td>vpc-01e23c4a5d6db78e9</td>
</tr>
<tr>
<td>False</td>
<td>{Name}</td>
<td>vpc-0123456b078b9d01f</td>
</tr>
<tr>
<td>False</td>
<td>{}</td>
<td>vpc-12cf3b4f</td>
</tr>
<tr>
<td>False</td>
<td>{Name}</td>
<td>vpc-0b12d3456a7e8901d</td>
</tr>
</tbody>
</table>

- For API details, see DescribeVpcClassicLink in AWS Tools for PowerShell Cmdlet Reference.

Get-EC2VpcClassicLinkDnsSupport

The following code example shows how to use Get-EC2VpcClassicLinkDnsSupport.
Tools for PowerShell

Example 1: This example describes the ClassicLink DNS support status of VPCs for the region eu-west-1

```powershell
Get-EC2VpcClassicLinkDnsSupport -VpcId vpc-0b12d3456a7e8910d -Region eu-west-1
```

Output:

```
ClassLinkDnsSupported  VpcId
----------------------- ----- 
False                   vpc-0b12d3456a7e8910d
False                   vpc-12cf3b4f
```

- For API details, see [DescribeVpcClassicLinkDnsSupport](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-EC2VpcEndpoint

The following code example shows how to use Get-EC2VpcEndpoint.

Tools for PowerShell

Example 1: This example describes one or more of your VPC endpoints for the region eu-west-1. It then pipes the output to the next command, which select the VpcEndpointId property and returns array VPC ID as string array

```powershell
Get-EC2VpcEndpoint -Region eu-west-1 | Select-Object -ExpandProperty VpcEndpointId
```

Output:

```
vpce-01a2ab3f4f5cc6f7d
vpce-01d2b345a6787890b
vpce-0012e34d567890e12
vpce-0c123db4567890123
```

Example 2: This example describes all the vpc endpoints for the region eu-west-1 and selects VpcEndpointId, VpcId, ServiceName and PrivateDnsEnabled properties to present it in a tabular format
Get-EC2VpcEndpoint -Region eu-west-1 | Select-Object VpcEndpointId, VpcId, ServiceName, PrivateDnsEnabled | Format-Table -AutoSize

**Output:**

<table>
<thead>
<tr>
<th>VpcEndpointId</th>
<th>VpcId</th>
<th>ServiceName</th>
</tr>
</thead>
<tbody>
<tr>
<td>vpce-02a2ab2f2f2cc2f2d</td>
<td>vpc-0fc6ff46f65b039eb</td>
<td>com.amazonaws.eu-west-1.ssm</td>
</tr>
<tr>
<td>True</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vpce-01d1b111a1114561b</td>
<td>vpc-0fc6ff46f65b039eb</td>
<td>com.amazonaws.eu-west-1.ec2</td>
</tr>
<tr>
<td>True</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vpce-0011e23d45167e838</td>
<td>vpc-0fc6ff46f65b039eb</td>
<td>com.amazonaws.eu-west-1.ec2messages</td>
</tr>
<tr>
<td>True</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vpce-0c123db4567890123</td>
<td>vpc-0fc6ff46f65b039eb</td>
<td>com.amazonaws.eu-west-1.ssmmessages</td>
</tr>
<tr>
<td>True</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example 3:** This example exports the policy document for the VPC Endpoint vpce-01a2ab3f4f5cc6f7d into a json file

Get-EC2VpcEndpoint -Region eu-west-1 -VpcEndpointId vpce-01a2ab3f4f5cc6f7d | Select-Object -expand PolicyDocument | Out-File vpce_policyDocument.json

- For API details, see [DescribeVpcEndpoints](https://aws.amazon.com/api-tools-powershell-cmdlet-reference/) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-EC2VpcEndpointService**

The following code example shows how to use `Get-EC2VpcEndpointService`.

**Tools for PowerShell**

**Example 1:** This example describes EC2 VPC endpoint service with the given filter, in this case `com.amazonaws.eu-west-1.ecs`. Further, it also expands the ServiceDetails property and displays the details

Get-EC2VpcEndpointService -Region eu-west-1 -MaxResult 5 -Filter @{Name="serviceName";Values="com.amazonaws.eu-west-1.ecs"} | Select-Object -ExpandProperty ServiceDetails
Output:

AcceptanceRequired : False
AvailabilityZones : {eu-west-1a, eu-west-1b, eu-west-1c}
BaseEndpointDnsNames : {ecs.eu-west-1.vpce.amazonaws.com}
Owner : amazon
PrivateDnsName : ecs.eu-west-1.amazonaws.com
ServiceName : com.amazonaws.eu-west-1.ecs
ServiceType : {Amazon.EC2.Model.ServiceTypeDetail}
VpcEndpointPolicySupported : False

Example 2: This example retrieves all the EC2 VPC Endpoint services and returns the ServiceNames matching "ssm"

Get-EC2VpcEndpointService -Region eu-west-1 | Select-Object -ExpandProperty Servicenames | Where-Object { -match "ssm"}

Output:

com.amazonaws.eu-west-1.ssm
com.amazonaws.eu-west-1.ssmmessages

• For API details, see DescribeVpcEndpointServices in AWS Tools for PowerShell Cmdlet Reference.

Get-EC2VpnConnection

The following code example shows how to use Get-EC2VpnConnection.

Tools for PowerShell

Example 1: This example describes the specified VPN connection.

Get-EC2VpnConnection -VpnConnectionId vpn-12345678

Output:

CustomerGatewayConfiguration : [XML document]
CustomerGatewayId : cgw-1a2b3c4d
Options : Amazon.EC2.Model.VpnConnectionOptions
Routes : {Amazon.EC2.Model.VpnStaticRoute}
Example 2: This example describes any VPN connection whose state is either pending or available.

```powershell
$filter = New-Object Amazon.EC2.Model.Filter
$filter.Name = "state"
$filter.Values = @( "pending", "available" )

Get-EC2VpnConnection -Filter $filter
```

Example 3: This example describes all your VPN connections.

```powershell
Get-EC2VpnConnection
```

- For API details, see [DescribeVpnConnections](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-EC2VpnGateway

The following code example shows how to use `Get-EC2VpnGateway`.

**Tools for PowerShell**

**Example 1: This example describes the specified virtual private gateway.**

```powershell
Get-EC2VpnGateway -VpnGatewayId vgw-1a2b3c4d
```

**Output:**

```
AvailabilityZone : 
State : available
Tags : {}
Type : ipsec.1
VpcAttachments : {vpc-12345678}
VpnGatewayId : vgw-1a2b3c4d
```
Example 2: This example describes any virtual private gateway whose state is either pending or available.

```powershell
$filter = New-Object Amazon.EC2.Model.Filter
$filter.Name = "state"
$filter.Values = @( "pending", "available" )

Get-EC2VpnGateway -Filter $filter
```

Example 3: This example describes all your virtual private gateways.

```powershell
Get-EC2VpnGateway
```

- For API details, see [DescribeVpnGateways](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Grant-EC2SecurityGroupEgress**

The following code example shows how to use `Grant-EC2SecurityGroupEgress`.

**Tools for PowerShell**

**Example 1:** This example defines an egress rule for the specified security group for EC2-VPC. The rule grants access to the specified IP address range on TCP port 80. The syntax used by this example requires PowerShell version 3 or higher.

```powershell
$ip = @{ IpProtocol="tcp"; FromPort="80"; ToPort="80"; IpRanges="203.0.113.0/24" }
Grant-EC2SecurityGroupEgress -GroupId sg-12345678 -IpPermission $ip
```

**Example 2:** With PowerShell version 2, you must use `New-Object` to create the `IpPermission` object.

```powershell
$ip = New-Object Amazon.EC2.Model.IpPermission
$ip.IpProtocol = "tcp"
$ip.FromPort = 80
$ip.ToPort = 80
$ip.IpRanges.Add("203.0.113.0/24")

Grant-EC2SecurityGroupEgress -GroupId sg-12345678 -IpPermission $ip
```

**Example 3:** This example grants access to the specified source security group on TCP port 80.
$ug = New-Object Amazon.EC2.Model.UserIdGroupPair
$ug.GroupId = "sg-1a2b3c4d"
$ug.UserId = "123456789012"

Grant-EC2SecurityGroupEgress -GroupId sg-12345678 -IpPermission
@{ IpProtocol="tcp"; FromPort="80"; ToPort="80"; UserIdGroupPairs=$ug } )

- For API details, see AuthorizeSecurityGroupEgress in AWS Tools for PowerShell Cmdlet Reference.

**Grant-EC2SecurityGroupIngress**

The following code example shows how to use Grant-EC2SecurityGroupIngress.

**Tools for PowerShell**

**Example 1:** This example defines ingress rules for a security group for EC2-VPC. These rules grant access to a specific IP address for SSH (port 22) and RDC (port 3389). Note that you must identify security groups for EC2-VPC using the security group ID not the security group name. The syntax used by this example requires PowerShell version 3 or higher.

```powershell
$ip1 = @{ IpProtocol="tcp"; FromPort="22"; ToPort="22"; IpRanges="203.0.113.25/32" }
$ip2 = @{ IpProtocol="tcp"; FromPort="3389"; ToPort="3389";
    IpRanges="203.0.113.25/32" }

Grant-EC2SecurityGroupIngress -GroupId sg-12345678 -IpPermission @($ip1, $ip2)
```

**Example 2:** With PowerShell version 2, you must use New-Object to create the IpPermission objects.

```powershell
$ip1 = New-Object Amazon.EC2.Model.IpPermission
$ip1.IpProtocol = "tcp"
$ip1.FromPort = 22
$ip1.ToPort = 22
$ip1.IpRanges.Add("203.0.113.25/32")

$ip2 = new-object Amazon.EC2.Model.IpPermission
$ip2.IpProtocol = "tcp"
$ip2.FromPort = 3389
$ip2.ToPort = 3389
$ip2.IpRanges.Add("203.0.113.25/32")
```
Grant-EC2SecurityGroupIngress -GroupId sg-12345678 -IpPermission @( $ip1, $ip2 )

**Example 3:** This example defines ingress rules for a security group for EC2-Classic. These rules grant access to a specific IP address for SSH (port 22) and RDC (port 3389). The syntax used by this example requires PowerShell version 3 or higher.

```powershell
$ip1 = @{ IpProtocol="tcp"; FromPort="22"; ToPort="22"; IpRanges="203.0.113.25/32" }
$ip2 = @{ IpProtocol="tcp"; FromPort="3389"; ToPort="3389";
         IpRanges="203.0.113.25/32" }
Grant-EC2SecurityGroupIngress -GroupName "my-security-group" -IpPermission @( $ip1, $ip2 )
```

**Example 4:** With PowerShell version 2, you must use New-Object to create the IpPermission objects.

```powershell
$ip1 = New-Object Amazon.EC2.Model.IpPermission
$ip1.IpProtocol = "tcp"
$ip1.FromPort = 22
$ip1.ToPort = 22
$ip1.IpRanges.Add("203.0.113.25/32")

$ip2 = New-Object Amazon.EC2.Model.IpPermission
$ip2.IpProtocol = "tcp"
$ip2.FromPort = 3389
$ip2.ToPort = 3389
$ip2.IpRanges.Add("203.0.113.25/32")

Grant-EC2SecurityGroupIngress -GroupName "my-security-group" -IpPermission @( $ip1, $ip2 )
```

**Example 5:** This example grants TCP port 8081 access from the specified source security group (sg-1a2b3c4d) to the specified security group (sg-12345678).

```powershell
$ug = New-Object Amazon.EC2.Model.UserIdGroupPair
$ug.GroupId = "sg-1a2b3c4d"
$ug.UserId = "123456789012"

Grant-EC2SecurityGroupIngress -GroupId sg-12345678 -IpPermission @( @{ IpProtocol="tcp"; FromPort="8081"; ToPort="8081"; UserIdGroupPairs=$ug } )
```
Example 6: This example adds the CIDR 5.5.5.5/32 to the Ingress rules of security Group sg-1234abcd for TCP port 22 traffic with a description.

```powershell
$IpRange = New-Object -TypeName Amazon.EC2.Model.IpRange
$IpRange.CidrIp = "5.5.5.5/32"
$IpRange.Description = "SSH from Office"
$IpPermission = New-Object Amazon.EC2.Model.IpPermission
$IpPermission.IpProtocol = "tcp"
$IpPermission.ToPort = 22
$IpPermission.FromPort = 22
$IpPermission.Ipv4Ranges = $IpRange
Grant-EC2SecurityGroupIngress -GroupId sg-1234abcd -IpPermission $IpPermission
```

- For API details, see [AuthorizeSecurityGroupIngress](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

**Import-EC2Image**

The following code example shows how to use Import-EC2Image.

**Tools for PowerShell**

Example 1: This example imports a single-disk virtual machine image from the specified Amazon S3 bucket to Amazon EC2 with an idempotency token. The example requires that a VM Import Service Role with the default name 'vmimport' exists, with a policy allowing Amazon EC2 access to the specified bucket, as explained in the VM Import Prequisities topic. To use a custom role, specify the role name using the -RoleName parameter.

```powershell
$container = New-Object Amazon.EC2.Model.ImageDiskContainer
$container.Format="VMDK"
$container.UserBucket.S3Bucket = "myVirtualMachineImages"
$container.UserBucket.S3Key = "Win_2008_Server_Standard_SP2_64-bit-disk1.vmdk"

$parms = @{
    "ClientToken"="idempotencyToken"
    "Description"="Windows 2008 Standard Image Import"
    "Platform"="Windows"
    "LicenseType"="AWS"
}
```
Import-EC2Image -DiskContainer $container @parms

Output:

Architecture : 
Description : Windows 2008 Standard Image 
Hypervisor : 
ImageId : 
ImportTaskId : import-ami-abcdefgh 
LicenseType : AWS 
Platform : Windows 
Progress : 2 
SnapshotDetails : {} 
Status : active 
StatusMessage : pending 

- For API details, see ImportImage in AWS Tools for PowerShell Cmdlet Reference.

Import-EC2KeyPair

The following code example shows how to use Import-EC2KeyPair.

Tools for PowerShell

Example 1: This example imports a public key to EC2. The first line stores the contents of the public key file (*.pub) in the variable $publickey. Next, the example converts the UTF8 format of the public key file to a Base64-encoded string, and stores the converted string in the variable $pkbase64. In the last line, the converted public key is imported to EC2. The cmdlet returns the key fingerprint and name as results.

```
$publickey=[Io.File]::ReadAllText("C:\Users\TestUser\.ssh\id_rsa.pub")
$pkbase64 = [System.Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes($publickey))
Import-EC2KeyPair -KeyName Example-user-key -PublicKey $pkbase64
```

Output:

KeyFingerprint KeyName
--------------- -------
d0:d0:15:8f:79:97:12:be:00:fd:df:31:z3:b1:42:z1 Example-user-key
For API details, see ImportKeyPair in AWS Tools for PowerShell Cmdlet Reference.

**Import-EC2Snapshot**

The following code example shows how to use Import-EC2Snapshot.

**Tools for PowerShell**

Example 1: This example imports a VM disk image of format 'VMDK' to an Amazon EBS snapshot. The example requires a VM Import Service Role with the default name 'vmimport', with a policy allowing Amazon EC2 access to the specified bucket, as explained in the VM Import Prequisites topic in http://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/VMImportPrerequisites.html. To use a custom role, specify the role name using the -RoleName parameter.

```powershell
$parms = @{
    "ClientToken"="idempotencyToken"
    "Description"="Disk Image Import"
    "DiskContainer_Description" = "Data disk"
    "DiskContainer_Format" = "VMDK"
    "DiskContainer_S3Bucket" = "myVirtualMachineImages"
    "DiskContainer_S3Key" = "datadiskimage.vmdk"
}

Import-EC2Snapshot @parms
```

**Output:**

<table>
<thead>
<tr>
<th>Description</th>
<th>ImportTaskId</th>
<th>SnapshotTaskDetail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk Image Import</td>
<td>import-snap-abcdefgh</td>
<td>Amazon.EC2.Model.SnapshotTaskDetail</td>
</tr>
</tbody>
</table>

For API details, see ImportSnapshot in AWS Tools for PowerShell Cmdlet Reference.

**Move-EC2AddressToVpc**

The following code example shows how to use Move-EC2AddressToVpc.
Example 1: This example moves an EC2 instance with a public IP address of 12.345.67.89 to the EC2-VPC platform in the US East (Northern Virginia) region.

```powershell
Move-EC2AddressToVpc -PublicIp 12.345.67.89 -Region us-east-1
```

Example 2: This example pipes the results of a Get-EC2Instance command to the Move-EC2AddressToVpc cmdlet. The Get-EC2Instance command gets an instance that is specified by instance ID, then returns the public IP address property of the instance.

```powershell
(Get-EC2Instance -Instance i-12345678).Instances.PublicIpAddress | Move-EC2AddressToVpc
```

- For API details, see [MoveAddressToVpc](https://aws.amazon.com/documentation/tools/powershell/cmdlet-reference/#MoveAddressToVpc) in *AWS Tools for PowerShell Cmdlet Reference*.

New-EC2Address

The following code example shows how to use New-EC2Address.

Example 1: This example allocates an Elastic IP address to use with an instance in a VPC.

```powershell
New-EC2Address -Domain Vpc
```

**Output:**

```
AllocationId   Domain   PublicIp
----------   ------   --------
eipalloc-12345678 vpc      198.51.100.2
```

Example 2: This example allocates an Elastic IP address to use with an instance in EC2-Classic.

```powershell
New-EC2Address
```

**Output:**

```
AllocationId   Domain   PublicIp
----------   ------   --------
```
For API details, see AllocateAddress in AWS Tools for PowerShell Cmdlet Reference.

**New-ECSCustomerGateway**

The following code example shows how to use New-ECSCustomerGateway.

**Tools for PowerShell**

**Example 1: This example creates the specified customer gateway.**

```powershell
New-ECSCustomerGateway -Type ipsec.1 -PublicIp 203.0.113.12 -BgpAsn 65534
```

**Output:**

- BgpAsn: 65534
- CustomerGatewayId: cgw-1a2b3c4d
- IpAddress: 203.0.113.12
- State: available
- Tags: {}
- Type: ipsec.1

For API details, see CreateCustomerGateway in AWS Tools for PowerShell Cmdlet Reference.

**New-ECSDhcpOption**

The following code example shows how to use New-ECSDhcpOption.

**Tools for PowerShell**

**Example 1: This example creates the specified set of DHCP options. The syntax used by this example requires PowerShell version 3 or later.**

```powershell
$options = @(
    @{Key="domain-name";Values="abc.local"},
    @{Key="domain-name-servers";Values="10.0.0.101","10.0.0.102"})
New-ECSDhcpOption -DhcPConfiguration $options
```

**Output:**
Example 2: With PowerShell version 2, you must use New-Object to create each DHCP option.

```
$option1 = New-Object Amazon.EC2.Model.DhcpConfiguration
$option1.Key = "domain-name"
$option1.Values = "abc.local"

$option2 = New-Object Amazon.EC2.Model.DhcpConfiguration
$option2.Key = "domain-name-servers"
$option2.Values = @("10.0.0.101","10.0.0.102")

New-EC2DhcpOption -DhcpConfiguration @($option1, $option2)
```

Output:

```
DhcpConfigurations                    DhcpOptionsId    Tags
------------------                    -------------    ----
{domain-name, domain-name-servers}    dopt-2a3b4c5d    {}
```

- For API details, see [CreateDhcpOptions](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**New-EC2FlowLog**

The following code example shows how to use New-EC2FlowLog.

**Tools for PowerShell**

**Example 1: This example creates EC2 flowlog for the subnet subnet-1d234567 to the cloud-watch-log named 'subnet1-log' for all 'REJECT' traffic using the perimssions of the 'Admin' role**

```
New-EC2FlowLog -ResourceId "subnet-1d234567" -LogDestinationType cloud-watch-logs -LogGroupName subnet1-log -TrafficType "REJECT" -ResourceType Subnet -DeliverLogsPermissionArn "arn:aws:iam::98765432109:role/Admin"
```

Output:
ClientToken | FlowLogIds | Unsuccessful
------------|-----------|--------------
m1VN2cxP3iB4qo//VUKl5EU6cF7gQL0xcqNefvjeTGw= | {fl-012fc34eed5678c9d} | {} |

- For API details, see CreateFlowLogs in AWS Tools for PowerShell Cmdlet Reference.

**New-EC2Host**

The following code example shows how to use New-EC2Host.

**Tools for PowerShell**

**Example 1:** This example allocates a Dedicated Host to your account for the given instance type and availability zone

New-EC2Host -AutoPlacement on -AvailabilityZone eu-west-1b -InstanceType m4.xlarge -Quantity 1

**Output:**

h-01e23f4cd567890f3

- For API details, see AllocateHosts in AWS Tools for PowerShell Cmdlet Reference.

**New-EC2HostReservation**

The following code example shows how to use New-EC2HostReservation.

**Tools for PowerShell**

**Example 1:** This example purchases the reservation offering hro-0c1f23456789d0ab with configurations that match those of your Dedicated Host h-01e23f4cd567890f1

New-EC2HostReservation -OfferingId hro-0c1f23456789d0ab HostIdSet h-01e23f4cd567890f1

**Output:**

ClientToken : 
CurrencyCode :

Amazon EC2
New-EC2Image

The following code example shows how to use New-EC2Image.

Tools for PowerShell

Example 1: This example creates an AMI with the specified name and description, from the specified instance. Amazon EC2 attempts to cleanly shut down the instance before creating the image, and restarts the instance on completion.

New-EC2Image -InstanceId i-12345678 -Name "my-web-server" -Description "My web server AMI"

Example 2: This example creates an AMI with the specified name and description, from the specified instance. Amazon EC2 creates the image without shutting down and restarting the instance; therefore, file system integrity on the created image can't be guaranteed.

New-EC2Image -InstanceId i-12345678 -Name "my-web-server" -Description "My web server AMI" -NoReboot $true

Example 3: This example creates an AMI with three volumes. The first volume is based on an Amazon EBS snapshot. The second volume is an empty 100 GiB Amazon EBS volume. The third volume is an instance store volume. The syntax used by this example requires PowerShell version 3 or higher.

```
$ebsBlock1 = @{SnapshotId="snap-1a2b3c4d"}
$ebsBlock2 = @{VolumeSize=100}
New-EC2Image -InstanceId i-12345678 -Name "my-web-server" -Description "My web server AMI" -BlockDeviceMapping @(@{DeviceName="/dev/sdf";Ebs=$ebsBlock1}, @{DeviceName="/dev/sdg";Ebs=$ebsBlock2}, @{DeviceName="/dev/sdc";VirtualName="ephemeral0"})
```

For API details, see CreateImage in AWS Tools for PowerShell Cmdlet Reference.
**New-EC2Instance**

The following code example shows how to use `New-EC2Instance`.

**Tools for PowerShell**

**Example 1:** This example launches a single instance of the specified AMI in EC2-Classic or a default VPC.

```powershell
New-EC2Instance -ImageId ami-12345678 -MinCount 1 -MaxCount 1 -InstanceType m3.medium -KeyName my-key-pair -SecurityGroup my-security-group
```

**Example 2:** This example launches a single instance of the specified AMI in a VPC.

```powershell
New-EC2Instance -ImageId ami-12345678 -MinCount 1 -MaxCount 1 -SubnetId subnet-12345678 -InstanceType t2.micro -KeyName my-key-pair -SecurityGroupId sg-12345678
```

**Example 3:** To add an EBS volume or an instance store volume, define a block device mapping and add it to the command. This example adds an instance store volume.

```powershell
$bdm = New-Object Amazon.EC2.Model.BlockDeviceMapping
$bdm.VirtualName = "ephemeral0"
$bdm.DeviceName = "/dev/sdf"

New-EC2Instance -ImageId ami-12345678 -BlockDeviceMapping $bdm ...
```

**Example 4:** To specify one of the current Windows AMIs, get its AMI ID using `Get-EC2ImageByName`. This example launches an instance from the current base AMI for Windows Server 2016.

```powershell
$ami = Get-EC2ImageByName WINDOWS_2016_BASE

New-EC2Instance -ImageId $ami.ImageId ...
```

**Example 5:** Launches an instance into the specified dedicated host environment.

```powershell
New-EC2Instance -ImageId ami-1a2b3c4d -InstanceType m4.large -KeyName my-key-pair -SecurityGroupId sg-1a2b3c4d -AvailabilityZone us-west-1a -Tenancy host -HostID h-1a2b3c4d5e6f1a2b3
```
Example 6: This request launches two instances and applies a tag with a key of webserver and a value of production to the instances. The request also applies a tag with a key of cost-center and a value of cc123 to the volumes that are created (in this case, the root volume for each instance).

```powershell
$tag1 = @{ Key="webserver"; Value="production" }
$tag2 = @{ Key="cost-center"; Value="cc123" }

$tagspec1 = new-object Amazon.EC2.Model.TagSpecification
$tagspec1.ResourceType = "instance"
$tagspec1.Tags.Add($tag1)

$tagspec2 = new-object Amazon.EC2.Model.TagSpecification
$tagspec2.ResourceType = "volume"
$tagspec2.Tags.Add($tag2)

New-EC2Instance -ImageId "ami-1a2b3c4d" -KeyName "my-key-pair" -MaxCount 2 -InstanceType "t2.large" -SubnetId "subnet-1a2b3c4d" -TagSpecification $tagspec1, $tagspec2
```

• For API details, see RunInstances in AWS Tools for PowerShell Cmdlet Reference.

New-EC2InstanceExportTask

The following code example shows how to use New-EC2InstanceExportTask.

Tools for PowerShell

Example 1: This example exports a stopped instance, i-0800b00a00EXAMPLE, as a virtual hard disk (VHD) to the S3 bucket testbucket-export-instances-2019. The target environment is Microsoft, and the region parameter is added because the instance is in the us-east-1 region, while the user's default AWS Region is not us-east-1. To get the status of the export task, copy the ExportTaskId value from the results of this command, then run Get-EC2ExportTask -ExportTaskId export_task_ID_from_results.

```powershell
New-EC2InstanceExportTask -InstanceId i-0800b00a00EXAMPLE -ExportToS3Task_DiskImageFormat VHD -ExportToS3Task_S3Bucket "testbucket-export-instances-2019" -TargetEnvironment Microsoft -Region us-east-1
```

Output:
New-EC2InternetGateway

The following code example shows how to use `New-EC2InternetGateway`.

Tools for PowerShell

**Example 1:** This example creates an Internet gateway.

```powershell
New-EC2InternetGateway
```

**Output:**

```
Attachments    InternetGatewayId    Tags
-----------    -----------------    ----
{}             igw-1a2b3c4d         {}
```

* For API details, see [CreateInternetGateway](#) in *AWS Tools for PowerShell Cmdlet Reference.*

New-EC2KeyPair

The following code example shows how to use `New-EC2KeyPair`.

Tools for PowerShell

**Example 1:** This example creates a key pair and captures the PEM-encoded RSA private key in a file with the specified name. When you are using PowerShell, the encoding must be set to ascii to generate a valid key. For more information, see Create, Display, and Delete Amazon EC2 Key Pairs (https://docs.aws.amazon.com/cli/latest/userguide/cli-services-ec2-keypairs.html) in the AWS Command Line Interface User Guide.
(New-EC2KeyPair -KeyName "my-key-pair").KeyMaterial | Out-File -Encoding ascii -FilePath C:\path\my-key-pair.pem

- For API details, see `CreateKeyPair` in *AWS Tools for PowerShell Cmdlet Reference*.

**New-EC2NetworkAcl**

The following code example shows how to use `New-EC2NetworkAcl`.

**Tools for PowerShell**

**Example 1:** This example creates a network ACL for the specified VPC.

```
New-EC2NetworkAcl -VpcId vpc-12345678
```

**Output:**

```
Associations : {}
IsDefault    : False
NetworkAclId : acl-12345678
Tags         : {}
VpcId        : vpc-12345678
```

- For API details, see `CreateNetworkAcl` in *AWS Tools for PowerShell Cmdlet Reference*.

**New-EC2NetworkAclEntry**

The following code example shows how to use `New-EC2NetworkAclEntry`.

**Tools for PowerShell**

**Example 1:** This example creates an entry for the specified network ACL. The rule allows inbound traffic from anywhere (0.0.0.0/0) on UDP port 53 (DNS) into any associated subnet.

```
New-EC2NetworkAclEntry -NetworkAclId acl-12345678 -Egress $false -RuleNumber 100 -Protocol 17 -PortRange_From 53 -PortRange_To 53 -CidrBlock 0.0.0.0/0 -RuleAction allow
```

- For API details, see `CreateNetworkAclEntry` in *AWS Tools for PowerShell Cmdlet Reference*. 
New-EC2NetworkInterface

The following code example shows how to use New-EC2NetworkInterface.

Tools for PowerShell

Example 1: This example creates the specified network interface.

```
New-EC2NetworkInterface -SubnetId subnet-1a2b3c4d -Description "my network interface" -Group sg-12345678 -PrivateIpAddress 10.0.0.17
```

Output:

```
Association : 
Attachment : 
AvailabilityZone : us-west-2c
Description : my network interface
Groups : {my-security-group}
MacAddress : 0a:72:bc:1a:cd:7f
NetworkInterfaceId : eni-12345678
OwnerId : 123456789012
PrivateDnsName : ip-10-0-0-17.us-west-2.compute.internal
PrivateIpAddress : 10.0.0.17
PrivateIpAddresses : {}
RequestId : 
RequesterManaged : False
SourceDestCheck : True
Status : pending
SubnetId : subnet-1a2b3c4d
TagSet : {}
VpcId : vpc-12345678
```

- For API details, see [CreateNetworkInterface](#) in AWS Tools for PowerShell Cmdlet Reference.

New-EC2PlacementGroup

The following code example shows how to use New-EC2PlacementGroup.

Tools for PowerShell

Example 1: This example creates a placement group with the specified name.
New-EC2PlacementGroup -GroupName my-placement-group -Strategy cluster

- For API details, see CreatePlacementGroup in AWS Tools for PowerShell Cmdlet Reference.

New-EC2Route

The following code example shows how to use New-EC2Route.

Tools for PowerShell

Example 1: This example creates the specified route for the specified route table. The route matches all traffic and sends it to the specified Internet gateway.

New-EC2Route -RouteTableId rtb-1a2b3c4d -DestinationCidrBlock 0.0.0.0/0 -GatewayId igw-1a2b3c4d

Output:

True

- For API details, see CreateRoute in AWS Tools for PowerShell Cmdlet Reference.

New-EC2RouteTable

The following code example shows how to use New-EC2RouteTable.

Tools for PowerShell

Example 1: This example creates a route table for the specified VPC.

New-EC2RouteTable -VpcId vpc-12345678

Output:

Associations : {}
PropagatingVgws : {}
Routes : {}
RouteTableId : rtb-1a2b3c4d
Tags : {}
VpcId : vpc-12345678
• For API details, see [CreateRouteTable](#) in AWS Tools for PowerShell Cmdlet Reference.

**New-EC2ScheduledInstance**

The following code example shows how to use `New-EC2ScheduledInstance`.

**Tools for PowerShell**

**Example 1: This example launches the specified Scheduled Instance.**

```
New-EC2ScheduledInstance -ScheduledInstanceId sci-1234-1234-1234-1234-123456789012 -InstanceCount 1 `  
-IamInstanceProfile_Name my-iam-role `  
-LaunchSpecification_ImageId ami-12345678 `  
-LaunchSpecification_InstanceType c4.large `  
-LaunchSpecification_SubnetId subnet-12345678`  
-LaunchSpecification_SecurityGroupId sg-12345678
```

• For API details, see [RunScheduledInstances](#) in AWS Tools for PowerShell Cmdlet Reference.

**New-EC2ScheduledInstancePurchase**

The following code example shows how to use `New-EC2ScheduledInstancePurchase`.

**Tools for PowerShell**

**Example 1: This example purchases a Scheduled Instance.**

```
$request = New-Object Amazon.EC2.Model.PurchaseRequest  
$request.InstanceCount = 1  
$request.PurchaseToken = "eyJ2IjoiMSIsInMiOjEsImMiOi..."  
New-EC2ScheduledInstancePurchase -PurchaseRequest $request
```

**Output:**

```
AvailabilityZone       : us-west-2b  
CreateDate            : 1/25/2016 1:43:38 PM  
HourlyPrice           : 0.095  
InstanceCount         : 1  
InstanceType          : c4.large  
NetworkPlatform       : EC2-VPC  
NextSlotStartTime     : 1/31/2016 1:00:00 AM
```
For API details, see [PurchaseScheduledInstances](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### New-EC2SecurityGroup

The following code example shows how to use `New-EC2SecurityGroup`.

#### Tools for PowerShell

**Example 1: This example creates a security group for the specified VPC.**

```powershell
New-EC2SecurityGroup -GroupName my-security-group -Description "my security group" -VpcId vpc-12345678
```

**Output:**

```
sg-12345678
```

**Example 2: This example creates a security group for EC2-Classic.**

```powershell
New-EC2SecurityGroup -GroupName my-security-group -Description "my security group"
```

**Output:**

```
sg-45678901
```

For API details, see [CreateSecurityGroup](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### New-EC2Snapshot

The following code example shows how to use `New-EC2Snapshot`.

---

---
Tools for PowerShell

Example 1: This example creates a snapshot of the specified volume.

```powershell
New-EC2Snapshot -VolumeId vol-12345678 -Description "This is a test"
```

Output:

```
DataEncryptionKeyId :
Description         : This is a test
Encrypted           : False
KmsKeyId            :
OwnerAlias          :
OwnerId             : 123456789012
Progress            :
SnapshotId          : snap-12345678
StartTime           : 12/22/2015 1:28:42 AM
State               : pending
StateMessage        :
Tags                : {}
VolumeId            : vol-12345678
VolumeSize          : 20
```

For API details, see [CreateSnapshot](#) in AWS Tools for PowerShell Cmdlet Reference.

---

New-EC2SpotDatafeedSubscription

The following code example shows how to use `New-EC2SpotDatafeedSubscription`.

Tools for PowerShell

Example 1: This example creates a Spot instance data feed.

```powershell
New-EC2SpotDatafeedSubscription -Bucket my-s3-bucket -Prefix spotdata
```

Output:

```
Bucket : my-s3-bucket
Fault  :
OwnerId : 123456789012
Prefix   : spotdata
```
State   : Active

- For API details, see CreateSpotDatafeedSubscription in AWS Tools for PowerShell Cmdlet Reference.

New-EC2Subnet

The following code example shows how to use New-EC2Subnet.

Tools for PowerShell

Example 1: This example creates a subnet with the specified CIDR.

New-EC2Subnet -VpcId vpc-12345678 -CidrBlock 10.0.0.0/24

Output:

AvailabilityZone : us-west-2c
AvailableIpAddressCount : 251
CidrBlock : 10.0.0.0/24
DefaultForAz : False
MapPublicIpOnLaunch : False
State : pending
SubnetId : subnet-1a2b3c4d
Tag : {}
VpcId : vpc-12345678

- For API details, see CreateSubnet in AWS Tools for PowerShell Cmdlet Reference.

New-EC2Tag

The following code example shows how to use New-EC2Tag.

Tools for PowerShell

Example 1: This example adds a single tag to the specified resource. The tag key is 'myTag' and the tag value is 'myTagValue'. The syntax used by this example requires PowerShell version 3 or higher.

New-EC2Tag -Resource i-12345678 -Tag @{ Key="myTag"; Value="myTagValue" }
Example 2: This example updates or adds the specified tags to the specified resource. The syntax used by this example requires PowerShell version 3 or higher.

```
New-EC2Tag -Resource i-12345678 -Tag @(
    @{ Key="myTag"; Value="newTagValue" },
    @{ Key="test"; Value="anotherTagValue" } )
```

Example 3: With PowerShell version 2, you must use New-Object to create the tag for the Tag parameter.

```
$tag = New-Object Amazon.EC2.Model.Tag
$tag.Key = "myTag"
$tag.Value = "myTagValue"

New-EC2Tag -Resource i-12345678 -Tag $tag
```

- For API details, see [CreateTags](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**New-EC2Volume**

The following code example shows how to use New-EC2Volume.

**Tools for PowerShell**

**Example 1: This example creates the specified volume.**

```
New-EC2Volume -Size 50 -AvailabilityZone us-west-2a -VolumeType gp2
```

**Output:**

```
Attachments          : {}
AvailabilityZone     : us-west-2a
CreateTime           : 12/22/2015 1:42:07 AM
Encrypted            : False
Iops                 : 150
KmsKeyId             : 
Size                 : 50
SnapshotId           : 
State                : creating
Tags                 : {}
VolumeId             : vol-12345678
VolumeType           : gp2
```
Example 2: This example request creates a volume and applies a tag with a key of stack and a value of production.

```powershell
$tag = @{ Key="stack"; Value="production" }

(tagspec = new-object Amazon.EC2.Model.TagSpecification
	$tagspec.ResourceType = "volume"
	$tagspec.Tags.Add($tag)

New-EC2Volume -Size 80 -AvailabilityZone "us-west-2a" -TagSpecification $tagspec
```

- For API details, see [CreateVolume](#) in *AWS Tools for PowerShell Cmdlet Reference*.

New-EC2Vpc

The following code example shows how to use New-EC2Vpc.

Tools for PowerShell

Example 1: This example creates a VPC with the specified CIDR. Amazon VPC also creates the following for the VPC: a default DHCP options set, a main route table, and a default network ACL.

```
New-EC2VPC -CidrBlock 10.0.0.0/16
```

Output:

```
CidrBlock       : 10.0.0.0/16
DhcpOptionsId   : dopt-1a2b3c4d
InstanceTenancy : default
IsDefault       : False
State           : pending
Tags            : {}
VpcId           : vpc-12345678
```

- For API details, see [CreateVpc](#) in *AWS Tools for PowerShell Cmdlet Reference*.

New-EC2VpcEndpoint

The following code example shows how to use New-EC2VpcEndpoint.
Tools for PowerShell

Example 1: This example create a new VPC Endpoint for the service com.amazonaws.eu-west-1.s3 in the VPC vpc-0fc1ff23f45b678eb

```powershell
New-EC2VpcEndpoint -ServiceName com.amazonaws.eu-west-1.s3 -VpcId vpc-0fc1ff23f45b678eb
```

Output:

```
ClientToken  VpcEndpoint
-----------  -----------
Amazon.EC2.Model.VpcEndpoint
```

- For API details, see CreateVpcEndpoint in AWS Tools for PowerShell Cmdlet Reference.

New-EC2VpnConnection

The following code example shows how to use New-EC2VpnConnection.

Tools for PowerShell

Example 1: This example creates a VPN connection between the specified virtual private gateway and the specified customer gateway. The output includes the configuration information that your network administrator needs, in XML format.

```powershell
New-EC2VpnConnection -Type ipsec.1 -CustomerGatewayId cgw-1a2b3c4d -VpnGatewayId vgw-1a2b3c4d
```

Output:

```
CustomerGatewayConfiguration : [XML document]
CustomerGatewayId : cgw-1a2b3c4d
Options : 
Routes : {}
State : pending
Tags : {}
Type : 
VgwTelemetry : {}
VpnConnectionId : vpn-12345678
```
Example 2: This example creates the VPN connection and captures the configuration in a file with the specified name.

```powershell
(New-EC2VpnConnection -CustomerGatewayId cgw-1a2b3c4d -VpnGatewayId vgw-1a2b3c4d).CustomerGatewayConfiguration | Out-File C:\path\vpn-configuration.xml
```

Example 3: This example creates a VPN connection, with static routing, between the specified virtual private gateway and the specified customer gateway.

```powershell
New-EC2VpnConnection -Type ipsec.1 -CustomerGatewayId cgw-1a2b3c4d -VpnGatewayId vgw-1a2b3c4d -Options_StaticRoutesOnly $true
```

- For API details, see [CreateVpnConnection](https://aws.amazon.com/tools/powershell/cmdlet-reference/) in *AWS Tools for PowerShell Cmdlet Reference*.

**New-EC2VpnConnectionRoute**

The following code example shows how to use `New-EC2VpnConnectionRoute`.

**Tools for PowerShell**

Example 1: This example creates the specified static route for the specified VPN connection.

```powershell
New-EC2VpnConnectionRoute -VpnConnectionId vpn-12345678 -DestinationCidrBlock 11.12.0.0/16
```

- For API details, see [CreateVpnConnectionRoute](https://aws.amazon.com/tools/powershell/cmdlet-reference/) in *AWS Tools for PowerShell Cmdlet Reference*.

**New-EC2VpnGateway**

The following code example shows how to use `New-EC2VpnGateway`.

**Tools for PowerShell**

Example 1: This example creates the specified virtual private gateway.

```powershell
New-EC2VpnGateway -Type ipsec.1
```
Output:

<table>
<thead>
<tr>
<th>AvailabilityZone :</th>
</tr>
</thead>
<tbody>
<tr>
<td>State              : available</td>
</tr>
<tr>
<td>Tags                : {}</td>
</tr>
<tr>
<td>Type                : ipsec.1</td>
</tr>
<tr>
<td>VpcAttachments      : {}</td>
</tr>
<tr>
<td>VpnGatewayId        : vgw-1a2b3c4d</td>
</tr>
</tbody>
</table>

- For API details, see [CreateVpnGateway](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Register-EC2Address

The following code example shows how to use Register-EC2Address.

**Tools for PowerShell**

**Example 1:** This example associates the specified Elastic IP address with the specified instance in a VPC.

```powershell
C:\> Register-EC2Address -InstanceId i-12345678 -AllocationId eipalloc-12345678
```

**Output:**

eipassoc-12345678

**Example 2:** This example associates the specified Elastic IP address with the specified instance in EC2-Classic.

```powershell
C:\> Register-EC2Address -InstanceId i-12345678 -PublicIp 203.0.113.17
```

- For API details, see [AssociateAddress](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Register-EC2DhcpOption

The following code example shows how to use Register-EC2DhcpOption.

**Tools for PowerShell**

**Example 1:** This example associates the specified DHCP options set with the specified VPC.
Register-EC2DhcpOption -DhcpOptionsId dopt-1a2b3c4d -VpcId vpc-12345678

Example 2: This example associates the default DHCP options set with the specified VPC.

Register-EC2DhcpOption -DhcpOptionsId default -VpcId vpc-12345678

• For API details, see AssociateDhcpOptions in AWS Tools for PowerShell Cmdlet Reference.

Register-EC2Image

The following code example shows how to use Register-EC2Image.

Tools for PowerShell

Example 1: This example registers an AMI using the specified manifest file in Amazon S3.

Register-EC2Image -ImageLocation my-s3-bucket/my-web-server-ami/image.manifest.xml -Name my-web-server-ami

• For API details, see RegisterImage in AWS Tools for PowerShell Cmdlet Reference.

Register-EC2PrivateIpAddress

The following code example shows how to use Register-EC2PrivateIpAddress.

Tools for PowerShell

Example 1: This example assigns the specified secondary private IP address to the specified network interface.

Register-EC2PrivateIpAddress -NetworkInterfaceId eni-1a2b3c4d -PrivateIpAddress 10.0.0.82

Example 2: This example creates two secondary private IP addresses and assigns them to the specified network interface.

Register-EC2PrivateIpAddress -NetworkInterfaceId eni-1a2b3c4d -SecondaryPrivateIpAddressCount 2
• For API details, see AssignPrivateIpAddresses in AWS Tools for PowerShell Cmdlet Reference.

Register-EC2RouteTable

The following code example shows how to use Register-EC2RouteTable.

Tools for PowerShell

Example 1: This example associates the specified route table with the specified subnet.

```
Register-EC2RouteTable -RouteTableId rtb-1a2b3c4d -SubnetId subnet-1a2b3c4d
```

Output:

```
rtbassoc-12345678
```

• For API details, see AssociateRouteTable in AWS Tools for PowerShell Cmdlet Reference.

Remove-EC2Address

The following code example shows how to use Remove-EC2Address.

Tools for PowerShell

Example 1: This example releases the specified Elastic IP address for instances in a VPC.

```
Remove-EC2Address -AllocationId eipalloc-12345678 -Force
```

Example 2: This example releases the specified Elastic IP address for instances in EC2-Classic.

```
Remove-EC2Address -PublicIp 198.51.100.2 -Force
```

• For API details, see ReleaseAddress in AWS Tools for PowerShell Cmdlet Reference.

Remove-EC2CapacityReservation

The following code example shows how to use Remove-EC2CapacityReservation.
Tools for PowerShell

**Example 1: This example cancels the capacity reservation cr-0c1f2345db6f7cdba**

Remove-EC2CapacityReservation -CapacityReservationId cr-0c1f2345db6f7cdba

**Output:**

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-EC2CapacityReservation (CancelCapacityReservation)"
on target "cr-0c1f2345db6f7cdba".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): y
True

- For API details, see [CancelCapacityReservation](https://aws.amazon.com/documentation/powerhell/#CancelCapacityReservation) in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-EC2CustomerGateway**

The following code example shows how to use Remove-EC2CustomerGateway.

Tools for PowerShell

**Example 1: This example deletes the specified customer gateway. You are prompted for confirmation before the operation proceeds, unless you also specify the Force parameter.**

Remove-EC2CustomerGateway -CustomerGatewayId cgw-1a2b3c4d

**Output:**

Confirm
Are you sure you want to perform this action?
Performing operation "Remove-EC2CustomerGateway (DeleteCustomerGateway)" on Target "cgw-1a2b3c4d".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): y

- For API details, see [DeleteCustomerGateway](https://aws.amazon.com/documentation/powerhell/#DeleteCustomerGateway) in *AWS Tools for PowerShell Cmdlet Reference*.
Remove-EC2DhcpOption

The following code example shows how to use Remove-EC2DhcpOption.

Tools for PowerShell

Example 1: This example deletes the specified DHCP options set. You are prompted for confirmation before the operation proceeds, unless you also specify the Force parameter.

```
Remove-EC2DhcpOption -DhcpOptionsId dopt-1a2b3c4d
```

Output:

```
Confirm
Are you sure you want to perform this action?
Performing operation "Remove-EC2DhcpOption (DeleteDhcpOptions)" on Target "dopt-1a2b3c4d".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):
```

- For API details, see [DeleteDhcpOptions](https://aws.amazon.com/) in [AWS Tools for PowerShell Cmdlet Reference](https://aws.amazon.com/).

Remove-EC2FlowLog

The following code example shows how to use Remove-EC2FlowLog.

Tools for PowerShell

Example 1: This example removes the given FlowLogId fl-01a2b3456a789c01

```
Remove-EC2FlowLog -FlowLogId fl-01a2b3456a789c01
```

Output:

```
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-EC2FlowLog (DeleteFlowLogs)" on target "fl-01a2b3456a789c01".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):
```

- For API details, see [DeleteFlowLogs](https://aws.amazon.com/) in [AWS Tools for PowerShell Cmdlet Reference](https://aws.amazon.com/).
Remove-EC2Host

The following code example shows how to use Remove-EC2Host.

Tools for PowerShell

Example 1: This example releases the given host ID h-0badafd1dcb2f3456

```
Remove-EC2Host -HostId h-0badafd1dcb2f3456
```

Output:

```
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-EC2Host (ReleaseHosts)" on target "h-0badafd1dcb2f3456".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): Y

Successful            Unsuccessful
----------            ------------
{h-0badafd1dcb2f3456} {}
```

- For API details, see [ReleaseHosts](#) in AWS Tools for PowerShell Cmdlet Reference.

Remove-EC2Instance

The following code example shows how to use Remove-EC2Instance.

Tools for PowerShell

Example 1: This example terminates the specified instance (the instance may be running or in 'stopped' state). The cmdlet will prompt for confirmation before proceeding; use the -Force switch to suppress the prompt.

```
Remove-EC2Instance -InstanceId i-12345678
```

Output:

```
CurrentState InstanceId PreviousState
----------  ---------  ------------
```
Remove-EC2InternetGateway

The following code example shows how to use Remove-EC2InternetGateway.

Tools for PowerShell

Example 1: This example deletes the specified Internet gateway. You are prompted for confirmation before the operation proceeds, unless you also specify the Force parameter.

Remove-EC2InternetGateway -InternetGatewayId igw-1a2b3c4d

Output:

Confirm
Are you sure you want to perform this action?
Performing operation "Remove-EC2InternetGateway (DeleteInternetGateway)" on Target "igw-1a2b3c4d".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):

• For API details, see DeleteInternetGateway in AWS Tools for PowerShell Cmdlet Reference.

Remove-EC2KeyPair

The following code example shows how to use Remove-EC2KeyPair.

Tools for PowerShell

Example 1: This example deletes the specified key pair. You are prompted for confirmation before the operation proceeds, unless you also specify the Force parameter.

Remove-EC2KeyPair -KeyName my-key-pair

Output:

Confirm
For API details, see \[DeleteKeyPair\] in \[AWS Tools for PowerShell Cmdlet Reference\].

**Remove-EC2NetworkAcl**

The following code example shows how to use Remove-EC2NetworkAcl.

**Tools for PowerShell**

**Example 1:** This example deletes the specified network ACL. You are prompted for confirmation before the operation proceeds, unless you also specify the Force parameter.

```
Remove-EC2NetworkAcl -NetworkAclId acl-12345678
```

**Output:**

```
Confirm
Are you sure you want to perform this action?
Performing operation "Remove-EC2NetworkAcl (DeleteNetworkAcl)" on Target "acl-12345678".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):
```

For API details, see \[DeleteNetworkAcl\] in \[AWS Tools for PowerShell Cmdlet Reference\].

**Remove-EC2NetworkAclEntry**

The following code example shows how to use Remove-EC2NetworkAclEntry.

**Tools for PowerShell**

**Example 1:** This example removes the specified rule from the specified network ACL. You are prompted for confirmation before the operation proceeds, unless you also specify the Force parameter.

```
Remove-EC2NetworkAclEntry -NetworkAclId acl-12345678 -Egress $false -RuleNumber 100
```
Output:

Confirm
Are you sure you want to perform this action?
Performing operation "Remove-EC2NetworkAclEntry (DeleteNetworkAclEntry)" on Target "acl-12345678".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):

- For API details, see [DeleteNetworkAclEntry](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

Remove-EC2NetworkInterface

The following code example shows how to use Remove-EC2NetworkInterface.

Tools for PowerShell

Example 1: This example deletes the specified network interface. You are prompted for confirmation before the operation proceeds, unless you also specify the Force parameter.

```
Remove-EC2NetworkInterface -NetworkInterfaceId eni-12345678
```

Output:

Confirm
Are you sure you want to perform this action?
Performing operation "Remove-EC2NetworkInterface (DeleteNetworkInterface)" on Target "eni-12345678".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):

- For API details, see [DeleteNetworkInterface](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

Remove-EC2PlacementGroup

The following code example shows how to use Remove-EC2PlacementGroup.

Tools for PowerShell

Example 1: This example deletes the specified placement group. You are prompted for confirmation before the operation proceeds, unless you also specify the Force parameter.
Remove-EC2PlacementGroup -GroupName my-placement-group

Output:

Confirm
Are you sure you want to perform this action?
Performing operation "Remove-EC2PlacementGroup (DeletePlacementGroup)" on Target "my-placement-group".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):

• For API details, see [DeletePlacementGroup](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

Remove-EC2Route

The following code example shows how to use Remove-EC2Route.

Tools for PowerShell

Example 1: This example deletes the specified route from the specified route table. You are prompted for confirmation before the operation proceeds, unless you also specify the Force parameter.

Remove-EC2Route -RouteTableId rtb-1a2b3c4d -DestinationCidrBlock 0.0.0.0/0

Output:

Confirm
Are you sure you want to perform this action?
Performing operation "Remove-EC2Route (DeleteRoute)" on Target "rtb-1a2b3c4d".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):

• For API details, see [DeleteRoute](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

Remove-EC2RouteTable

The following code example shows how to use Remove-EC2RouteTable.
Tools for PowerShell

**Example 1:** This example deletes the specified route table. You are prompted for confirmation before the operation proceeds, unless you also specify the Force parameter.

```
Remove-EC2RouteTable -RouteTableId rtb-1a2b3c4d
```

**Output:**

```
Confirm
Are you sure you want to perform this action?
Performing operation "Remove-EC2RouteTable (DeleteRouteTable)" on Target "rtb-1a2b3c4d".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):
```

- For API details, see [DeleteRouteTable](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-EC2SecurityGroup**

The following code example shows how to use Remove-EC2SecurityGroup.

Tools for PowerShell

**Example 1:** This example deletes the specified security group for EC2-VPC. You are prompted for confirmation before the operation proceeds, unless you also specify the Force parameter.

```
Remove-EC2SecurityGroup -GroupId sg-12345678
```

**Output:**

```
Confirm
Are you sure you want to perform this action?
Performing operation "Remove-EC2SecurityGroup (DeleteSecurityGroup)" on Target "sg-12345678".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):
```

**Example 2:** This example deletes the specified security group for EC2-Classic.
Remove-EC2SecurityGroup -GroupName my-security-group -Force

- For API details, see `DeleteSecurityGroup` in *AWS Tools for PowerShell Cmdlet Reference*.

### Remove-EC2Snapshot

The following code example shows how to use Remove-EC2Snapshot.

**Tools for PowerShell**

**Example 1:** This example deletes the specified snapshot. You are prompted for confirmation before the operation proceeds, unless you also specify the Force parameter.

```
Remove-EC2Snapshot -SnapshotId snap-12345678
```

**Output:**

```
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-EC2Snapshot (DeleteSnapshot)" on target "snap-12345678".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):
```

- For API details, see `DeleteSnapshot` in *AWS Tools for PowerShell Cmdlet Reference*.

### Remove-EC2SpotDatafeedSubscription

The following code example shows how to use Remove-EC2SpotDatafeedSubscription.

**Tools for PowerShell**

**Example 1:** This example deletes your Spot instance data feed. You are prompted for confirmation before the operation proceeds, unless you also specify the Force parameter.

```
Remove-EC2SpotDatafeedSubscription
```

**Output:**

```
Confirm
```
Are you sure you want to perform this action?
Performing operation "Remove-EC2SpotDatafeedSubscription (DeleteSpotDatafeedSubscription)" on Target "".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):

- For API details, see DeleteSpotDatafeedSubscription in AWS Tools for PowerShell Cmdlet Reference.

Remove-EC2Subnet

The following code example shows how to use Remove-EC2Subnet.

Tools for PowerShell

Example 1: This example deletes the specified subnet. You are prompted for confirmation before the operation proceeds, unless you also specify the Force parameter.

Remove-EC2Subnet -SubnetId subnet-1a2b3c4d

Output:

Confirm
Are you sure you want to perform this action?
Performing operation "Remove-EC2Subnet (DeleteSubnet)" on Target "subnet-1a2b3c4d".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):

- For API details, see DeleteSubnet in AWS Tools for PowerShell Cmdlet Reference.

Remove-EC2Tag

The following code example shows how to use Remove-EC2Tag.

Tools for PowerShell

Example 1: This example deletes the specified tag from the specified resource, regardless of the tag value. The syntax used by this example requires PowerShell version 3 or later.

Remove-EC2Tag -Resource i-12345678 -Tag @{ Key="myTag" } -Force
Example 2: This example deletes the specified tag from the specified resource, but only if the tag value matches. The syntax used by this example requires PowerShell version 3 or later.

```
Remove-EC2Tag -Resource i-12345678 -Tag @{ Key="myTag";Value="myTagValue" } -Force
```

Example 3: This example deletes the specified tag from the specified resource, regardless of the tag value.

```
$tag = New-Object Amazon.EC2.Model.Tag
$tag.Key = "myTag"
Remove-EC2Tag -Resource i-12345678 -Tag $tag -Force
```

Example 4: This example deletes the specified tag from the specified resource, but only if the tag value matches.

```
$tag = New-Object Amazon.EC2.Model.Tag
$tag.Key = "myTag"
$tag.Value = "myTagValue"
Remove-EC2Tag -Resource i-12345678 -Tag $tag -Force
```

- For API details, see [DeleteTags](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Remove-EC2Volume

The following code example shows how to use Remove-EC2Volume.

**Tools for PowerShell**

Example 1: This example detaches the specified volume. You are prompted for confirmation before the operation proceeds, unless you also specify the Force parameter.

```
Remove-EC2Volume -VolumeId vol-12345678
```

**Output:**

```
Confirm
Are you sure you want to perform this action?
```
Performing the operation "Remove-EC2Volume (DeleteVolume)" on target "vol-12345678". [Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): 

- For API details, see DeleteVolume in AWS Tools for PowerShell Cmdlet Reference.

Remove-EC2Vpc

The following code example shows how to use Remove-EC2Vpc.

Tools for PowerShell

Example 1: This example deletes the specified VPC. You are prompted for confirmation before the operation proceeds, unless you also specify the Force parameter.

Remove-EC2Vpc -VpcId vpc-12345678

Output:

Confirm
Are you sure you want to perform this action?
Performing operation "Remove-EC2Vpc (DeleteVpc)" on Target "vpc-12345678". [Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): 

- For API details, see DeleteVpc in AWS Tools for PowerShell Cmdlet Reference.

Remove-EC2VpnConnection

The following code example shows how to use Remove-EC2VpnConnection.

Tools for PowerShell

Example 1: This example deletes the specified VPN connection. You are prompted for confirmation before the operation proceeds, unless you also specify the Force parameter.

Remove-EC2VpnConnection -VpnConnectionId vpn-12345678

Output:

Confirm
Are you sure you want to perform this action?
Performing operation "Remove-EC2VpnConnection (DeleteVpnConnection)" on Target "vpn-12345678".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):  

- For API details, see [DeleteVpnConnection](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

**Remove-EC2VpnConnectionRoute**

The following code example shows how to use Remove-EC2VpnConnectionRoute.

**Tools for PowerShell**

**Example 1:** This example removes the specified static route from the specified VPN connection. You are prompted for confirmation before the operation proceeds, unless you also specify the Force parameter.

```powershell
Remove-EC2VpnConnectionRoute -VpnConnectionId vpn-12345678 -DestinationCidrBlock 11.12.0.0/16
```

**Output:**

Confirm
Are you sure you want to perform this action?
Performing operation "Remove-EC2VpnConnectionRoute (DeleteVpnConnectionRoute)" on Target "vpn-12345678".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):  

- For API details, see [DeleteVpnConnectionRoute](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

**Remove-EC2VpnGateway**

The following code example shows how to use Remove-EC2VpnGateway.

**Tools for PowerShell**

**Example 1:** This example deletes the specified virtual private gateway. You are prompted for confirmation before the operation proceeds, unless you also specify the Force parameter.
Remove-EC2VpnGateway -VpnGatewayId vgw-1a2b3c4d

Output:

Confirm
Are you sure you want to perform this action?
Performing operation "Remove-EC2VpnGateway (DeleteVpnGateway)" on Target "vgw-1a2b3c4d".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):

- For API details, see DeleteVpnGateway in AWS Tools for PowerShell Cmdlet Reference.

Request-EC2SpotFleet

The following code example shows how to use Request-EC2SpotFleet.

Tools for PowerShell

Example 1: This example creates a Spot fleet request in the Availability Zone with the lowest price for the specified instance type. If your account supports EC2-VPC only, the Spot fleet launches the instances in the lowest-priced Availability Zone that has a default subnet. If your account supports EC2-Classic, the Spot fleet launches the instances in EC2-Classic in the lowest-priced Availability Zone. Note that the price you pay will not exceed the specified Spot price for the request.

```powershell
$sg = New-Object Amazon.EC2.Model.GroupIdentifier
$sg.GroupId = "sg-12345678"
$lc = New-Object Amazon.EC2.Model.SpotFleetLaunchSpecification
$lc.ImageId = "ami-12345678"
$lc.InstanceType = "m3.medium"
$lc.SecurityGroups.Add($sg)
Request-EC2SpotFleet -SpotFleetRequestConfig_SpotPrice 0.04 `-SpotFleetRequestConfig_TargetCapacity 2 `-SpotFleetRequestConfig_IamFleetRole arn:aws:iam::123456789012:role/my-spot-fleet-role `-SpotFleetRequestConfig_LaunchSpecification $lc
```

- For API details, see RequestSpotFleet in AWS Tools for PowerShell Cmdlet Reference.
Request-EC2SpotInstance

The following code example shows how to use Request-EC2SpotInstance.

Tools for PowerShell

Example 1: This example requests a one-time Spot instance in the specified subnet. Note that the security group must be created for the VPC that contains the specified subnet, and it must be specified by ID using the network interface. When you specify a network interface, you must include the subnet ID using the network interface.

```powershell
$n = New-Object Amazon.EC2.Model.InstanceNetworkInterfaceSpecification
$n.DeviceIndex = 0
$n.SubnetId = "subnet-12345678"
$n.Groups.Add("sg-12345678")
Request-EC2SpotInstance -InstanceCount 1 -SpotPrice 0.050 -Type one-time ` -IamInstanceProfile_Arn arn:aws:iam::123456789012:instance-profile/my-iam-role ` -LaunchSpecification_ImageId ami-12345678 ` -LaunchSpecification_InstanceType m3.medium ` -LaunchSpecification_NetworkInterface $n
```

Output:

```
ActualBlockHourlyPrice : 
AvailabilityZoneGroup   : 
BlockDurationMinutes   : 0
CreateTime             : 12/26/2015 7:44:10 AM
Fault                  : 
InstanceId             : 
LaunchedAvailabilityZone : 
LaunchGroup            : 
ProductDescription     : Linux/UNIX
SpotInstanceRequestId  : sir-12345678
SpotPrice              : 0.050000
State                  : open
Status                 : Amazon.EC2.Model.SpotInstanceStatus
Tags                   : {} 
Type                   : one-time
```

- For API details, see RequestSpotInstances in AWS Tools for PowerShell Cmdlet Reference.
**Reset-EC2ImageAttribute**

The following code example shows how to use Reset-EC2ImageAttribute.

**Tools for PowerShell**

**Example 1:** This example resets the 'launchPermission' attribute to its default value. By default, AMIs are private.

```
Reset-EC2ImageAttribute -ImageId ami-12345678 -Attribute launchPermission
```

- For API details, see [ResetImageAttribute](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Reset-EC2InstanceAttribute**

The following code example shows how to use Reset-EC2InstanceAttribute.

**Tools for PowerShell**

**Example 1:** This example resets the 'sriovNetSupport' attribute for the specified instance.

```
Reset-EC2InstanceAttribute -InstanceId i-12345678 -Attribute sriovNetSupport
```

**Example 2:** This example resets the 'ebsOptimized' attribute for the specified instance.

```
Reset-EC2InstanceAttribute -InstanceId i-12345678 -Attribute ebsOptimized
```

**Example 3:** This example resets the 'sourceDestCheck' attribute for the specified instance.

```
Reset-EC2InstanceAttribute -InstanceId i-12345678 -Attribute sourceDestCheck
```

**Example 4:** This example resets the 'disableApiTermination' attribute for the specified instance.

```
Reset-EC2InstanceAttribute -InstanceId i-12345678 -Attribute disableApiTermination
```

**Example 5:** This example resets the 'instanceInitiatedShutdownBehavior' attribute for the specified instance.
Reset-EC2InstanceAttribute -InstanceId i-12345678 -Attribute instanceInitiatedShutdownBehavior

- For API details, see ResetInstanceAttribute in AWS Tools for PowerShell Cmdlet Reference.

**Reset-EC2NetworkInterfaceAttribute**

The following code example shows how to use Reset-EC2NetworkInterfaceAttribute.

**Tools for PowerShell**

**Example 1:** This example resets source/destination checking for the specified network interface.

Reset-EC2NetworkInterfaceAttribute -NetworkInterfaceId eni-1a2b3c4d -SourceDestCheck

- For API details, see ResetNetworkInterfaceAttribute in AWS Tools for PowerShell Cmdlet Reference.

**Reset-EC2SnapshotAttribute**

The following code example shows how to use Reset-EC2SnapshotAttribute.

**Tools for PowerShell**

**Example 1:** This example resets the specified attribute of the specified snapshot.

Reset-EC2SnapshotAttribute -SnapshotId snap-12345678 -Attribute CreateVolumePermission

- For API details, see ResetSnapshotAttribute in AWS Tools for PowerShell Cmdlet Reference.

**Restart-EC2Instance**

The following code example shows how to use Restart-EC2Instance.

**Tools for PowerShell**

**Example 1:** This example reboots the specified instance.
Restart-EC2Instance -InstanceId i-12345678

- For API details, see RebootInstances in AWS Tools for PowerShell Cmdlet Reference.

Revoke-EC2SecurityGroupEgress

The following code example shows how to use Revoke-EC2SecurityGroupEgress.

Tools for PowerShell

Example 1: This example removes the rule for the specified security group for EC2-VPC. This revokes access to the specified IP address range on TCP port 80. The syntax used by this example requires PowerShell version 3 or higher.

```powershell
$ip = @{ IpProtocol="tcp"; FromPort="80"; ToPort="80"; IpRanges="203.0.113.0/24" }
Revoke-EC2SecurityGroupEgress -GroupId sg-12345678 -IpPermission $ip
```

Example 2: With PowerShell version 2, you must use New-Object to create the IpPermission object.

```powershell
$ip = New-Object Amazon.EC2.Model.IpPermission
$ip.IpProtocol = "tcp"
$ip.FromPort = 80
$ip.ToPort = 80
$ip.IpRanges.Add("203.0.113.0/24")
Revoke-EC2SecurityGroupEgress -GroupId sg-12345678 -IpPermission $ip
```

Example 3: This example revokes access to the specified source security group on TCP port 80.

```powershell
$ug = New-Object Amazon.EC2.Model.UserIdGroupPair
$ug.GroupId = "sg-1a2b3c4d"
$ug.UserId = "123456789012"
Revoke-EC2SecurityGroupEgress -GroupId sg-12345678 -IpPermission @( @{ IpProtocol="tcp"; FromPort="80"; ToPort="80"; UserIdGroupPairs=$ug } )
```

- For API details, see RevokeSecurityGroupEgress in AWS Tools for PowerShell Cmdlet Reference.
Revoke-EC2SecurityGroupIngress

The following code example shows how to use Revoke-EC2SecurityGroupIngress.

**Tools for PowerShell**

**Example 1:** This example revokes access to TCP port 22 from the specified address range for the specified security group for EC2-VPC. Note that you must identify security groups for EC2-VPC using the security group ID not the security group name. The syntax used by this example requires PowerShell version 3 or higher.

```powershell
$ip = @{ IpProtocol="tcp"; FromPort="22"; ToPort="22"; IpRanges="203.0.113.0/24" }
Revoke-EC2SecurityGroupIngress -GroupId sg-12345678 -IpPermission $ip
```

**Example 2:** With PowerShell version 2, you must use New-Object to create the IpPermission object.

```powershell
$ip = New-Object Amazon.EC2.Model.IpPermission
$ip.IpProtocol = "tcp"
$ip.FromPort = 22
$ip.ToPort = 22
$ip.IpRanges.Add("203.0.113.0/24")
Revoke-EC2SecurityGroupIngress -GroupId sg-12345678 -IpPermission $ip
```

**Example 3:** This example revokes access to TCP port 22 from the specified address range for the specified security group for EC2-Classic. The syntax used by this example requires PowerShell version 3 or higher.

```powershell
$ip = @{ IpProtocol="tcp"; FromPort="22"; ToPort="22"; IpRanges="203.0.113.0/24" }
Revoke-EC2SecurityGroupIngress -GroupName "my-security-group" - IpPermission $ip
```

**Example 4:** With PowerShell version 2, you must use New-Object to create the IpPermission object.

```powershell
$ip = New-Object Amazon.EC2.Model.IpPermission
$ip.IpProtocol = "tcp"
$ip.FromPort = 22
```
$ip.ToPort = 22
$ip.IpRanges.Add("203.0.113.0/24")

Revoke-EC2SecurityGroupIngress -GroupName "my-security-group" -IpPermission $ip

- For API details, see RevokeSecurityGroupIngress in AWS Tools for PowerShell Cmdlet Reference.

Send-EC2InstanceStatus

The following code example shows how to use Send-EC2InstanceStatus.

Tools for PowerShell

Example 1: This example reports status feedback for the specified instance.

Send-EC2InstanceStatus -Instance i-12345678 -Status impaired -ReasonCode unresponsive

- For API details, see ReportInstanceStatus in AWS Tools for PowerShell Cmdlet Reference.

Set-EC2NetworkAclAssociation

The following code example shows how to use Set-EC2NetworkAclAssociation.

Tools for PowerShell

Example 1: This example associates the specified network ACL with the subnet for the specified network ACL association.

Set-EC2NetworkAclAssociation -NetworkAclId acl-12345678 -AssociationId aclassoc-1a2b3c4d

Output:

aclassoc-87654321

- For API details, see ReplaceNetworkAclAssociation in AWS Tools for PowerShell Cmdlet Reference.
Set-EC2NetworkAclEntry

The following code example shows how to use Set-EC2NetworkAclEntry.

Tools for PowerShell

Example 1: This example replaces the specified entry for the specified network ACL. The new rule allows inbound traffic from the specified address to any associated subnet.

```
Set-EC2NetworkAclEntry -NetworkAclId acl-12345678 -Egress $false -RuleNumber 100 -Protocol 17 -PortRange_From 53 -PortRange_To 53 -CidrBlock 203.0.113.12/24 -RuleAction allow
```

- For API details, see ReplaceNetworkAclEntry in AWS Tools for PowerShell Cmdlet Reference.

Set-EC2Route

The following code example shows how to use Set-EC2Route.

Tools for PowerShell

Example 1: This example replaces the specified route for the specified route table. The new route sends the specified traffic to the specified virtual private gateway.

```
Set-EC2Route -RouteTableId rtb-1a2b3c4d -DestinationCidrBlock 10.0.0.0/24 -GatewayId vgw-1a2b3c4d
```

- For API details, see ReplaceRoute in AWS Tools for PowerShell Cmdlet Reference.

Set-EC2RouteTableAssociation

The following code example shows how to use Set-EC2RouteTableAssociation.

Tools for PowerShell

Example 1: This example associates the specified route table with the subnet for the specified route table association.

```
Set-EC2RouteTableAssociation -RouteTableId rtb-1a2b3c4d -AssociationId rtbassoc-12345678
```
Output:

rtbassoc-87654321

- For API details, see [ReplaceRouteTableAssociation](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

**Start-EC2Instance**

The following code example shows how to use `Start-EC2Instance`.

**Tools for PowerShell**

**Example 1:** This example starts the specified instance.

```
Start-EC2Instance -InstanceId i-12345678
```

**Output:**

<table>
<thead>
<tr>
<th>CurrentState</th>
<th>InstanceId</th>
<th>PreviousState</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon.EC2.Model.InstanceState</td>
<td>i-12345678</td>
<td>Amazon.EC2.Model.InstanceState</td>
</tr>
</tbody>
</table>

**Example 2:** This example starts the specified instances.

```
@("i-12345678", "i-76543210") | Start-EC2Instance
```

**Example 3:** This example starts the set of instances that are currently stopped. The Instance objects returned by Get-EC2Instance are piped to Start-EC2Instance. The syntax used by this example requires PowerShell version 3 or higher.

```
(Get-EC2Instance -Filter @{ Name="instance-state-name"; Values="stopped"}).Instances | Start-EC2Instance
```

**Example 4:** With PowerShell version 2, you must use New-Object to create the filter for the Filter parameter.

```
$filter = New-Object Amazon.EC2.Model.Filter
$filter.Name = "instance-state-name"
```
$filter.Values = "stopped"

(Get-EC2Instance -Filter $filter).Instances | Start-EC2Instance

- For API details, see [StartInstances](https://docs.aws.amazon.com/powershell/latest/reference/startinstances.html) in *AWS Tools for PowerShell Cmdlet Reference*.

### Start-EC2InstanceMonitoring

The following code example shows how to use `Start-EC2InstanceMonitoring`.

**Tools for PowerShell**

**Example 1:** This example enables detailed monitoring for the specified instance.

```powershell
Start-EC2InstanceMonitoring -InstanceId i-12345678
```

**Output:**

<table>
<thead>
<tr>
<th>InstanceId</th>
<th>Monitoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>i-12345678</td>
<td>Amazon.EC2.Model.Monitoring</td>
</tr>
</tbody>
</table>

- For API details, see [MonitorInstances](https://docs.aws.amazon.com/powershell/latest/reference/monitorinstances.html) in *AWS Tools for PowerShell Cmdlet Reference*.

### Stop-EC2ImportTask

The following code example shows how to use `Stop-EC2ImportTask`.

**Tools for PowerShell**

**Example 1:** This example cancels the specified import task (either snapshot or image import). If required, a reason can be providing using the `-CancelReason` parameter.

```powershell
Stop-EC2ImportTask -ImportTaskId import-ami-abcdefgh
```

- For API details, see [CancelImportTask](https://docs.aws.amazon.com/powershell/latest/reference/cancelimporttask.html) in *AWS Tools for PowerShell Cmdlet Reference*.

### Stop-EC2Instance

The following code example shows how to use `Stop-EC2Instance`.
Tools for PowerShell

Example 1: This example stops the specified instance.

Stop-EC2Instance -InstanceId i-12345678

Output:

<table>
<thead>
<tr>
<th>CurrentState</th>
<th>InstanceId</th>
<th>PreviousState</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon.EC2.Model.InstanceState</td>
<td>i-12345678</td>
<td>Amazon.EC2.Model.InstanceState</td>
</tr>
</tbody>
</table>

• For API details, see StopInstances in AWS Tools for PowerShell Cmdlet Reference.

Stop-EC2InstanceMonitoring

The following code example shows how to use Stop-EC2InstanceMonitoring.

Tools for PowerShell

Example 1: This example disables detailed monitoring for the specified instance.

Stop-EC2InstanceMonitoring -InstanceId i-12345678

Output:

<table>
<thead>
<tr>
<th>InstanceId</th>
<th>Monitoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>i-12345678</td>
<td>Amazon.EC2.Model.Monitoring</td>
</tr>
</tbody>
</table>

• For API details, see UnmonitorInstances in AWS Tools for PowerShell Cmdlet Reference.

Stop-EC2SpotFleetRequest

The following code example shows how to use Stop-EC2SpotFleetRequest.

Tools for PowerShell

Example 1: This example cancels the specified Spot fleet request and terminates the associated Spot instances.
Example 2: This example cancels the specified Spot fleet request without terminating the associated Spot instances.

Stop-EC2SpotFleetRequest -SpotFleetRequestId sfr-73fbd2ce-aa30-494c-8788-1cee4EXAMPLE -TerminateInstance $false

- For API details, see CancelSpotFleetRequests in AWS Tools for PowerShell Cmdlet Reference.

Stop-EC2SpotInstanceRequest

The following code example shows how to use Stop-EC2SpotInstanceRequest.

Tools for PowerShell

Example 1: This example cancels the specified Spot instance request.

Stop-EC2SpotInstanceRequest -SpotInstanceRequestId sir-12345678

Output:

<table>
<thead>
<tr>
<th>SpotInstanceRequestId</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>sir-12345678</td>
<td>cancelled</td>
</tr>
</tbody>
</table>

- For API details, see CancelSpotInstanceRequests in AWS Tools for PowerShell Cmdlet Reference.

Unregister-EC2Address

The following code example shows how to use Unregister-EC2Address.

Tools for PowerShell

Example 1: This example disassociates the specified Elastic IP address from the specified instance in a VPC.
Unregister-EC2Address -AssociationId eipassoc-12345678

Example 2: This example disassociates the specified Elastic IP address from the specified instance in EC2-Classic.

Unregister-EC2Address -PublicIp 203.0.113.17

- For API details, see DisassociateAddress in AWS Tools for PowerShell Cmdlet Reference.

Unregister-EC2Image

The following code example shows how to use Unregister-EC2Image.

Tools for PowerShell

   Example 1: This example deregisters the specified AMI.

Unregister-EC2Image -ImageId ami-12345678

- For API details, see DeregisterImage in AWS Tools for PowerShell Cmdlet Reference.

Unregister-EC2PrivateIpAddress

The following code example shows how to use Unregister-EC2PrivateIpAddress.

Tools for PowerShell

   Example 1: This example unassigns the specified private IP address from the specified network interface.

Unregister-EC2PrivateIpAddress -NetworkInterfaceId eni-1a2b3c4d -PrivateIpAddress 10.0.0.82

- For API details, see UnassignPrivateIpAddresses in AWS Tools for PowerShell Cmdlet Reference.

Unregister-EC2RouteTable

The following code example shows how to use Unregister-EC2RouteTable.
Example 1: This example removes the specified association between a route table and a subnet.

Unregister-EC2RouteTable -AssociationId rtbassoc-1a2b3c4d

- For API details, see DisassociateRouteTable in AWS Tools for PowerShell Cmdlet Reference.

Amazon ECR examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with Amazon ECR.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics
- Actions

Actions

Get-ECRLoginCommand

The following code example shows how to use Get-ECRLoginCommand.

Tools for PowerShell

Example 1: Returns a PSObject containing login information that can be used to authenticate to any Amazon ECR registry that your IAM principal has access to. The credentials and region endpoint required for the call to obtain the authorization token are obtained from the shell defaults (set up by the Set-AWSCredential/Set-
DefaultAWSRegion or Initialize-AWSDefaultConfiguration cmdlets). You can use the Command property with Invoke-Expression to log in to the specified registry or use the returned credentials in other tools requiring login.

Get-ECRLoginCommand

Output:

Username      : AWS
Password      : eyJwYXlsb2Fk...kRBVEFFS0VZIn0=
ProxyEndpoint : https://123456789012.dkr.ecr.us-west-2.amazonaws.com
Endpoint      : https://123456789012.dkr.ecr.us-west-2.amazonaws.com
ExpiresAt     : 9/26/2017 6:08:23 AM
Command       : docker login --username AWS --password eyJwYXlsb2Fk...kRBVEFFS0VZIn0= https://123456789012.dkr.ecr.us-west-2.amazonaws.com

Example 2: Retrieves a PSObject containing login information that you use as an input to a docker login command. You can specify any Amazon ECR registry URI to authenticate to as long as your IAM principal has access to that registry.

(Get-ECRLoginCommand).Password | docker login --username AWS --password-stdin 012345678910.dkr.ecr.us-east-1.amazonaws.com

• For API details, see [Get-ECRLoginCommand](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

Amazon ECS examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with Amazon ECS.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.
Actions

Get-ECSClusterDetail

The following code example shows how to use Get-ECSClusterDetail.

**Example 1:** This cmdlet describes one or more of your ECS clusters.

```
Get-ECSClusterDetail -Cluster "LAB-ECS-CL" -Include SETTINGS | Select-Object *
```

**Output:**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoggedAt</td>
<td>12/27/2019 9:27:41 PM</td>
</tr>
<tr>
<td>Clusters</td>
<td>{LAB-ECS-CL}</td>
</tr>
<tr>
<td>Failures</td>
<td>{}</td>
</tr>
<tr>
<td>ResponseMetadata</td>
<td>Amazon.Runtime.ResponseMetadata</td>
</tr>
<tr>
<td>ContentLength</td>
<td>396</td>
</tr>
<tr>
<td>HttpStatusCode</td>
<td>OK</td>
</tr>
</tbody>
</table>

- For API details, see [DescribeClusters](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-ECSClusterList

The following code example shows how to use Get-ECSClusterList.

**Tools for PowerShell**

**Example 1:** This cmdlet returns a list of existing ECS clusters.

```
Get-ECSClusterList
```

**Output:**

```
```

- For API details, see ListClusters in AWS Tools for PowerShell Cmdlet Reference.

**Get-ECSClusterService**

The following code example shows how to use Get-ECSClusterService.

**Tools for PowerShell**

**Example 1:** This example lists all services running in your default cluster.

Get-ECSClusterService

**Example 2:** This example lists all services running in the specified cluster.

Get-ECSClusterService -Cluster myCluster

**Example 3:** This example lists the services running in the specified cluster, fetching a maximum of 10 service details at a time.

```powershell
$nextToken = $null
do {
    Get-ECSClusterService -Cluster myCluster -MaxResult 10 -NextToken $nextToken
    $nextToken = $AWSHistory.LastServiceResponse.NextToken
} while ($nextToken -ne $null)
```

- For API details, see ListServices in AWS Tools for PowerShell Cmdlet Reference.

**Get-ECSService**

The following code example shows how to use Get-ECSService.

**Tools for PowerShell**

**Example 1:** This example shows how to retrieve details of a specific service from your default cluster.

Get-ECSService -Service my-http-service
Example 2: This example shows how to retrieve details of a specific service running in the named cluster.

```
Get-ECSService -Cluster myCluster -Service my-http-service
```

- For API details, see `DescribeServices` in *AWS Tools for PowerShell Cmdlet Reference*.

New-ECSCluster

The following code example shows how to use New-ECSCluster.

**Tools for PowerShell**

Example 1: This cmdlet creates a new Amazon ECS cluster.

```
New-ECSCluster -ClusterName "LAB-ECS-CL" -Setting @{Name="containerInsights"; Value="enabled"}
```

Output:

```
ActiveServicesCount : 0
Attachments         : {}
AttachmentsStatus   : {}
CapacityProviders   : {}
ClusterName         : LAB-ECS-CL
DefaultCapacityProviderStrategy : {}
PendingTasksCount   : 0
RegisteredContainerInstancesCount : 0
RunningTasksCount   : 0
Settings            : {containerInsights}
Statistics          : {}
Status              : ACTIVE
Tags                : {}
```

- For API details, see `CreateCluster` in *AWS Tools for PowerShell Cmdlet Reference*.

New-ECSService

The following code example shows how to use New-ECSService.
Tools for PowerShell

Example 1: This example command creates a service in your default cluster called `ecs-simple-service`. The service uses the `ecs-demo` task definition and it maintains 10 instantiations of that task.

New-ECSService -ServiceName ecs-simple-service -TaskDefinition ecs-demo -DesiredCount 10

Example 2: This example command creates a service behind a load balancer in your default cluster called `ecs-simple-service`. The service uses the `ecs-demo` task definition and it maintains 10 instantiations of that task.

$lb = @{
    LoadBalancerName = "EC2Contai-EcsElast-S06278JGSJCM"
    ContainerName = "simple-demo"
    ContainerPort = 80
}
New-ECSService -ServiceName ecs-simple-service -TaskDefinition ecs-demo -DesiredCount 10 -LoadBalancer $lb

• For API details, see [CreateService](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Remove-ECSCluster

The following code example shows how to use Remove-ECSCluster.

Tools for PowerShell

Example 1: This cmdlet deletes the specified ECS cluster. You must deregister all container instances from this cluster before you may delete it.

Remove-ECSCluster -Cluster "LAB-ECS"

Output:

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-ECSCluster (DeleteCluster)" on target "LAB-ECS".
For API details, see `DeleteCluster` in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-ECSService**

The following code example shows how to use `Remove-ECSService`.

**Tools for PowerShell**

**Example 1:** Deletes the service named 'my-http-service' in the default cluster. The service must have a desired count and running count of 0 before you can delete it. You are prompted for confirmation before the command proceeds. To bypass the confirmation prompt add the `-Force` switch.

```
Remove-ECSService -Service my-http-service
```

**Example 2:** Deletes the service named 'my-http-service' in the named cluster.

```
Remove-ECSService -Cluster myCluster -Service my-http-service
```

For API details, see `DeleteService` in *AWS Tools for PowerShell Cmdlet Reference*.

**Update-ECSClusterSetting**

The following code example shows how to use `Update-ECSClusterSetting`.

**Tools for PowerShell**

**Example 1:** This cmdlet modifies the settings to use for an ECS cluster.

```
Update-ECSClusterSetting -Cluster "LAB-ECS-CL" -Setting @{Name="containerInsights"; Value="disabled"}
```

**Output:**

```
ActiveServicesCount : 0
Attachments         : {}
AttachmentsStatus  :
CapacityProviders   : {}
```
ClusterName                       : LAB-ECS-CL
DefaultCapacityProviderStrategy   : {}
PendingTasksCount                 : 0
RegisteredContainerInstancesCount : 0
RunningTasksCount                 : 0
Settings                          : {containerInsights}
Statistics                        : {}
Status                            : ACTIVE
Tags                              : {}

For API details, see [UpdateClusterSettings](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Update-ECSService**

The following code example shows how to use `Update-ECSService`.

**Tools for PowerShell**

**Example 1:** This example command updates the `my-http-service` service to use the `amazon-ecs-sample` task definition.

```powershell
Update-ECSService -Service my-http-service -TaskDefinition amazon-ecs-sample
```

**Example 2:** This example command updates the desired count of the `my-http-service` service to 10.

```powershell
Update-ECSService -Service my-http-service -DesiredCount 10
```

For API details, see [UpdateService](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Amazon EFS examples using Tools for PowerShell**

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with Amazon EFS.

*Actions* are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.
Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

- Actions

Actions

Edit-EFSMountTargetSecurityGroup

The following code example shows how to use Edit-EFSMountTargetSecurityGroup.

Tools for PowerShell

Example 1: Updates the security groups in effect for the specified mount target. Up to 5 may be specified, in the format "sg-xxxxxxxx".

```
Edit-EFSMountTargetSecurityGroup -MountTargetId fsmt-1a2b3c4d -SecurityGroup sg-group1,sg-group3
```

- For API details, see ModifyMountTargetSecurityGroups in AWS Tools for PowerShell Cmdlet Reference.

Get-EFSFileSystem

The following code example shows how to use Get-EFSFileSystem.

Tools for PowerShell

Example 1: Returns the collection of all file systems owned by the caller's account in the region.

```
Get-EFSFileSystem
```

Output:

```
CreationTime : 5/26/2015 4:02:38 PM
```

Amazon EFS
Example 2: Returns the details of the specified file system.

Get-EFSFileSystem -FileSystemId fs-1a2b3c4d

Example 3: Returns the details of a file system using the idempotency creation token that was specified at the time the file system was created.

Get-EFSFileSystem -CreationToken 1a2bff54-85e0-4747-bd95-7bc172c4f555

• For API details, see DescribeFileSystems in AWS Tools for PowerShell Cmdlet Reference.

Get-EFSMountTarget

The following code example shows how to use Get-EFSMountTarget.

Tools for PowerShell

Example 1: Returns the collection of mount targets associated with the specified file system.

Get-EFSMountTarget -FileSystemId fs-1a2b3c4d

Output:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileSystemId</td>
<td>fs-1a2b3c4d</td>
</tr>
<tr>
<td>IpAddress</td>
<td>10.0.0.131</td>
</tr>
<tr>
<td>LifeCycleState</td>
<td>available</td>
</tr>
<tr>
<td>MountTargetId</td>
<td>fsmt-1a2b3c4d</td>
</tr>
<tr>
<td>NetworkInterfaceId</td>
<td>eni-1a2b3c4d</td>
</tr>
</tbody>
</table>

Amazon EFS
OwnerId            : 123456789012
SubnetId           : subnet-1a2b3c4d

- For API details, see DescribeMountTargets in AWS Tools for PowerShell Cmdlet Reference.

Get-EFSMountTargetSecurityGroup

The following code example shows how to use Get-EFSMountTargetSecurityGroup.

Tools for PowerShell

Example 1: Returns the ids of the security groups currently assigned to the network interface associated with the mount target.

Get-EFSMountTargetSecurityGroup -MountTargetId fsmt-1a2b3c4d

Output:

sg-1a2b3c4d

- For API details, see DescribeMountTargetSecurityGroups in AWS Tools for PowerShell Cmdlet Reference.

Get-EFSTag

The following code example shows how to use Get-EFSTag.

Tools for PowerShell

Example 1: Returns the collection of tags currently associated with the specified file system.

Get-EFSTag -FileSystemId fs-1a2b3c4d

Output:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>My File System</td>
</tr>
<tr>
<td>tagkey1</td>
<td>tagvalue1</td>
</tr>
<tr>
<td>tagkey2</td>
<td>tagvalue2</td>
</tr>
</tbody>
</table>
For API details, see DESCRIBETAGS in AWS Tools for PowerShell Cmdlet Reference.

New-EFSFileSystem

The following code example shows how to use New-EFSFileSystem.

Tools for PowerShell

Example 1: Creates a new, empty file system. The token used to ensure idempotent creation will be generated automatically and can be accessed from the CreationToken member of the returned object.

New-EFSFileSystem

Output:

<table>
<thead>
<tr>
<th>CreationTime</th>
<th>5/26/2015 4:02:38 PM</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreationToken</td>
<td>1a2bff54-85e0-4747-bd95-7bc172c4f555</td>
</tr>
<tr>
<td>FileSystemId</td>
<td>fs-1a2b3c4d</td>
</tr>
<tr>
<td>LifeCycleState</td>
<td>creating</td>
</tr>
<tr>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>NumberOfMountTargets</td>
<td>0</td>
</tr>
<tr>
<td>OwnerId</td>
<td>123456789012</td>
</tr>
<tr>
<td>SizeInBytes</td>
<td>Amazon.ElasticFileSystem.Model.FileSystemSize</td>
</tr>
</tbody>
</table>

Example 2: Creates a new, empty file system using a custom token to ensure idempotent creation.

New-EFSFileSystem -CreationToken "MyUniqueToken"

Tools for PowerShell

Example 1: Creates a new mount target for a file system. The specified subnet will be used to determine the Virtual Private Cloud (VPC) that the mount target will be created in and the
IP address that will be auto-assigned (from the address range of the subnet). The assigned IP address can be used to then mount this file system on an Amazon EC2 instance. As no security groups were specified the network interface created for the target is associated with the default security group for the subnet's VPC.

```
New-EFSMountTarget -FileSystemId fs-1a2b3c4d -SubnetId subnet-1a2b3c4d
```

**Output:**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileSystemId</td>
<td>fs-1a2b3c4d</td>
</tr>
<tr>
<td>IpAddress</td>
<td>10.0.0.131</td>
</tr>
<tr>
<td>LifeCycleState</td>
<td>creating</td>
</tr>
<tr>
<td>MountTargetId</td>
<td>fsmt-1a2b3c4d</td>
</tr>
<tr>
<td>NetworkInterfaceId</td>
<td>eni-1a2b3c4d</td>
</tr>
<tr>
<td>OwnerId</td>
<td>123456789012</td>
</tr>
<tr>
<td>SubnetId</td>
<td>subnet-1a2b3c4d</td>
</tr>
</tbody>
</table>

**Example 2:** Creates a new mount target for the specified file system with auto-assigned IP address. The network interface created for the mount target is associated with the specified security groups (up to 5, in the format "sg-xxxxxxxx", may be specified).

```
New-EFSMountTarget -FileSystemId fs-1a2b3c4d -SubnetId subnet-1a2b3c4d -SecurityGroup sg-group1,sg-group2,sg-group3
```

**Example 3:** Creates a new mount target for the specified file system with the specified IP address.

```
New-EFSMountTarget -FileSystemId fs-1a2b3c4d -SubnetId subnet-1a2b3c4d -IpAddress 10.0.0.131
```


**New-EFSTag**

The following code example shows how to use New-EFSTag.

**Tools for PowerShell**

**Example 1:** Applies the collection of tags to the specified file system. If a tag with key specified already exists on the file system the value of the tag is updated.
New-EFSTag -FileSystemId fs-1a2b3c4d -Tag @{Key="tagkey1";Value="tagvalue1"},@{Key="tagkey2";Value="tagvalue2"}

Example 2: Sets the name tag for the specified file system. This value is returned along with other file system details when the Get-EFSFileSystem cmdlet is used.

New-EFSTag -FileSystemId fs-1a2b3c4d -Tag @{Key="Name";Value="My File System"}

• For API details, see CreateTags in AWS Tools for PowerShell Cmdlet Reference.

Remove-EFSFileSystem

The following code example shows how to use Remove-EFSFileSystem.

Tools for PowerShell

Example 1: Deletes the specified file system that is no longer in use (if the file system has mount targets they must be removed first). You are prompted for confirmation before the cmdlet proceeds - to suppress confirmation, use the -Force switch.

Remove-EFSFileSystem -FileSystemId fs-1a2b3c4d

• For API details, see DeleteFileSystem in AWS Tools for PowerShell Cmdlet Reference.

Remove-EFSMountTarget

The following code example shows how to use Remove-EFSMountTarget.

Tools for PowerShell

Example 1: Deletes the specified mount target. You are prompted for confirmation before the operation proceeds. To suppress the prompt use the -Force switch. Note that this operation forcibly breaks any mounts of the file system via the target - you may want to consider unmounting the file system before running this command, if feasible.

Remove-EFSMountTarget -MountTargetId fsmt-1a2b3c4d

• For API details, see DeleteMountTarget in AWS Tools for PowerShell Cmdlet Reference.
Remove-EFSTag

The following code example shows how to use Remove-EFSTag.

Tools for PowerShell

Example 1: Deletes the collection of one or more tags from a file system. You are prompted for confirmation before the cmdlet proceeds - to suppress confirmation, use the -Force switch.

Remove-EFSTag -FileSystemId fs-1a2b3c4d -TagKey "tagkey1","tagkey2"

• For API details, see DeleteTags in AWS Tools for PowerShell Cmdlet Reference.

Amazon EKS examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with Amazon EKS.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

• Actions

Actions

Add-EKSRessourceTag

The following code example shows how to use Add-EKSRessourceTag.
**Tools for PowerShell**

**Example 1:** This cmdlet associates the specified tags to a resource with the specified resourceArn.

```
```

- For API details, see [TagResource](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-EKSCluster**

The following code example shows how to use `Get-EKSCluster`.

**Tools for PowerShell**

**Example 1:** This cmdlet returns descriptive information about an Amazon EKS cluster.

```
Get-EKSCluster -Name "PROD"
```

**Output:**

```
CertificateAuthority : Amazon.EKS.Model.Certificate
ClientRequestToken   :
CreatedAt            : 12/25/2019 6:46:17 AM
Endpoint             : https://669608765450FBBE54D1D78A3D71B72C.gr8.us-west-2.eks.amazonaws.com
Identity             : Amazon.EKS.Model.Identity
Logging              : Amazon.EKS.Model.Logging
Name                 : PROD
PlatformVersion      : eks.7
ResourcesVpcConfig   : Amazon.EKS.Model.VpcConfigResponse
RoleArn              : arn:aws:iam::012345678912:role/eks-iam-role
Status               : ACTIVE
Tags                 : {}
Version              : 1.14
```

- For API details, see [DescribeCluster](#) in *AWS Tools for PowerShell Cmdlet Reference*.
Get-EKSClusterList

The following code example shows how to use Get-EKSClusterList.

Tools for PowerShell

Example 1: This cmdlet lists the Amazon EKS clusters in your AWS account in the specified Region.

Get-EKSClusterList

Output:

PROD

- For API details, see ListClusters in AWS Tools for PowerShell Cmdlet Reference.

Get-EKSFargateProfile

The following code example shows how to use Get-EKSFargateProfile.

Tools for PowerShell

Example 1: This cmdlet returns descriptive information about an AWS Fargate profile.

Get-EKSFargateProfile -FargateProfileName "EKSFargate" -ClusterName "TEST"

Output:

<table>
<thead>
<tr>
<th>ClusterName</th>
<th>TEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreatedAt</td>
<td>12/26/2019 12:34:47 PM</td>
</tr>
<tr>
<td>FargateProfileArn</td>
<td>arn:aws:eks:us-east-2:012345678912:fargateprofile/TEST/EKSFargate/42b7a119-e16b-a279-ce97-bdf303adec92</td>
</tr>
<tr>
<td>FargateProfileName</td>
<td>EKSFargate</td>
</tr>
<tr>
<td>PodExecutionRoleArn</td>
<td>arn:aws:iam::012345678912:role/AmazonEKSFargatePodExecutionRole</td>
</tr>
<tr>
<td>Selectors</td>
<td>{Amazon.EKS.Model.FargateProfileSelector}</td>
</tr>
<tr>
<td>Status</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>Subnets</td>
<td>{subnet-0cd976f08d5fbfaae, subnet-02f6ff500ff2067a0}</td>
</tr>
<tr>
<td>Tags</td>
<td>{}</td>
</tr>
</tbody>
</table>
• For API details, see DescribeFargateProfile in AWS Tools for PowerShell Cmdlet Reference.

Get-EKSFargateProfileList

The following code example shows how to use Get-EKSFargateProfileList.

Tools for PowerShell

Example 1: This cmdlet lists the AWS Fargate profiles associated with the specified cluster in your AWS account in the specified Region.

Get-EKSFargateProfileList -ClusterName "TEST"

Output:

EKSFargate
EKSFargateProfile

• For API details, see ListFargateProfiles in AWS Tools for PowerShell Cmdlet Reference.

Get-EKSNodegroup

The following code example shows how to use Get-EKSNodegroup.

Tools for PowerShell

Example 1: This cmdlet returns descriptive information about an Amazon EKS node group.

Get-EKSNodegroup -NodegroupName "ProdEKSNodeGroup" -ClusterName "PROD"

Output:

amiType : AL2_x86_64
clusterName : PROD
createdAt : 12/25/2019 10:16:45 AM
diskSize : 40
health : Amazon.EKS.Model.NodegroupHealth
instanceTypes : {t3.large}
labels : {}
For API details, see [DescribeNodegroup](in AWS Tools for PowerShell Cmdlet Reference).

**Get-EKSNodegroupList**

The following code example shows how to use Get-EKSNodegroupList.

**Tools for PowerShell**

**Example 1:** This cmdlet lists the Amazon EKS node groups associated with the specified cluster in your AWS account in the specified Region.

```
Get-EKSNodegroupList -ClusterName PROD
```

**Output:**

```
ProdEKSNodeGroup
```

For API details, see [ListNodegroups](in AWS Tools for PowerShell Cmdlet Reference).

**Get-EKSResourceTag**

The following code example shows how to use Get-EKSResourceTag.

**Tools for PowerShell**

**Example 1:** This cmdlet list the tags for an Amazon EKS resource.

Output:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>EKSPRODCLUSTER</td>
</tr>
</tbody>
</table>

For API details, see `ListTagsForResource` in *AWS Tools for PowerShell Cmdlet Reference*.

Get-EKSUpdate

The following code example shows how to use Get-EKSUpdate.

Tools for PowerShell

**Example 1:** This cmdlet returns descriptive information about an update against your Amazon EKS cluster or associated managed node group.

```powershell
Get-EKSUpdate -Name "PROD" -UpdateId "ee708232-7d2e-4ed7-9270-d0b5176f0726"
```

Output:

<table>
<thead>
<tr>
<th>CreatedAt</th>
<th>12/25/2019 5:03:07 PM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Errors</td>
<td>{}</td>
</tr>
<tr>
<td>Id</td>
<td>ee708232-7d2e-4ed7-9270-d0b5176f0726</td>
</tr>
<tr>
<td>Params</td>
<td>[Amazon.EKS.Model.UpdateParam]</td>
</tr>
<tr>
<td>Status</td>
<td>Successful</td>
</tr>
<tr>
<td>Type</td>
<td>LoggingUpdate</td>
</tr>
</tbody>
</table>

For API details, see `DescribeUpdate` in *AWS Tools for PowerShell Cmdlet Reference*.

Get-EKSUpdateList

The following code example shows how to use Get-EKSUpdateList.

Tools for PowerShell

**Example 1:** This cmdlet lists the updates associated with an Amazon EKS cluster or managed node group in your AWS account, in the specified Region.
Get-EKSUpdateList -Name "PROD"

Output:

ee708232-7d2e-4ed7-9270-d0b5176f0726

- For API details, see ListUpdates in AWS Tools for PowerShell Cmdlet Reference.

New-EKSCluster

The following code example shows how to use New-EKSCluster.

Tools for PowerShell

Example 1: This example creates a new cluster called 'prod'.

```
New-EKSCluster -Name prod -ResourcesVpcConfig @{SubnetIds=@("subnet-0a1b2c3d","subnet-3a2b1c0d");SecurityGroupIds="sg-6979fe18"} -RoleArn "arn:aws:iam::012345678901:role/eks-service-role"
```

Output:

```
CertificateAuthority : Amazon.EKS.Model.Certificate
ClientRequestToken   :
CreatedAt            : 12/10/2018 9:25:31 PM
Endpoint             :
Name                 : prod
PlatformVersion      : eks.3
ResourcesVpcConfig   : Amazon.EKS.Model.VpcConfigResponse
RoleArn              : arn:aws:iam::012345678901:role/eks-service-role
Status               : CREATING
Version              : 1.10
```

- For API details, see CreateCluster in AWS Tools for PowerShell Cmdlet Reference.

New-EKSFargateProfile

The following code example shows how to use New-EKSFargateProfile.
Example 1: This cmdlet creates an AWS Fargate profile for your Amazon EKS cluster. You must have at least one Fargate profile in a cluster to be able to schedule pods on Fargate infrastructure.

```powershell
New-EKSFargateProfile -FargateProfileName EKSFargateProfile -ClusterName TEST -Subnet "subnet-02f6ff500ff2067a0", "subnet-0cd976f08d5fbfaae" -PodExecutionRoleArn arn:aws:iam::012345678912:role/AmazonEKSFargatePodExecutionRole -Selector @{Namespace="default"}
```

Output:

```
ClusterName : TEST
CreatedAt   : 12/26/2019 12:38:21 PM
FargateProfileArn   : arn:aws:eks:us-east-2:012345678912:fargateprofile/TEST/EKSFargateProfile/20b7a11b-8292-41c1-bc56-ffa5e60f6224
FargateProfileName  : EKSFargateProfile
PodExecutionRoleArn : arn:aws:iam::012345678912:role/AmazonEKSFargatePodExecutionRole
Selectors          : {Amazon.EKS.Model.FargateProfileSelector}
Status             : CREATING
Subnets            : {subnet-0cd976f08d5fbfaae, subnet-02f6ff500ff2067a0}
Tags               : {}
```

- For API details, see CreateFargateProfile in AWS Tools for PowerShell Cmdlet Reference.

**New-EKSNodeGroup**

The following code example shows how to use New-EKSNodeGroup.

Tools for PowerShell

Example 1: This cmdlet creates a managed worker node group for an Amazon EKS cluster. You can only create a node group for your cluster that is equal to the current Kubernetes version for the cluster. All node groups are created with the latest AMI release version for the respective minor Kubernetes version of the cluster.

```powershell
New-EKSNodeGroup -NodeGroupName "ProdEKSNodeGroup" -AmiType "AL2_x86_64" -DiskSize 40 -ClusterName "PROD" -ScalingConfig_DesiredSize 2 -ScalingConfig_MinSize 2 -ScalingConfig_MaxSize 5 -InstanceType t3.large
```
-NodeRole "arn:aws:iam::012345678912:role/NodeInstanceRole" -Subnet "subnet-0d1a9fff35efa7691","subnet-0a3f4928edbc224d4"

Output:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AmiType</td>
<td>AL2_x86_64</td>
</tr>
<tr>
<td>ClusterName</td>
<td>PROD</td>
</tr>
<tr>
<td>CreatedAt</td>
<td>12/25/2019 10:16:45 AM</td>
</tr>
<tr>
<td>DiskSize</td>
<td>40</td>
</tr>
<tr>
<td>Health</td>
<td>Amazon.EKS.Model.NodegroupHealth</td>
</tr>
<tr>
<td>InstanceTypes</td>
<td>{t3.large}</td>
</tr>
<tr>
<td>Labels</td>
<td>{}</td>
</tr>
<tr>
<td>ModifiedAt</td>
<td>12/25/2019 10:16:45 AM</td>
</tr>
<tr>
<td>NodegroupArn</td>
<td>arn:aws:eks:us-west-2:012345678912:nodegroup/PROD/ProdEKSNodeGroup/7eb79e47-82b6-04d9-e984-95110db6fa85</td>
</tr>
<tr>
<td>NodegroupName</td>
<td>ProdEKSNodeGroup</td>
</tr>
<tr>
<td>NodeRole</td>
<td>arn:aws:iam::012345678912:role/NodeInstanceRole</td>
</tr>
<tr>
<td>ReleaseVersion</td>
<td>1.14.7-20190927</td>
</tr>
<tr>
<td>RemoteAccess</td>
<td></td>
</tr>
<tr>
<td>Resources</td>
<td></td>
</tr>
<tr>
<td>ScalingConfig</td>
<td>Amazon.EKS.Model.NodegroupScalingConfig</td>
</tr>
<tr>
<td>Status</td>
<td>CREATING</td>
</tr>
<tr>
<td>Subnets</td>
<td>{subnet-0d1a9fff35efa7691, subnet-0a3f4928edbc224d4}</td>
</tr>
<tr>
<td>Tags</td>
<td>{}</td>
</tr>
<tr>
<td>Version</td>
<td>1.14</td>
</tr>
</tbody>
</table>

- For API details, see [CreateNodegroup](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-EKSCluster**

The following code example shows how to use Remove-EKSCluster.

**Tools for PowerShell**

**Example 1: This cmdlet deletes the Amazon EKS cluster control plane.**

```
Remove-EKSCluster -Name "DEV-KUBE-CL"
```

**Output:**

```
Confirm
Are you sure you want to perform this action?
```
Performing the operation "Remove-EKSCluster (DeleteCluster)" on target "DEV-KUBE-CL".

[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): Y

CertificateAuthority : Amazon.EKS.Model.Certificate
ClientRequestToken   :
CreatedAt            : 12/25/2019 9:33:25 AM
Endpoint             : https://02E6D31E3E4F8C15D7BE7F58D527776A.yl4.us-west-2.eks.amazonaws.com
Identity             : Amazon.EKS.Model.Identity
Logging              : Amazon.EKS.Model.Logging
Name                 : DEV-KUBE-CL
PlatformVersion      : eks.7
ResourcesVpcConfig   : Amazon.EKS.Model.VpcConfigResponse
RoleArn              : arn:aws:iam::012345678912:role/eks-iam-role
Status               : DELETING
Tags                 : {}
Version              : 1.14

For API details, see DeleteCluster in AWS Tools for PowerShell Cmdlet Reference.

Remove-EKSFargateProfile

The following code example shows how to use Remove-EKSFargateProfile.

Tools for PowerShell

Example 1: This cmdlet deletes an AWS Fargate profile. When you delete a Fargate profile, any pods running on Fargate that were created with the profile are deleted.

Remove-EKSFargateProfile -FargateProfileName "EKSFargate" -ClusterName "TEST"

Output:

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-EKSFargateProfile (DeleteFargateProfile)" on target "EKSFargate".

[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): Y
ClusterName         : TEST
CreatedAt           : 12/26/2019 12:34:47 PM
FargateProfileArn   : arn:aws:eks:us-east-2:012345678912:fargateprofile/TEST/EKSFargate/42b7a119-e16b-a279-ce97-bdf303adec92
FargateProfileName  : EKSFargate
PodExecutionRoleArn : arn:aws:iam::012345678912:role/AmazonEKSFargatePodExecutionRole
Selectors           : {Amazon.EKS.Model.FargateProfileSelector}
Status              : DELETING
Subnets             : {subnet-0cd976f08d5fbfaae, subnet-02f6ff500ff2067a0}
Tags                : {}

• For API details, see DeleteFargateProfile in AWS Tools for PowerShell Cmdlet Reference.

Remove-EKSNodegroup

The following code example shows how to use Remove-EKSNodegroup.

Tools for PowerShell

Example 1: This cmdlet deletes an Amazon EKS node group for a cluster.

Remove-EKSNodegroup -NodegroupName "ProdEKSNodeGroup" -ClusterName "PROD"

Output:

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-EKSNodegroup (DeleteNodegroup)" on target "ProdEKSNodeGroup".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): Y

    AmiType        : AL2_x86_64
    ClusterName    : PROD
    CreatedAt      : 12/25/2019 10:16:45 AM
    DiskSize       : 40
    Health         : Amazon.EKS.Model.NodegroupHealth
    InstanceTypes  : {t3.large}
    Labels         : {}
    ModifiedAt     : 12/25/2019 11:01:16 AM
NodegroupArn   : arn:aws:eks:us-west-2:012345678912:nodegroup/PROD/ProdEKSNodeGroup/7eb79e47-82b6-04d9-e984-95110db6fa85
NodegroupName  : ProdEKSNodeGroup
NodeRole       : arn:aws:iam::012345678912:role/NodeInstanceRole
ReleaseVersion : 1.14.7-20190927
RemoteAccess   :
Resources      : Amazon.EKS.Model.NodegroupResources
ScalingConfig  : Amazon.EKS.Model.NodegroupScalingConfig
Status         : DELETING
Subnets        : {subnet-0d1a9fff35efa7691, subnet-0a3f4928edbc224d4}
Tags           : {}
Version        : 1.14

- For API details, see [DeleteNodegroup](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Remove-EKSRessourceTag

The following code example shows how to use Remove-EKSRessourceTag.

**Tools for PowerShell**

**Example 1: This cmdlet deletes specified tags from an EKS resource.**

```powershell
Remove-EKSRessourceTag -ResourceArn "arn:aws:eks:us-west-2:012345678912:cluster/PROD" -TagKey "Name"
```

**Output:**

```
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-EKSRessourceTag (UntagResource)" on target "arn:aws:eks:us-west-2:012345678912:cluster/PROD".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): Y
```

- For API details, see [UntagResource](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Update-EKSClusterConfig

The following code example shows how to use Update-EKSClusterConfig.

---

Amazon EKS

489
Tools for PowerShell

Example 1: Updates an Amazon EKS cluster configuration. Your cluster continues to function during the update.

```powershell
Update-EKSClusterConfig -Name "PROD" -Logging_ClusterLogging @{Types="api","audit","authenticator","controllerManager","scheduler",Enabled="True"}
```

Output:

```powershell
CreatedAt : 12/25/2019 5:03:07 PM
Errors    : {}
Id        : ee708232-7d2e-4ed7-9270-d0b5176f0726
Params    : {Amazon.EKS.Model.UpdateParam}
Status    : InProgress
Type      : LoggingUpdate
```

• For API details, see [UpdateClusterConfig](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

Update-EKSClusterVersion

The following code example shows how to use Update-EKSClusterVersion.

Tools for PowerShell

Example 1: This cmdlet updates an Amazon EKS cluster to the specified Kubernetes version. Your cluster continues to function during the update.

```powershell
Update-EKSClusterVersion -Name "PROD-KUBE-CL" -Version 1.14
```

Output:

```powershell
CreatedAt : 12/26/2019 9:50:37 AM
Errors    : {}
Id        : ef186eff-3b3a-4c25-bcfc-3dcdf9e898a8
Params    : {Amazon.EKS.Model.UpdateParam, Amazon.EKS.Model.UpdateParam}
Status    : InProgress
Type      : VersionUpdate
```

• For API details, see [UpdateClusterVersion](#) in [AWS Tools for PowerShell Cmdlet Reference](#).
Elastic Load Balancing - Version 1 examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with Elastic Load Balancing - Version 1.

*Actions* are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

*Scenarios* are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

**Topics**
- **Actions**

**Actions**

Add-ELBLoadBalancerToSubnet

The following code example shows how to use Add-ELBLoadBalancerToSubnet.

**Tools for PowerShell**

*Example 1:* This example adds the specified subnet to the set of subnets configured for the specified load balancer. The output includes the complete list of subnets.

```powershell
Add-ELBLoadBalancerToSubnet -LoadBalancerName my-load-balancer -Subnet subnet-12345678
```

**Output:**

```
subnet-12345678
subnet-87654321
```

- For API details, see [AttachLoadBalancerToSubnets](#) in AWS Tools for PowerShell Cmdlet Reference.
Add-ELBResourceTag

The following code example shows how to use Add-ELBResourceTag.

Tools for PowerShell

Example 1: This example adds the specified tags to the specified load balancer. The syntax used by this example requires PowerShell version 3 or later.

```
Add-ELBResourceTag -LoadBalancerName my-load-balancer -Tag
@{ Key="project";Value="lima" },@{ Key="department";Value="digital-media" }
```

Example 2: With PowerShell version 2, you must use New-Object to create a tag for the Tag parameter.

```
$tag = New-Object Amazon.ElasticLoadBalancing.Model.Tag
$tag.Key = "project"
$tag.Value = "lima"
Add-ELBResourceTag -LoadBalancerName my-load-balancer -Tag $tag
```

- For API details, see AddTags in AWS Tools for PowerShell Cmdlet Reference.

Disable-ELBAvailabilityZoneForLoadBalancer

The following code example shows how to use Disable-ELBAvailabilityZoneForLoadBalancer.

Tools for PowerShell

Example 1: This example removes the specified Availability Zone from the specified load balancer. The output includes the remaining Availability Zones.

```
Disable-ELBAvailabilityZoneForLoadBalancer -LoadBalancerName my-load-balancer -AvailabilityZone us-west-2a
```

Output:

```
us-west-2b
```

- For API details, see DisableAvailabilityZonesForLoadBalancer in AWS Tools for PowerShell Cmdlet Reference.
**Dismount-ELBLoadBalancerFromSubnet**

The following code example shows how to use Dismount-ELBLoadBalancerFromSubnet.

**Tools for PowerShell**

**Example 1:** This example removes the specified subnet from the set of subnets configured for the specified load balancer. The output includes the remaining subnets.

```
Dismount-ELBLoadBalancerFromSubnet -LoadBalancerName my-load-balancer -Subnet subnet-12345678
```

**Output:**

```
subnet-87654321
```

- For API details, see [DetachLoadBalancerFromSubnets](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Edit-ELBLoadBalancerAttribute**

The following code example shows how to use Edit-ELBLoadBalancerAttribute.

**Tools for PowerShell**

**Example 1:** This example enables cross-zone load balancing for the specified load balancer.

```
Edit-ELBLoadBalancerAttribute -LoadBalancerName my-load-balancer -CrossZoneLoadBalancing_Enabled $true
```

**Example 2:** This example disables connection draining for the specified load balancer.

```
Edit-ELBLoadBalancerAttribute -LoadBalancerName my-load-balancer -ConnectionDraining_Enabled $false
```

**Example 3:** This example enables access logging for the specified load balancer.

```
Edit-ELBLoadBalancerAttribute -LoadBalancerName my-load-balancer `>> -AccessLog_Enabled $true`
```
For API details, see ModifyLoadBalancerAttributes in AWS Tools for PowerShell Cmdlet Reference.

Enable-ELBAvailabilityZoneForLoadBalancer

The following code example shows how to use Enable-ELBAvailabilityZoneForLoadBalancer.

Tools for PowerShell

Example 1: This example adds the specified Availability Zone to the specified load balancer. The output includes the complete list of Availability Zones.

```
Enable-ELBAvailabilityZoneForLoadBalancer -LoadBalancerName my-load-balancer -AvailabilityZone us-west-2a
```

Output:

```
us-west-2a
us-west-2b
```

For API details, see EnableAvailabilityZonesForLoadBalancer in AWS Tools for PowerShell Cmdlet Reference.

Get-ELBInstanceHealth

The following code example shows how to use Get-ELBInstanceHealth.

Tools for PowerShell

Example 1: This example describes the state of the instances registered with the specified load balancer.

```
Get-ELBInstanceHealth -LoadBalancerName my-load-balancer
```

Elastic Load Balancing - Version 1
Output:

<table>
<thead>
<tr>
<th>Description</th>
<th>InstanceId</th>
<th>ReasonCode</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>i-87654321</td>
<td>N/A</td>
</tr>
<tr>
<td>InService</td>
<td>i-12345678</td>
<td></td>
</tr>
<tr>
<td>OutOfService</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example 2: This example describes the state of the specified instance registered with the specified load balancer.

```
Get-ELBInstanceHealth -LoadBalancerName my-load-balancer -Instance i-12345678
```

Example 3: This example displays the complete description of the state of the specified instance.

```
(Get-ELBInstanceHealth -LoadBalancerName my-load-balancer -Instance i-12345678).Description
```

Output:

Instance has failed at least the UnhealthyThreshold number of health checks consecutively.

- For API details, see [DescribeInstanceHealth](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-ELBLoadBalancer

The following code example shows how to use Get-ELBLoadBalancer.

Tools for PowerShell

Example 1: This example lists the names of your load balancers.

```
Get-ELBLoadBalancer | format-table -property LoadBalancerName
```
Output:

LoadBalancerName
----------------
my-load-balancer
my-other-load-balancer
my-internal-load-balancer

Example 2: This example describes the specified load balancer.

Get-ELBLoadBalancer -LoadBalancerName my-load-balancer

Output:

AvailabilityZones : {us-west-2a, us-west-2b}
BackendServerDescriptions :
    {Amazon.ElasticLoadBalancing.Model.BackendServerDescription}
CanonicalHostedZoneName : my-load-balancer-1234567890.us-west-2.elb.amazonaws.com
CanonicalHostedZoneNameID : Z3DZXEOEXAMPLE
CreatedTime : 4/11/2015 12:12:45 PM
DNSName : my-load-balancer-1234567890.us-west-2.elb.amazonaws.com
Instances : {i-207d9717, i-afefb49b}
ListenerDescriptions : {Amazon.ElasticLoadBalancing.Model.ListenerDescription}
LoadBalancerName : my-load-balancer
Scheme : internet-facing
SecurityGroups : {sg-a61988c3}
Subnets : {subnet-15aaab61}
VPCId : vpc-a01106c2

Example 3: This example describes all your load balancers in the current AWS region.

Get-ELBLoadBalancer

Example 4: This example describes all your load balancers across all available AWS Regions.

Get-AWSRegion | % { Get-ELBLoadBalancer -Region $._ }

• For API details, see DescribeLoadBalancers in AWS Tools for PowerShell Cmdlet Reference.
Get-ELBLoadBalancerAttribute

The following code example shows how to use Get-ELBLoadBalancerAttribute.

Tools for PowerShell

Example 1: This example describes the attributes for the specified load balancer.

Get-ELBLoadBalancerAttribute -LoadBalancerName my-load-balancer

Output:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdditionalAttributes</td>
<td>{}</td>
</tr>
<tr>
<td>ConnectionDraining</td>
<td>Amazon.ElasticLoadBalancing.Model.ConnectionDraining</td>
</tr>
<tr>
<td>ConnectionSettings</td>
<td>Amazon.ElasticLoadBalancing.Model.ConnectionSettings</td>
</tr>
<tr>
<td>CrossZoneLoadBalancing</td>
<td>Amazon.ElasticLoadBalancing.Model.CrossZoneLoadBalancing</td>
</tr>
</tbody>
</table>

- For API details, see DescribeLoadBalancerAttributes in AWS Tools for PowerShell Cmdlet Reference.

Get-ELBLoadBalancerPolicy

The following code example shows how to use Get-ELBLoadBalancerPolicy.

Tools for PowerShell

Example 1: This example describes the policies associated with the specified load balancer.

Get-ELBLoadBalancerPolicy -LoadBalancerName my-load-balancer

Output:

<table>
<thead>
<tr>
<th>PolicyAttributeDescriptions</th>
<th>PolicyName</th>
</tr>
</thead>
<tbody>
<tr>
<td>PolicyTypeName</td>
<td></td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>{ProxyProtocol}</td>
<td>my-ProxyProtocol-policy</td>
</tr>
<tr>
<td>ProxyProtocolPolicyType</td>
<td></td>
</tr>
<tr>
<td>{CookieName}</td>
<td>my-app-cookie-policy</td>
</tr>
<tr>
<td>AppCookieStickinessPolicyType</td>
<td></td>
</tr>
</tbody>
</table>
Example 2: This example describes the attributes of the specified policy.

```powershell
(Get-ELBLoadBalancerPolicy -LoadBalancerName my-load-balancer -PolicyName my-ProxyProtocol-policy).PolicyAttributeDescriptions
```

**Output:**

<table>
<thead>
<tr>
<th>AttributeName</th>
<th>AttributeValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProxyProtocol</td>
<td>true</td>
</tr>
</tbody>
</table>

Example 3: This example describes the predefined policies, including the sample policies. The names of the sample policies have the ELBSample-prefix.

```powershell
Get-ELBLoadBalancerPolicy
```

**Output:**

<table>
<thead>
<tr>
<th>PolicyAttributeDescriptions</th>
<th>PolicyName</th>
</tr>
</thead>
<tbody>
<tr>
<td>PolicyTypeName</td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>{Protocol-SSLv2, Protocol-TLSv1, Protocol-TLSv1}</td>
<td>ELBSecurityPolicy-2011-08</td>
</tr>
<tr>
<td>{Protocol-SSLv2, Protocol-TLSv1, Protocol-TLSv1}</td>
<td>ELBSample-ELBDefaultCipherPolicy</td>
</tr>
<tr>
<td>{Protocol-SSLv2, Protocol-TLSv1, Protocol-TLSv1}</td>
<td>ELBSample-OpenSSLDefaultCipherPolicy</td>
</tr>
</tbody>
</table>

- For API details, see [DescribeLoadBalancerPolicies](#) in *AWS Tools for PowerShell Cmdlet Reference*.
Get-ELBLoadBalancerPolicyType

The following code example shows how to use Get-ELBLoadBalancerPolicyType.

**Tools for PowerShell**

**Example 1: This example gets the policy types supported by Elastic Load Balancing.**

```powershell
Get-ELBLoadBalancerPolicyType
```

**Output:**

<table>
<thead>
<tr>
<th>Description</th>
<th>PolicyAttributeTypeDescriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>PolicyTypeName</td>
<td></td>
</tr>
<tr>
<td>Stickiness policy with session lifet...</td>
<td>{CookieExpirationPeriod}</td>
</tr>
<tr>
<td>LBCookieStickinessPolicyType</td>
<td></td>
</tr>
<tr>
<td>Policy that controls authentication ...</td>
<td>{PublicKeyPolicyName}</td>
</tr>
<tr>
<td>BackendServerAuthenticationPolicyType</td>
<td></td>
</tr>
<tr>
<td>Listener policy that defines the cip...</td>
<td>{Protocol-SSLv2, Protocol-TLSv1, Protocol-HTTPv1, Protocol-HTTPv2}</td>
</tr>
<tr>
<td>SSLNegotiationPolicyType</td>
<td></td>
</tr>
<tr>
<td>Policy containing a list of public k...</td>
<td>{PublicKey}</td>
</tr>
<tr>
<td>PublicKeyPolicyType</td>
<td></td>
</tr>
<tr>
<td>Stickiness policy with session lifet...</td>
<td>{CookieName}</td>
</tr>
<tr>
<td>AppCookieStickinessPolicyType</td>
<td></td>
</tr>
<tr>
<td>Policy that controls whether to incl...</td>
<td>{ProxyProtocol}</td>
</tr>
<tr>
<td>ProxyProtocolPolicyType</td>
<td></td>
</tr>
</tbody>
</table>

**Example 2: This example describes the specified policy type.**

```powershell
Get-ELBLoadBalancerPolicyType -PolicyTypeName ProxyProtocolPolicyType
```

**Output:**

<table>
<thead>
<tr>
<th>Description</th>
<th>PolicyAttributeTypeDescriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>PolicyTypeName</td>
<td></td>
</tr>
<tr>
<td>Stickiness policy with session lifet...</td>
<td>{CookieName}</td>
</tr>
<tr>
<td>LBCookieStickinessPolicyType</td>
<td></td>
</tr>
<tr>
<td>Policy that controls whether to incl...</td>
<td>{ProxyProtocol}</td>
</tr>
<tr>
<td>ProxyProtocolPolicyType</td>
<td></td>
</tr>
</tbody>
</table>
Example 3: This example displays the complete description of the specified policy type.

```
(Get-ELBLoadBalancerPolicyType -PolicyTypeName).Description
```

Output:

```
Policy that controls whether to include the IP address and port of the originating request for TCP messages. This policy operates on TCP/SSL listeners only.
```

- For API details, see [DescribeLoadBalancerPolicyTypes](#) in AWS Tools for PowerShell Cmdlet Reference.

Get-ELBResourceTag

The following code example shows how to use Get-ELBResourceTag.

Tools for PowerShell

Example 1: This example lists the tags for the specified load balancers.

```
Get-ELBResourceTag -LoadBalancerName @("my-load-balancer","my-internal-load-balancer")
```

Output:

```
LoadBalancerName          Tags
------------------------  ----
my-load-balancer          {project, department}
my-internal-load-balancer {project, department}
```

Example 2: This example describes the tags for the specified load balancer.

```
(Get-ELBResourceTag -LoadBalancerName my-load-balancer).Tags
```

Output:

```
Key   Value
---   ----
```
Join-ELBSecurityGroupToLoadBalancer

The following code example shows how to use `Join-ELBSecurityGroupToLoadBalancer`.

**Tools for PowerShell**

**Example 1:** This example replaces the current security group for the specified load balancer with the specified security group.

```
Join-ELBSecurityGroupToLoadBalancer -LoadBalancerName my-load-balancer -SecurityGroup sg-87654321
```

**Output:**

```
sg-87654321
```

**Example 2:** To keep the current security group and specify an additional security group, specify both the existing and new security groups.

```
Join-ELBSecurityGroupToLoadBalancer -LoadBalancerName my-load-balancer -SecurityGroup @("sg-12345678", "sg-87654321")
```

**Output:**

```
sg-12345678
sg-87654321
```

For API details, see [ApplySecurityGroupsToLoadBalancer](#) in *AWS Tools for PowerShell Cmdlet Reference*.

New-ELBAppCookieStickinessPolicy

The following code example shows how to use `New-ELBAppCookieStickinessPolicy`.

For API details, see [DescribeTags](#) in *AWS Tools for PowerShell Cmdlet Reference*.
Tools for PowerShell

Example 1: This example creates a stickiness policy that follows the sticky session lifetimes of the specified application-generated cookie.

```
New-ELBAppCookieStickinessPolicy -LoadBalancerName my-load-balancer -PolicyName my-app-cookie-policy -CookieName my-app-cookie
```

- For API details, see [CreateAppCookieStickinessPolicy](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

**New-ELBLBCookieStickinessPolicy**

The following code example shows how to use `New-ELBLBCookieStickinessPolicy`.

Tools for PowerShell

Example 1: This example creates a stickiness policy with sticky session lifetimes controlled by the specified expiration period (in seconds).

```
New-ELBLBCookieStickinessPolicy -LoadBalancerName my-load-balancer -PolicyName my-duration-cookie-policy -CookieExpirationPeriod 60
```

Example 2: This example creates a stickiness policy with sticky session lifetimes controlled by the lifetime of the browser (user-agent).

```
New-ELBLBCookieStickinessPolicy -LoadBalancerName my-load-balancer -PolicyName my-duration-cookie-policy
```

- For API details, see [CreateLbCookieStickinessPolicy](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

**New-ELBLoadBalancer**

The following code example shows how to use `New-ELBLoadBalancer`.

Tools for PowerShell

Example 1: This example creates a load balancer with an HTTP listener in a VPC.

```
$httpListener = New-Object Amazon.ElasticLoadBalancing.Model.Listener
```
$httpListener.Protocol = "http"
$httpListener.LoadBalancerPort = 80
$httpListener.InstanceProtocol = "http"
$httpListener.InstancePort = 80

my-vpc-load-balancer-1234567890.us-west-2.elb.amazonaws.com

Example 2: This example creates a load balancer with an HTTP listener in EC2-Classic.

New-ELBLoadBalancer -LoadBalancerName my-classic-load-balancer -AvailabilityZone us-west-2a -Listener $httpListener

Output:

my-classic-load-balancer-123456789.us-west-2.elb.amazonaws.com

Example 3: This example creates a load balancer with an HTTPS listener.

$httpsListener = New-Object Amazon.ElasticLoadBalancing.Model.Listener
$httpsListener.Protocol = "https"
$httpsListener.LoadBalancerPort = 443
$httpsListener.InstanceProtocol = "http"
$httpsListener.InstancePort = 80
$httpsListener.SSLCertificateId="arn:aws:iamp::123456789012:server-certificate/my-server-cert"
New-ELBLoadBalancer -LoadBalancerName my-load-balancer -AvailabilityZone us-west-2a
                             -Listener $httpsListener

my-load-balancer-123456789.us-west-2.elb.amazonaws.com

- For API details, see [CreateLoadBalancer](#) in AWS Tools for PowerShell Cmdlet Reference.

**New-ELBLoadBalancerListener**

The following code example shows how to use New-ELBLoadBalancerListener.

**Tools for PowerShell**

**Example 1:** This example adds an HTTPS listener to the specified load balancer.
$httpsListener = New-Object Amazon.ElasticLoadBalancing.Model.Listener
$httpsListener.Protocol = "https"
$httpsListener.LoadBalancerPort = 443
$httpsListener.InstanceProtocol = "https"
$httpsListener.InstancePort = 443
$httpsListener.SSLCertificateId="arn:aws:iam::123456789012:server-certificate/my-server-cert"
New-ELBLoadBalancerListener -LoadBalancerName my-load-balancer -Listener $httpsListener

- For API details, see CreateLoadBalancerListeners in AWS Tools for PowerShell Cmdlet Reference.

New-ELBLoadBalancerPolicy

The following code example shows how to use New-ELBLoadBalancerPolicy.

Tools for PowerShell

Example 1: This example creates a new proxy protocol policy for a specified load balancer.

    AttributeName="ProxyProtocol"
    AttributeValue="True"
}
New-ELBLoadBalancerPolicy -LoadBalancerName my-load-balancer -PolicyName my-ProxyProtocol-policy -PolicyTypeName ProxyProtocolPolicyType -PolicyAttribute $attribute

- For API details, see CreateLoadBalancerPolicy in AWS Tools for PowerShell Cmdlet Reference.

Register-ELBInstanceWithLoadBalancer

The following code example shows how to use Register-ELBInstanceWithLoadBalancer.

Tools for PowerShell

Example 1: This example registers the specified EC2 instance with the specified load balancer.
Register-ELBInstanceWithLoadBalancer -LoadBalancerName my-load-balancer -Instance i-12345678

Output:

InstanceId
----------
i-12345678
i-87654321

For API details, see RegisterInstancesWithLoadBalancer in AWS Tools for PowerShell Cmdlet Reference.

Remove-ELBInstanceFromLoadBalancer

The following code example shows how to use Remove-ELBInstanceFromLoadBalancer.

Tools for PowerShell

Example 1: This example removes the specified EC2 instance from the specified load balancer. You are prompted for confirmation before the operation proceeds, unless you also specify the Force parameter.

Remove-ELBInstanceFromLoadBalancer -LoadBalancerName my-load-balancer -Instance i-12345678

Output:

Confirm
Are you sure you want to perform this action?
Performing operation "Remove-ELBInstanceFromLoadBalancer (DeregisterInstancesFromLoadBalancer)" on Target "Amazon.ElasticLoadBalancing.Model.Instance".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):

InstanceId
----------
i-87654321
For API details, see DeregisterInstancesFromLoadBalancer in AWS Tools for PowerShell Cmdlet Reference.

Remove-ELBLoadBalancer

The following code example shows how to use Remove-ELBLoadBalancer.

Tools for PowerShell

Example 1: This example deletes the specified load balancer. You are prompted for confirmation before the operation proceeds, unless you also specify the Force parameter.

Remove-ELBLoadBalancer -LoadBalancerName my-load-balancer

Output:

Confirm
Are you sure you want to perform this action?
Performing operation "Remove-ELBLoadBalancer (DeleteLoadBalancer)" on Target "my-load-balancer".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):

For API details, see DeleteLoadBalancer in AWS Tools for PowerShell Cmdlet Reference.

Remove-ELBLoadBalancerListener

The following code example shows how to use Remove-ELBLoadBalancerListener.

Tools for PowerShell

Example 1: This example deletes the listener on port 80 for the specified load balancer. You are prompted for confirmation before the operation proceeds, unless you also specify the Force parameter.

Remove-ELBLoadBalancerListener -LoadBalancerName my-load-balancer -LoadBalancerPort 80

Output:
Confirm
Are you sure you want to perform this action?
Performing operation "Remove-ELBLoadBalancerListener (DeleteLoadBalancerListeners)"
on Target "80".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):

- For API details, see [DeleteLoadBalancerListeners](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Remove-ELBLoadBalancerPolicy

The following code example shows how to use Remove-ELBLoadBalancerPolicy.

**Tools for PowerShell**

**Example 1:** This example deletes the specified policy from the specified load balancer. You are prompted for confirmation before the operation proceeds, unless you also specify the Force parameter.

```
Remove-ELBLoadBalancerPolicy -LoadBalancerName my-load-balancer -PolicyName my-duration-cookie-policy
```

**Output:**

Confirm
Are you sure you want to perform this action?
Performing operation "Remove-ELBLoadBalancerPolicy (DeleteLoadBalancerPolicy)" on Target "my-duration-cookie-policy".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):

- For API details, see [DeleteLoadBalancerPolicy](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Remove-ELBResourceTag

The following code example shows how to use Remove-ELBResourceTag.
Tools for PowerShell

Example 1: This example removes the specified tag from the specified load balancer. You are prompted for confirmation before the operation proceeds, unless you also specify the Force parameter. The syntax used by this example requires PowerShell version 3 or later.

```
Remove-ELBResourceTag -LoadBalancerName my-load-balancer -Tag @{ Key="project" }
```

Output:

```
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-ELBResourceTag (RemoveTags)" on target
"Amazon.ElasticLoadBalancing.Model.TagKeyOnly".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):
```

Example 2: With Powershell version 2, you must use New-Object to create the tag for the Tag parameter.

```
$tag = New-Object Amazon.ElasticLoadBalancing.Model.TagKeyOnly
$tag.Key = "project"
Remove-ELBResourceTag -Tag $tag -Force
```

- For API details, see RemoveTags in AWS Tools for PowerShell Cmdlet Reference.

Set-ELBHealthCheck

The following code example shows how to use Set-ELBHealthCheck.

Tools for PowerShell

Example 1: This example configures the health check settings for the specified load balancer.

```
Set-ELBHealthCheck -LoadBalancerName my-load-balancer `
>> -HealthCheck_HealthyThreshold 2 `
>> -HealthCheck_UnhealthyThreshold 2 `
>> -HealthCheck_Target "HTTP:80/ping" `
>> -HealthCheck_Interval 30 `
>> -HealthCheck_Timeout 3
```
Output:

<table>
<thead>
<tr>
<th>HealthyThreshold</th>
<th>Interval</th>
<th>Target</th>
<th>Timeout</th>
<th>UnhealthyThreshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>30</td>
<td>HTTP:80/ping</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

- For API details, see [ConfigureHealthCheck](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Set-ELBLoadBalancerListenerSSLCertificate

The following code example shows how to use `Set-ELBLoadBalancerListenerSSLCertificate`.

**Tools for PowerShell**

**Example 1:** This example replaces the certificate that terminates the SSL connections for the specified listener.

```powershell
Set-ELBLoadBalancerListenerSSLCertificate -LoadBalancerName my-load-balancer `
>> -LoadBalancerPort 443 `
>> -SSLCertificateId "arn:aws:iam::123456789012:server-certificate/new-server-cert"
```

- For API details, see [SetLoadBalancerListenerSslCertificate](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Set-ELBLoadBalancerPolicyForBackendServer

The following code example shows how to use `Set-ELBLoadBalancerPolicyForBackendServer`.

**Tools for PowerShell**

**Example 1:** This example replaces the policies for the specified port with the specified policy.

```powershell
Set-ELBLoadBalancerPolicyForBackendServer -LoadBalancerName my-load-balancer - `
InstancePort 80 -PolicyName my-ProxyProtocol-policy
```
**Example 2: This example removes all policies associated with the specified port.**

```
Set-ELBLoadBalancerPolicyForBackendServer -LoadBalancerName my-load-balancer -InstancePort 80
```

- For API details, see [SetLoadBalancerPoliciesForBackendServer](#) in AWS Tools for PowerShell Cmdlet Reference.

**Set-ELBLoadBalancerPolicyOfListener**

The following code example shows how to use Set-ELBLoadBalancerPolicyOfListener.

**Tools for PowerShell**

**Example 1: This example replaces the policies for the specified listener with the specified policy.**

```
Set-ELBLoadBalancerPolicyOfListener -LoadBalancerName my-load-balancer -LoadBalancerPort 443 -PolicyName my-SSLNegotiation-policy
```

**Example 2: This example removes all policies associated with the specified listener.**

```
Set-ELBLoadBalancerPolicyOfListener -LoadBalancerName my-load-balancer -LoadBalancerPort 443
```

- For API details, see [SetLoadBalancerPoliciesOfListener](#) in AWS Tools for PowerShell Cmdlet Reference.

**Elastic Load Balancing - Version 2 examples using Tools for PowerShell**

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with Elastic Load Balancing - Version 2.

*Actions* are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

*Scenarios* are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.
Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

**Topics**
- **Actions**

**Actions**

**Add-ELB2ListenerCertificate**

The following code example shows how to use Add-ELB2ListenerCertificate.

**Tools for PowerShell**

**Example 1:** This example adds additional certificate to the specified Listener.

```powershell
```

**Output:**

<table>
<thead>
<tr>
<th>CertificateArn</th>
<th>IsDefault</th>
</tr>
</thead>
<tbody>
<tr>
<td>arn:aws:acm:us-east-1:123456789012:certificate/19478bd5-491d-47d4-b1d7-5217feba1d97</td>
<td>False</td>
</tr>
</tbody>
</table>

- For API details, see [AddListenerCertificates](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Add-ELB2Tag**

The following code example shows how to use Add-ELB2Tag.

**Tools for PowerShell**

**Example 1:** This example add new Tag to specified AWS.Tools.ElasticLoadBalancingV2 resource.
Add-ELB2Tag -ResourceArn 'arn:aws:elasticloadbalancing:us-east-1:123456789012:loadbalancer/app/test-alb/3651b4394dd9a24f' -Tag @{Key = 'productVersion'; Value = '1.0.0'}

- For API details, see AddTags in AWS Tools for PowerShell Cmdlet Reference.

Edit-ELB2Listener

The following code example shows how to use Edit-ELB2Listener.

Tools for PowerShell

Example 1: This example modifies the specified listeners default action to fixed-response.

```powershell
    "FixedResponseConfig" = @{
        "ContentType" = "text/plain"
        "MessageBody" = "Hello World"
        "StatusCode" = "200"
    }
    "Type" = [Amazon.ElasticLoadBalancingV2.ActionTypeEnum]::FixedResponse
}

Edit-ELB2Listener -ListenerArn 'arn:aws:elasticloadbalancing:us-east-1:123456789012:listener/app/testALB/3e2f03b558e19676/d19f2f14974db685' -Port 8080 -DefaultAction $newDefaultAction

Output:

Certificates    : {}
DefaultActions  : {Amazon.ElasticLoadBalancingV2.Model.Action}
ListenerArn     : arn:aws:elasticloadbalancing:us-east-1:123456789012:listener/app/testALB/3e2f03b558e19676/d19f2f14974db685
LoadBalancerArn : arn:aws:elasticloadbalancing:us-east-1:123456789012:loadbalancer/app/testALB/3e2f03b558e19676
Port            : 8080
Protocol        : HTTP
SslPolicy       :

- For API details, see ModifyListener in AWS Tools for PowerShell Cmdlet Reference.
**Edit-ELB2LoadBalancerAttribute**

The following code example shows how to use `Edit-ELB2LoadBalancerAttribute`.

**Tools for PowerShell**

**Example 1: This example modifies the Attributes of the specified load balancer.**

```
Edit-ELB2LoadBalancerAttribute -LoadBalancerArn 'arn:aws:elasticloadbalancing:us-east-1:123456789012:loadbalancer/app/test-alb/3651b4394dd9a24f' -Attribute @{'Key' = 'deletion_protection.enabled'; 'Value' = 'true'}
```

**Output:**

```
<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>deletion_protection.enabled</td>
<td>true</td>
</tr>
<tr>
<td>access_logs.s3.enabled</td>
<td>false</td>
</tr>
<tr>
<td>access_logs.s3.bucket</td>
<td></td>
</tr>
<tr>
<td>access_logs.s3.prefix</td>
<td></td>
</tr>
<tr>
<td>idle_timeout.timeout_seconds</td>
<td>60</td>
</tr>
<tr>
<td>routing.http2.enabled</td>
<td>true</td>
</tr>
<tr>
<td>routing.http.drop_invalid_header_fields.enabled</td>
<td>false</td>
</tr>
</tbody>
</table>
```

- For API details, see [ModifyLoadBalancerAttributes](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Edit-ELB2Rule**

The following code example shows how to use `Edit-ELB2Rule`.

**Tools for PowerShell**

**Example 1: This example modifies the specified Listener rule configurations.**

```powershell
    "PathPatternConfig" = @{
        "Values" = "/login1","/login2","/login3"
    }
    "Field" = "path-pattern"
}
```
Edit-ELB2Rule -RuleArn 'arn:aws:elasticloadbalancing:us-east-1:123456789012:listener-rule/app/testALB/3e2f03b558e19676/1c84f02aec143e80/f4f51dfaa033a8cc' -Condition $newRuleCondition

Output:

Actions : {Amazon.ElasticLoadBalancingV2.Model.Action}
Conditions : {Amazon.ElasticLoadBalancingV2.Model.RuleCondition}
IsDefault : False
Priority : 10
RuleArn : arn:aws:elasticloadbalancing:us-east-1:123456789012:listener-rule/app/testALB/3e2f03b558e19676/1c84f02aec143e80/f4f51dfaa033a8cc

For API details, see ModifyRule in AWS Tools for PowerShell Cmdlet Reference.

Edit-ELB2TargetGroup

The following code example shows how to use Edit-ELB2TargetGroup.

Tools for PowerShell

Example 1: This example modifies the properties of the specified Target Group.


Output:

HealthCheckEnabled : True
HealthCheckIntervalSeconds : 60
HealthCheckPath : /index.html
HealthCheckPort : 8080
HealthCheckProtocol : HTTP
HealthCheckTimeoutSeconds : 5
HealthyThresholdCount : 5
LoadBalancerArns : {}
Port : 80
Protocol : HTTP
TargetGroupArn : arn:aws:elasticloadbalancing:us-east-1:123456789012:targetgroup/test-tg/a4e04b3688be1970
TargetGroupName     : test-tg
TargetType          : instance
UnhealthyThresholdCount : 2
VpcId               : vpc-2cfd7000

- For API details, see [ModifyTargetGroup](#) in *AWS Tools for PowerShell Cmdlet Reference*.

## Edit-ELB2TargetGroupAttribute

The following code example shows how to use `Edit-ELB2TargetGroupAttribute`.

### Tools for PowerShell

#### Example 1: This example modifies the deregistration_delay attribute of the specified Target Group.

```powershell
Edit-ELB2TargetGroupAttribute -TargetGroupArn 'arn:aws:elasticloadbalancing:us-east-1:123456789012:targetgroup/test-tg/a4e04b3688be1970' -Attribute @{Key = 'deregistration_delay.timeout_seconds'; Value = 600}
```

### Output:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>stickiness.enabled</td>
<td>false</td>
</tr>
<tr>
<td>deregistration_delay.timeout_seconds</td>
<td>600</td>
</tr>
<tr>
<td>stickiness.type</td>
<td>lb_cookie</td>
</tr>
<tr>
<td>stickiness.lb_cookie.duration_seconds</td>
<td>86400</td>
</tr>
<tr>
<td>slow_start.duration_seconds</td>
<td>0</td>
</tr>
<tr>
<td>load_balancing.algorithm.type</td>
<td>round_robin</td>
</tr>
</tbody>
</table>

- For API details, see [ModifyTargetGroupAttributes](#) in *AWS Tools for PowerShell Cmdlet Reference*.

## Get-ELB2AccountLimit

The following code example shows how to use `Get-ELB2AccountLimit`.

### Tools for PowerShell

#### Example 1: This command lists ELB2 account limits for a given region.
Get-ELB2AccountLimit

Output:

Max   Name
---   ----
3000  target-groups
1000  targets-per-application-load-balancer
50    listeners-per-application-load-balancer
100   rules-per-application-load-balancer
50    network-load-balancers
3000  targets-per-network-load-balancer
500   targets-per-availability-zone-per-network-load-balancer
50    listeners-per-network-load-balancer
5     condition-values-per-alb-rule
5     condition-wildcards-per-alb-rule
100   target-groups-per-application-load-balancer
5     target-groups-per-action-on-application-load-balancer
1     target-groups-per-action-on-network-load-balancer
50    application-load-balancers

- For API details, see DescribeAccountLimits in AWS Tools for PowerShell Cmdlet Reference.

Get-ELB2Listener

The following code example shows how to use Get-ELB2Listener.

Tools for PowerShell

Example 1: This examples describes listeners of the specified ALB/NLB.


Output:

Certificates   : {}
DefaultActions : {Amazon.ElasticLoadBalancingV2.Model.Action}
ListenerArn    : arn:aws:elasticloadbalancing:us-east-1:123456789012:listener/app/test-alb/3651b4394dd9a24f/1dac07c21187d41e
Port            : 80
Protocol        : HTTP
SslPolicy       :
Certificates    : {Amazon.ElasticLoadBalancingV2.Model.Certificate}
DefaultActions  : {Amazon.ElasticLoadBalancingV2.Model.Action}
ListenerArn     : arn:aws:elasticloadbalancing:us-east-1:123456789012:listener/app/test-alb/3651b4394dd9a24f/66e10e3aaf5b6d9b
Port            : 443
Protocol        : HTTPS
SslPolicy       : ELBSecurityPolicy-2016-08

- For API details, see [DescribeListeners](#) in **AWS Tools for PowerShell Cmdlet Reference**.

**Get-ELB2ListenerCertificate**

The following code example shows how to use `Get-ELB2ListenerCertificate`.

**Tools for PowerShell**

**Example 1: This examples describes the certificate for the specified listener.**

```powershell
Get-ELB2ListenerCertificate -ListenerArn 'arn:aws:elasticloadbalancing:us-east-1:123456789012:listener/app/test-alb/3651b4394dd9a24f/66e10e3aaf5b6d9b'
```

**Output:**

```
CertificateArn | IsDefault |
--------------|-----------
arn:aws:acm:us-east-1:123456789012:certificate/5fc7c092-68bf-4862-969c-22fd48b6e17c | True |
```

- For API details, see [DescribeListenerCertificates](#) in **AWS Tools for PowerShell Cmdlet Reference**.
Get-ELB2LoadBalancer

The following code example shows how to use Get-ELB2LoadBalancer.

Tools for PowerShell

Example 1: This sample displays all the load balancers for the given region.

Get-ELB2LoadBalancer

Output:

AvailabilityZones : {us-east-1c}
CanonicalHostedZoneId : Z26RNL4JYFT0TI
CreatedTime : 6/22/18 11:21:50 AM
DNSName : test-elb1234567890-238d34ad8d94bc2e.elb.us-east-1.amazonaws.com
IpAddressType : ipv4
LoadBalancerName : test-elb1234567890
Scheme : internet-facing
SecurityGroups : {}
Type : network
VpcId : vpc-2cf00000

• For API details, see DescribeLoadBalancers in AWS Tools for PowerShell Cmdlet Reference.

Get-ELB2LoadBalancerAttribute

The following code example shows how to use Get-ELB2LoadBalancerAttribute.

Tools for PowerShell

Example 1: This command describes the attributes of given Load balancer.


Output:
### Key                               Value
---                               -----
access_logs.s3.enabled            false
load_balancing.cross_zone.enabled true
access_logs.s3.prefix
deletion_protection.enabled       false
access_logs.s3.bucket

- For API details, see [DescribeLoadBalancerAttributes](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-ELB2Rule

The following code example shows how to use Get-ELB2Rule.

**Tools for PowerShell**

**Example 1: This example describes the listener rules for the specified Listener ARN.**

```powershell
Get-ELB2Rule -ListenerArn 'arn:aws:elasticloadbalancing:us-east-1:123456789012:listener/app/test-alb/3651b4394dd9a24f/66e10e3aaf5b6d9b'
```

**Output:**

```
Actions    : {Amazon.ElasticLoadBalancingV2.Model.Action}
Conditions : {Amazon.ElasticLoadBalancingV2.Model.RuleCondition}
IsDefault  : False
Priority   : 1
RuleArn    : arn:aws:elasticloadbalancing:us-east-1:123456789012:listener-rule/app/test-alb/3651b4394dd9a24f/66e10e3aaf5b6d9b/2286fff5055e0f79

Actions    : {Amazon.ElasticLoadBalancingV2.Model.Action}
Conditions : {Amazon.ElasticLoadBalancingV2.Model.RuleCondition}
IsDefault  : False
Priority   : 2
RuleArn    : arn:aws:elasticloadbalancing:us-east-1:123456789012:listener-rule/app/test-alb/3651b4394dd9a24f/66e10e3aaf5b6d9b/14e7b036567623ba

Actions    : {Amazon.ElasticLoadBalancingV2.Model.Action}
Conditions : {}
IsDefault  : True
Priority   : default
```
• For API details, see DescribeRules in AWS Tools for PowerShell Cmdlet Reference.

Get-ELB2SSLPolicy

The following code example shows how to use Get-ELB2SSLPolicy.

Tools for PowerShell

Example 1: This examples lists all available listener policies for ElasticLoadBalancingV2.

Get-ELB2SSLPolicy

Output:

<table>
<thead>
<tr>
<th>Ciphers</th>
<th>Name</th>
<th>SslProtocols</th>
</tr>
</thead>
<tbody>
<tr>
<td>--------</td>
<td>------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>{ECDHE-ECDSA-AES128-GCM-SHA256, ECDHE-RSA-AES128-GCM-SHA256, ECDHE-ECDSA-AES128-SHA256, ECDHE-RSA-AES128-SHA256}</td>
<td>ELBSecurityPolicy-2016-08</td>
<td>{TLSv1, TLSv1.1, TLSv1.2}</td>
</tr>
</tbody>
</table>
For API details, see [DescribeSslPolicies](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-ELB2Tag**

The following code example shows how to use Get-ELB2Tag.

**Tools for PowerShell**

**Example 1: This example lists the Tags for the specified resource.**

```
Get-ELB2Tag -ResourceArn 'arn:aws:elasticloadbalancing:us-east-1:123456789012:loadbalancer/app/test-alb/3651b4394dd9a24f'
```

**Output:**

<table>
<thead>
<tr>
<th>ResourceArn</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>arn:aws:elasticloadbalancing:us-east-1:123456789012:loadbalancer/app/test-alb/3651b4394dd9a24f</td>
<td>{stage, internalName, version}</td>
</tr>
</tbody>
</table>

For API details, see [DescribeTags](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-ELB2TargetGroup**

The following code example shows how to use Get-ELB2TargetGroup.

**Tools for PowerShell**

**Example 1: This example describes the specified Target Group.**

```
Get-ELB2TargetGroup -TargetGroupArn 'arn:aws:elasticloadbalancing:us-east-1:123456789012:targetgroup/test-tg/a4e04b3688be1970'
```
Output:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>HealthCheckEnabled</td>
<td>True</td>
</tr>
<tr>
<td>HealthCheckIntervalSeconds</td>
<td>30</td>
</tr>
<tr>
<td>HealthCheckPath</td>
<td>/</td>
</tr>
<tr>
<td>HealthCheckPort</td>
<td>traffic-port</td>
</tr>
<tr>
<td>HealthCheckProtocol</td>
<td>HTTP</td>
</tr>
<tr>
<td>HealthCheckTimeoutSeconds</td>
<td>5</td>
</tr>
<tr>
<td>HealthyThresholdCount</td>
<td>5</td>
</tr>
<tr>
<td>LoadBalancerArns</td>
<td>{arn:aws:elasticloadbalancing:us-east-1:123456789012:loadbalancer/app/test-alb/3651b4394dd9a24f}</td>
</tr>
<tr>
<td>Port</td>
<td>80</td>
</tr>
<tr>
<td>Protocol</td>
<td>HTTP</td>
</tr>
<tr>
<td>TargetGroupArn</td>
<td>{arn:aws:elasticloadbalancing:us-east-1:123456789012:targetgroup/test-tg/a4e04b3688be1970}</td>
</tr>
<tr>
<td>TargetGroupName</td>
<td>test-tg</td>
</tr>
<tr>
<td>TargetType</td>
<td>instance</td>
</tr>
<tr>
<td>UnhealthyThresholdCount</td>
<td>2</td>
</tr>
<tr>
<td>VpcId</td>
<td>vpc-2cf7d00</td>
</tr>
</tbody>
</table>

- For API details, see [DescribeTargetGroups](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-ELB2TargetGroupAttribute**

The following code example shows how to use Get-ELB2TargetGroupAttribute.

**Tools for PowerShell**

**Example 1: This example describes the attributes of the specified Target Group.**

```
Get-ELB2TargetGroupAttribute -TargetGroupArn 'arn:aws:elasticloadbalancing:us-east-1:123456789012:targetgroup/test-tg/a4e04b3688be1970'
```

**Output:**

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
<td>-----------------------</td>
</tr>
<tr>
<td>stickiness.enabled</td>
<td>false</td>
</tr>
<tr>
<td>deregistration_delay.timeout_seconds</td>
<td>300</td>
</tr>
<tr>
<td>stickiness.type</td>
<td>lb_cookie</td>
</tr>
</tbody>
</table>
For API details, see [DescribeTargetGroupAttributes](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-ELB2TargetHealth

The following code example shows how to use Get-ELB2TargetHealth.

**Tools for PowerShell**

**Example 1:** This example returns the health status of the Targets present in the specified Target Group.

```powershell
Get-ELB2TargetHealth -TargetGroupArn 'arn:aws:elasticloadbalancing:us-east-1:123456789012:targetgroup/test-tg/a4e04b3688be1970'
```

**Output:**

<table>
<thead>
<tr>
<th>HealthCheckPort</th>
<th>Target</th>
<th>TargetHealth</th>
</tr>
</thead>
</table>

For API details, see [DescribeTargetHealth](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### New-ELB2Listener

The following code example shows how to use New-ELB2Listener.

**Tools for PowerShell**

**Example 1:** This example creates new ALB listener with the default action 'Forward' to send traffic to specified Target Group.

```powershell
    ForwardConfig = @{
        TargetGroups = @(}
```
@{ TargetGroupArn = "arn:aws:elasticloadbalancing:us-east-1:123456789012:targetgroup/testAlbTG/3d61c2f20a5bccc" }
   }
   TargetGroupStickinessConfig = @{
   DurationSeconds = 900
   Enabled = $true
   }
   Type = "Forward"
   }

New-ELB2Listener -LoadBalancerArn 'arn:aws:elasticloadbalancing:us-east-1:123456789012:loadbalancer/app/testALB/3e2f03b558e19676' -Port 8001 -Protocol "HTTP" -DefaultAction $defaultAction

Output:

Certificates : {}
DefaultActions : {Amazon.ElasticLoadBalancingV2.Model.Action}
ListenerArn : arn:aws:elasticloadbalancing:us-east-1:123456789012:listener/app/testALB/3e2f03b558e19676/1c84f02aec143e80
LoadBalancerArn : arn:aws:elasticloadbalancing:us-east-1:123456789012:loadbalancer/app/testALB/3e2f03b558e19676
Port : 8001
Protocol : HTTP
SslPolicy :

- For API details, see [CreateListener](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**New-ELB2LoadBalancer**

The following code example shows how to use New-ELB2LoadBalancer.

**Tools for PowerShell**

**Example 1: This example creates new internet facing Application load balancer with two subnets.**

New-ELB2LoadBalancer -Type application -Scheme internet-facing -IpAddressType ipv4 -Name 'New-Test-ALB' -SecurityGroup 'sg-07c3414abb8811cbd' -subnet 'subnet-c37a67a6','subnet-fc02eea0'
Output:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AvailabilityZones</td>
<td>{us-east-1b, us-east-1a}</td>
</tr>
<tr>
<td>CanonicalHostedZoneId</td>
<td>Z35SXDOTRQ7X7K</td>
</tr>
<tr>
<td>CreatedTime</td>
<td>12/28/19 2:58:03 PM</td>
</tr>
<tr>
<td>DNSName</td>
<td>New-Test-ALB-1391502222.us-east-1.elb.amazonaws.com</td>
</tr>
<tr>
<td>IpAddressType</td>
<td>ipv4</td>
</tr>
<tr>
<td>LoadBalancerName</td>
<td>New-Test-ALB</td>
</tr>
<tr>
<td>Scheme</td>
<td>internet-facing</td>
</tr>
<tr>
<td>SecurityGroups</td>
<td>{sg-07c3414abb8811cbd}</td>
</tr>
<tr>
<td>Type</td>
<td>application</td>
</tr>
<tr>
<td>VpcId</td>
<td>vpc-2cfd7000</td>
</tr>
</tbody>
</table>

- For API details, see [CreateLoadBalancer](#) in *AWS Tools for PowerShell Cmdlet Reference*.

New-ELB2Rule

The following code example shows how to use New-ELB2Rule.

Tools for PowerShell

Example 1: This example creates new Listener rule with fixed-response action based on the customer header value for the specified Listener.

```powershell
    "FixedResponseConfig" = @{
        "ContentType" = "text/plain"
        "MessageBody" = "Hello World"
        "StatusCode" = "200"
    }
    "Type" = [Amazon.ElasticLoadBalancingV2.ActionTypeEnum]::FixedResponse
}

    "httpHeaderConfig" = @{
        "HttpHeaderName" = "customHeader"
        "Values" = "header2","header1"
    }
    "Field" = "http-header"
}
New-ELB2Rule -ListenerArn 'arn:aws:elasticloadbalancing:us-east-1:123456789012:listener/app/testALB/3e2f03b558e19676/1c84f02aec143e80' -Action $newRuleAction -Condition $newRuleCondition -Priority 10

Output:

Actions    : {Amazon.ElasticLoadBalancingV2.Model.Action}
Conditions : {Amazon.ElasticLoadBalancingV2.Model.RuleCondition}
IsDefault  : False
Priority   : 10
RuleArn    : arn:aws:elasticloadbalancing:us-east-1:123456789012:listener-rule/app/testALB/3e2f03b558e19676/1c84f02aec143e80/f4f51dfaa033a8cc

For API details, see CreateRule in AWS Tools for PowerShell Cmdlet Reference.

New-ELB2TargetGroup

The following code example shows how to use New-ELB2TargetGroup.

Tools for PowerShell

Example 1: This example creates new Target group with the provided parameters.

New-ELB2TargetGroup -HealthCheckEnabled 1 -HealthCheckIntervalSeconds 30 -HealthCheckPath '/index.html' -HealthCheckPort 80 -HealthCheckTimeoutSecond 5 -HealthyThresholdCount 2 -UnhealthyThresholdCount 5 -Port 80 -Protocol 'HTTP' -TargetType instance -VpcId 'vpc-2cfd7000' -Name 'NewTargetGroup'

Output:

HealthCheckEnabled : True
HealthCheckIntervalSeconds : 30
HealthCheckPath : /index.html
HealthCheckPort : 80
HealthCheckProtocol : HTTP
HealthCheckTimeoutSeconds : 5
HealthyThresholdCount : 2
LoadBalancerArns : {}
Port: 80
Protocol: HTTP
TargetGroupName: NewTargetGroup
TargetType: instance
UnhealthyThresholdCount: 5
VpcId: vpc-2cfd7000

- For API details, see [CreateTargetGroup](#) in *AWS Tools for PowerShell Cmdlet Reference.*

### Register-ELB2Target

The following code example shows how to use `Register-ELB2Target`.

**Tools for PowerShell**

**Example 1:** This example registers 'i-0672a4c4cdeae3111' instance with the specified target group.

```
Register-ELB2Target -TargetGroupArn 'arn:aws:elasticloadbalancing:us-east-1:123456789012:targetgroup/test-tg/a4e04b3688be1970' -Target @{Port = 80; Id = 'i-0672a4c4cdeae3111'}
```

- For API details, see [RegisterTargets](#) in *AWS Tools for PowerShell Cmdlet Reference.*

### Remove-ELB2Listener

The following code example shows how to use `Remove-ELB2Listener`.

**Tools for PowerShell**

**Example 1:** This example deletes the specified Listener.

```
Remove-ELB2Listener -ListenerArn 'arn:aws:elasticloadbalancing:us-east-1:123456789012:listener/app/test-alb/3651b4394dd9a24f/66e10e3aaf5b6d9b'
```

**Output:**

```
Confirm
```
Example 2: This example removes specified listener from the Load balancer.

Remove-ELB2Listener -ListenerArn 'arn:aws:elasticloadbalancing:us-east-1:123456789012:listener/app/test-alb/3651b4394dd9a24f/66e10e3aaf5b6d9b'.

[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): y

Output:

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-ELB2Listener (DeleteListener)" on target "arn:aws:elasticloadbalancing:us-east-1:123456789012:listener/app/test-alb/3651b4394dd9a24f/66e10e3aaf5b6d9b".

[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): y

- For API details, see DeleteListener in AWS Tools for PowerShell Cmdlet Reference.

Remove-ELB2ListenerCertificate

The following code example shows how to use Remove-ELB2ListenerCertificate.

Tools for PowerShell

Example 1: This example removes specified certificate from the specified Target group.


Output:

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-ELB2ListenerCertificate (RemoveListenerCertificates)" on target "arn:aws:elasticloadbalancing:us-east-1:123456789012:listener/app/test-alb/3651b4394dd9a24f/3873f123b98f7618".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): y

- For API details, see RemoveListenerCertificates in AWS Tools for PowerShell Cmdlet Reference.

Remove-ELB2LoadBalancer

The following code example shows how to use Remove-ELB2LoadBalancer.

Tools for PowerShell

Example 1: This example deletes the specified Load balancer.

Remove-ELB2LoadBalancer -LoadBalancerArn 'arn:aws:elasticloadbalancing:us-east-1:123456789012:loadbalancer/app/test-alb/3651b4394dd9a24f'

Output:

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-ELB2LoadBalancer (DeleteLoadBalancer)" on target "arn:aws:elasticloadbalancing:us-east-1:123456789012:loadbalancer/app/test-alb/3651b4394dd9a24f".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): y

- For API details, see DeleteLoadBalancer in AWS Tools for PowerShell Cmdlet Reference.

Remove-ELB2Rule

The following code example shows how to use Remove-ELB2Rule.

Tools for PowerShell

Example 1: This example removes the specified rule from the Listener
Remove-ELB2Rule -RuleArn 'arn:aws:elasticloadbalancing:us-east-1:123456789012:listener-rule/app/test-alb/3651b4394dd9a24f/3873f123b98f7618/4b25eb10a42e33ab'

Output:

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-ELB2Rule (DeleteRule)" on target "arn:aws:elasticloadbalancing:us-east-1:123456789012:listener-rule/app/test-alb/3651b4394dd9a24f/3873f123b98f7618/4b25eb10a42e33ab".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): y

For API details, see DeleteRule in AWS Tools for PowerShell Cmdlet Reference.

Remove-ELB2Tag

The following code example shows how to use Remove-ELB2Tag.

Tools for PowerShell

Example 1: This example removes the tag for the specified key.

Remove-ELB2Tag -ResourceArn 'arn:aws:elasticloadbalancing:us-east-1:123456789012:loadbalancer/app/test-alb/3651b4394dd9a24f' -TagKey 'productVersion'

Output:

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-ELB2Tag (RemoveTags)" on target "arn:aws:elasticloadbalancing:us-east-1:123456789012:loadbalancer/app/test-alb/3651b4394dd9a24f".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): y

For API details, see RemoveTags in AWS Tools for PowerShell Cmdlet Reference.
**Remove-ELB2TargetGroup**

The following code example shows how to use Remove-ELB2TargetGroup.

**Tools for PowerShell**

**Example 1: This example removes the specified Target Group.**

```powershell
Remove-ELB2TargetGroup -TargetGroupArn 'arn:aws:elasticloadbalancing:us-east-1:123456789012:targetgroup/testssss/4e0b6076bc6483a7'
```

**Output:**

```
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-ELB2TargetGroup (DeleteTargetGroup)" on target "arn:aws:elasticloadbalancing:us-east-1:123456789012:targetgroup/testssss/4e0b6076bc6483a7".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):
y
• For API details, see [DeleteTargetGroup](#) in *AWS Tools for PowerShell Cmdlet Reference.*
```

**Set-ELB2IpAddressType**

The following code example shows how to use Set-ELB2IpAddressType.

**Tools for PowerShell**

**Example 1: This example changes Load balancer IP address type from 'IPv4' to 'DualStack'.**

```powershell
Set-ELB2IpAddressType -LoadBalancerArn 'arn:aws:elasticloadbalancing:us-east-1:123456789012:loadbalancer/app/test-alb/3651b4394dd9a24f' -IpAddressType dualstack
```

**Output:**

```
Value ---- dualstack
```
• For API details, see `SetIpAddressType` in *AWS Tools for PowerShell Cmdlet Reference*.

**Set-ELB2RulePriority**

The following code example shows how to use Set-ELB2RulePriority.

**Tools for PowerShell**

**Example 1:** This example changes the priority of the specified listener rule.

```powershell
Set-ELB2RulePriority -RulePriority @{Priority = 11; RuleArn = 'arn:aws:elasticloadbalancing:us-east-1:123456789012:listener-rule/app/test-alb/3651b4394dd9a24f/a4eb199fa5046f80/dbf4c6dceef3ec6f8'}
```

**Output:**

```
Actions    : {Amazon.ElasticLoadBalancingV2.Model.Action}
Conditions : {Amazon.ElasticLoadBalancingV2.Model.RuleCondition}
IsDefault  : False
Priority   : 11
RuleArn    : arn:aws:elasticloadbalancing:us-east-1:123456789012:listener-rule/app/test-alb/3651b4394dd9a24f/a4eb199fa5046f80/dbf4c6dceef3ec6f8
```

• For API details, see `SetRulePriorities` in *AWS Tools for PowerShell Cmdlet Reference*.

**Set-ELB2SecurityGroup**

The following code example shows how to use Set-ELB2SecurityGroup.

**Tools for PowerShell**

**Example 1:** This example adds security group 'sg-07c3414abb8811cbd' to the specified Load balancer.

```powershell
```

**Output:**
Set-ELB2Subnet

The following code example shows how to use Set-ELB2Subnet.

Tools for PowerShell

Example 1: This example modifies the subnets of the specified Load balancer.

```
Set-ELB2Subnet -LoadBalancerArn 'arn:aws:elasticloadbalancing:us-east-1:123456789012:loadbalancer/app/test-alb/3651b4394dd9a24f' -Subnet 'subnet-7d8a0a51','subnet-c37a67a6'
```

Output:

<table>
<thead>
<tr>
<th>LoadBalancerAddresses</th>
<th>SubnetId</th>
<th>ZoneName</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>subnet-7d8a0a51</td>
<td>us-east-1c</td>
</tr>
<tr>
<td>{}</td>
<td>subnet-c37a67a6</td>
<td>us-east-1b</td>
</tr>
</tbody>
</table>

Unregister-ELB2Target

The following code example shows how to use Unregister-ELB2Target.

Tools for PowerShell

Example 1: This example deregisters instance 'i-0672a4c4cdeae3111' from the specified Target group.

```
$targetDescription = New-Object Amazon.ElasticLoadBalancingV2.Model.TargetDescription
$targetDescription.Id = 'i-0672a4c4cdeae3111'
Unregister-ELB2Target -Target $targetDescription -TargetGroupArn 'arn:aws:elasticloadbalancing:us-east-1:123456789012:targetgroup/test-tg/a4e04b3688be1970'
```
Amazon FSx examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with Amazon FSx.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

• Actions

Actions

Add-FSXResourceTag

The following code example shows how to use Add-FSXResourceTag.

Tools for PowerShell

Example 1: This example adds tags to the given resource.

```powershell
Add-FSXResourceTag -ResourceARN "arn:aws:fsx.eu-west-1:123456789012:file-system/fs-01cd23bc4bdf5678a" -Tag @{Key="Users";Value="Test"} -PassThru
```

Output:

```
arn:aws:fsx.eu-west-1:123456789012:file-system/fs-01cd23bc4bdf5678a
```

• For API details, see TagResource in AWS Tools for PowerShell Cmdlet Reference.
Get-FSXBackup

The following code example shows how to use Get-FSXBackup.

Example 1: This example fetches backups created since yesterday for the given file system id.

```powershell
Get-FSXBackup -Filter @{Name="file-system-id";Values=$fsx.FileSystemId} | Where-Object CreationTime -gt (Get-Date).AddDays(-1)
```

Output:

```
BackupId        : backup-01dac234e56782bcc
CreationTime    : 6/14/2019 3:35:14 AM
FailureDetails  :
FileSystem      : Amazon.FSx.Model.FileSystem
KmsKeyId        : arn:aws:kms:eu-west-1:123456789012:key/f1af23c4-1b23-1bde-a1f1-e1234c5af123
Lifecycle       : AVAILABLE
ProgressPercent : 100
Tags            : {}
Type            : AUTOMATIC
```

• For API details, see [DescribeBackups](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-FSXFileSystem

The following code example shows how to use Get-FSXFileSystem.

Example 1: This example returns the description of given filesystemId.

```powershell
Get-FSXFileSystem -FileSystemId fs-01cd23bc4bdf5678a
```

Output:

```
CreationTime         : 1/17/2019 9:55:30 AM
DNSName              : fs-01cd23bc4bdf5678a.ktmsad.local
```

Amazon FSx
Get-FSXResourceTagList

The following code example shows how to use Get-FSXResourceTagList.

Example 1: This example lists tags for provided resource arn.

```
```

**Output:**

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSx-Service</td>
<td>Windows</td>
</tr>
<tr>
<td>Users</td>
<td>Dev</td>
</tr>
</tbody>
</table>

New-FSXBackup

The following code example shows how to use New-FSXBackup.
Tools for PowerShell

Example 1: This example creates a back up of the given file system.

```powershell
New-FSXBackup -FileSystemId fs-0b1fac2345623456ba
```

Output:

```
BackupId        : backup-0b1fac2345623456ba
CreationTime    : 6/14/2019 5:37:17 PM
FailureDetails  :
FileSystem      : Amazon.FSx.Model.FileSystem
KmsKeyId        : arn:aws:kms:eu-west-1:123456789012:key/f1af23c4-1b23-1bde-a1f3-e1234c5af678
Lifecycle       : CREATING
ProgressPercent : 0
ResourceARN     : arn:aws:fsx:eu-west-1:123456789012:backup/backup-0b1fac2345623456ba
Tags            : {}
Type            : USER_INITIATED
```

- For API details, see [CreateBackup](#) in *AWS Tools for PowerShell Cmdlet Reference*.

New-FSXFileSystem

The following code example shows how to use `New-FSXFileSystem`.

Tools for PowerShell

Example 1: This example creates a new 300GB Windows file system, permitting access from the specified subnet, that supports throughput up to 8 megabytes per second. The new file system is automatically joined to the specified Microsoft Active Directory.

```powershell
New-FSXFileSystem -FileSystemType WINDOWS -StorageCapacity 300 -SubnetId subnet-1a2b3c4d5e6f -WindowsConfiguration @{ThroughputCapacity=8;ActiveDirectoryId='d-1a2b3c4d'}
```

Output:

```
CreationTime         : 12/10/2018 6:06:59 PM
DNSName              : fs-abcdef01234567890.example.com
FailureDetails       :
```

Amazon FSx
For API details, see [CreateFileSystem](https://docs.aws.amazon.com/PowerShell/latest/release/PowerShellGuide.html#CreateFileSystem) in *AWS Tools for PowerShell Cmdlet Reference*.

**New-FSXFileSystemFromBackup**

The following code example shows how to use `New-FSXFileSystemFromBackup`.

### Tools for PowerShell

**Example 1:** This example creates new Amazon FSx file system from an existing Amazon FSx for Windows File Server backup.

```
New-FSXFileSystemFromBackup -BackupId $backupID -Tag @{Key="tag:Name";Value="from-manual-backup"} -SubnetId $SubnetID -SecurityGroupId $SG_ID -WindowsConfiguration @{ThroughputCapacity=8;ActiveDirectoryId=$DirectoryID}
```

**Output:**

```
CreationTime         : 8/8/2019 12:59:58 PM
DNSName              : fs-012ff34e56789120.ktmsad.local
FailureDetails       : 
FileSystemId         : fs-012ff34e56789120
FileSystemType       : WINDOWS
KmsKeyId             : arn:aws:kms:eu-west-1:123456789012:key/f1af23c4-5b67-1bde-a2f3-e4567c8a9321
Lifecycle            : CREATING
LustreConfiguration  : 
NetworkInterfaceIds  : 
OwnerId              : 123456789012
StorageCapacity      : 300
SubnetIds            : {subnet-1a2b3c4d5e6f}
Tags                 : 
VpcId                : vpc-1a2b3c4d5e6f
WindowsConfiguration : Amazon.FSx.Model.WindowsFileSystemConfiguration
```
NetworkInterfaceIds : {}
OwnerId              : 933303704102
StorageCapacity      : 300
SubnetIds            : {subnet-fa1ae23c}
Tags                 : {tag:Name}
VpcId                : vpc-12cf3b4f
WindowsConfiguration : Amazon.FSx.Model.WindowsFileSystemConfiguration

• For API details, see CreateFileSystemFromBackup in AWS Tools for PowerShell Cmdlet Reference.

Remove-FSXBackup

The following code example shows how to use Remove-FSXBackup.

Tools for PowerShell

Example 1: This example removes the given backup-id.

Remove-FSXBackup -BackupId $backupID

Output:

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-FSXBackup (DeleteBackup)" on target
"backup-0bbca1e2345678e12".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y") : Y

BackupId                 Lifecycle
--------                 ---------
backup-0bbca1e2345678e12 DELETED

• For API details, see DeleteBackup in AWS Tools for PowerShell Cmdlet Reference.

Remove-FSXFileSystem

The following code example shows how to use Remove-FSXFileSystem.
Tools for PowerShell

**Example 1: This example removes the given FSX file system ID.**

```
Remove-FSXFileSystem -FileSystemId fs-012ff34e567890120
```

**Output:**

```
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-FSXFileSystem (DeleteFileSystem)" on target "fs-012ff34e567890120".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): Y

FileSystemId         Lifecycle WindowsResponse
------------         --------- ---------------
fs-012ff34e567890120 DELETING  Amazon.FSx.Model.DeleteFileSystemWindowsResponse
```

- For API details, see [DeleteFileSystem](https://aws.amazon.com/tools/powershell/cmdlet-reference) in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-FSXResourceTag**

The following code example shows how to use Remove-FSXResourceTag.

**Tools for PowerShell**

**Example 1: This example removes the resource tag for the given FSX file system resource ARN.**

```
Remove-FSXResourceTag -ResourceARN $FSX.ResourceARN -TagKey Users
```

**Output:**

```
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-FSXResourceTag (UntagResource)" on target "arn:aws:fsx:eu-west-1:933303704102:file-system/fs-07cd45bc6bdf2674a".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): Y
```
• For API details, see UnTagResource in AWS Tools for PowerShell Cmdlet Reference.

Update-FSXFileSystem

The following code example shows how to use Update-FSXFileSystem.

Tools for PowerShell

Example 1: This example updates the FSX file system automatic backup retention days via UpdateFileSystemWindowsConfiguration.

```powershell
```

Output:

```
CreationTime         : 1/17/2019 9:55:30 AM
DNSName              : fs-01cd23bc4bdf5678a.ktmsad.local
FailureDetails       :
FileSystemId         : fs-01cd23bc4bdf5678a
FileSystemType       : WINDOWS
KmsKeyId             : arn:aws:kms:eu-west-1:123456789012:key/f1af23c4-1b23-1bde-a1f2-e12345af678
Lifecycle            : AVAILABLE
LustreConfiguration  :
NetworkInterfaceIds  : {eni-01cd23bc4bdf5678a}
OwnerId              : 933303704102
ResourceARN          : arn:aws:fsx:eu-west-1:933303704102:file-system/fs-07cd45bc6bdf2674a
StorageCapacity      : 300
SubnetIds            : {subnet-1d234567}
Tags                 : {FSx-Service}
VpcId                : vpc-23cf4b5f
WindowsConfiguration : Amazon.FSx.Model.WindowsFileSystemConfiguration
```

• For API details, see UpdateFileSystem in AWS Tools for PowerShell Cmdlet Reference.
AWS Glue examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with AWS Glue.

*Actions* are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

*Scenarios* are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

**Topics**

- *Actions*

**Actions**

**New-GLUEJob**

The following code example shows how to use `New-GLUEJob`.

**Tools for PowerShell**

Example 1: This example creates a new job in AWS Glue. The command name value is always `glueetl`. AWS Glue supports running job scripts written in Python or Scala. In this example, the job script (MyTestGlueJob.py) is written in Python. Python parameters are specified in the `$DefArgs` variable, and then passed to the PowerShell command in the `DefaultArguments` parameter, which accepts a hashtable. The parameters in the `$JobParams` variable come from the CreateJob API, documented in the Jobs (https://docs.aws.amazon.com/glue/latest/dg/aws-glue-api-jobs-job.html) topic of the AWS Glue API reference.

```powershell
$Command = New-Object Amazon.Glue.Model.JobCommand
$Command.Name = 'glueetl'
$Command.ScriptLocation = 's3://aws-glue-scripts-000000000000-us-west-2/admin/MyTestGlueJob.py'
```
$Command

$Source = "source_test_table"
$Target = "target_test_table"
$Connections = $Source, $Target

$DefArgs = @{
    '--TempDir' = 's3://aws-glue-temporary-000000000000-us-west-2/admin'
    '--job-bookmark-option' = 'job-bookmark-disable'
    '--job-language' = 'python'
}

$DefArgs

$ExecutionProp = New-Object Amazon.Glue.Model.ExecutionProperty
$ExecutionProp.MaxConcurrentRuns = 1

$JobParams = @{
    "AllocatedCapacity" = "5"
    "Command" = $Command
    "Connections_Connection" = $Connections
    "DefaultArguments" = $DefArgs
    "Description" = "This is a test"
    "ExecutionProperty" = $ExecutionProp
    "MaxRetries" = "1"
    "Name" = "MyOregonTestGlueJob"
    "Role" = "Amazon-GlueServiceRoleForSSM"
    "Timeout" = "20"
}

New-GlueJob @JobParams

- For API details, see CreateJob in AWS Tools for PowerShell Cmdlet Reference.

AWS Health examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with AWS Health.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.
Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

• Actions

Actions

Get-HLTHEvent

The following code example shows how to use Get-HLTHEvent.

Tools for PowerShell

Example 1: This command returns events from AWS Personal Health Dashboard. The user adds the -Region parameter to see events available to the service in the US East (N. Virginia) Region, but the -Filter_Region parameter filters for events that are logged in the EU (London) and US West (Oregon) Regions (eu-west-2 and us-west-2). The -Filter_StartTime parameter filters for a range of times that events can start, while the -Filter_EndTime parameter filters for a range of times that events can end. The result is a scheduled maintenance event for RDS that starts within the specified -Filter_StartTime range, and ends within the scheduled -Filter_EndTime range.

Get-HLTHEvent -Region us-east-1 -Filter_Region "eu-west-2","us-west-2" -Filter_StartTime @{from="3/14/2019 6:30:00AM";to="3/15/2019 5:00:00PM"} -Filter_EndTime @{from="3/21/2019 7:00:00AM";to="3/21/2019 5:00:00PM"}

Output:

<table>
<thead>
<tr>
<th>Arn</th>
<th>arn:aws:health:us-west-2::event/RDS/AWS_RDS_HARDWARE_MAINTENANCE_SCHEDULED/AWS_RDS_HARDWARE_MAINTENANCE_SCHEDULED_USW2_20190314_20190321</th>
</tr>
</thead>
<tbody>
<tr>
<td>AvailabilityZone</td>
<td>3/21/2019 2:00:00 PM</td>
</tr>
<tr>
<td>EventTypeCategory</td>
<td>scheduledChange</td>
</tr>
<tr>
<td>EventTypeCode</td>
<td>AWS_RDS_HARDWARE_MAINTENANCE_SCHEDULED</td>
</tr>
<tr>
<td>LastUpdatedTime</td>
<td>2/28/2019 2:26:07 PM</td>
</tr>
</tbody>
</table>
IAM examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with IAM.

*Actions* are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

*Scenarios* are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

**Topics**

- **Actions**

**Actions**

**Add-IAMClientIDToOpenIDConnectProvider**

The following code example shows how to use `Add-IAMClientIDToOpenIDConnectProvider`.

**Tools for PowerShell**

*Example 1: This command adds the client ID (or audience) my-application-ID to the existing OIDC provider named server.example.com.*

```
Add-IAMClientIDToOpenIDConnectProvider -ClientID "my-application-ID"
-OpenIDConnectProviderARN "arn:aws:iam::123456789012:oidc-provider/server.example.com"
```
For API details, see AddClientIdToOpenIdConnectProvider in AWS Tools for PowerShell Cmdlet Reference.

Add-IAMRoleTag

The following code example shows how to use Add-IAMRoleTag.

Tools for PowerShell

Example 1: This example adds tag to Role in Identity Management Service

Add-IAMRoleTag -RoleName AdminRoleaccess -Tag @{'Key = 'abac'; Value = 'testing'}

For API details, see TagRole in AWS Tools for PowerShell Cmdlet Reference.

Add-IAMRoleToInstanceProfile

The following code example shows how to use Add-IAMRoleToInstanceProfile.

Tools for PowerShell

Example 1: This command adds the role named S3Access to an existing instance profile named webserver. To create the instance profile, use the New-IAMInstanceProfile command. After you create the instance profile and associate it with a role using this command, you can attach it to an EC2 instance. To do that, use the New-EC2Instance cmdlet with either the InstanceProfile_Arn or the InstanceProfile-Name parameter to launch the new instance.

Add-IAMRoleToInstanceProfile -RoleName "S3Access" -InstanceProfileName "webserver"

For API details, see AddRoleToInstanceProfile in AWS Tools for PowerShell Cmdlet Reference.

Add-IAMUserTag

The following code example shows how to use Add-IAMUserTag.

Tools for PowerShell

Example 1: This example adds tag to User in Identity Management Service
Add-IAMUserTag

Add-IAMUserTag -UserName joe -Tag @{ Key = 'abac'; Value = 'testing'}

- For API details, see TagUser in AWS Tools for PowerShell Cmdlet Reference.

Add-IAMUserToGroup

The following code example shows how to use Add-IAMUserToGroup.

Tools for PowerShell

Example 1: This command adds the user named Bob to the group named Admins.

Add-IAMUserToGroup -UserName "Bob" -GroupName "Admins"

- For API details, see AddUserToGroup in AWS Tools for PowerShell Cmdlet Reference.

Disable-IAMMFADevice

The following code example shows how to use Disable-IAMMFADevice.

Tools for PowerShell

Example 1: This command disables the hardware MFA device associated with the user Bob that has the serial number 123456789012.

Disable-IAMMFADevice -UserName "Bob" -SerialNumber "123456789012"

Example 2: This command disables the virtual MFA device associated with the user David that has the ARN arn:aws:iam::210987654321:mfa/David. Note that virtual MFA device is not deleted from the account. The virtual device is still present and appears in the output of the Get-IAMVirtualMFADevice command. Before you can create a new virtual MFA device for the same user, you must delete the old one by using the Remove-IAMVirtualMFADevice command.

Disable-IAMMFADevice -UserName "David" -SerialNumber "arn:aws:iam::210987654321:mfa/David"

- For API details, see DeactivateMfaDevice in AWS Tools for PowerShell Cmdlet Reference.
Edit-IAMPassword

The following code example shows how to use Edit-IAMPassword.

Tools for PowerShell

Example 1: This command changes the password for the user that is running the command. This command can be called by IAM users only. If this command is called when you are signed-in with AWS account (root) credentials, the command returns an InvalidUserType error.

```
Edit-IAMPassword -OldPassword "MyOldP@ssw0rd" -NewPassword "MyNewP@ssw0rd"
```

- For API details, see ChangePassword in AWS Tools for PowerShell Cmdlet Reference.

Enable-IAMMFADevice

The following code example shows how to use Enable-IAMMFADevice.

Tools for PowerShell

Example 1: This command enables the hardware MFA device with the serial number 987654321098 and associates the device with the user Bob. It includes the first two codes in sequence from the device.

```
Enable-IAMMFADevice -UserName "Bob" -SerialNumber "987654321098" -AuthenticationCode1 "12345678" -AuthenticationCode2 "87654321"
```

Example 2: This example creates and enables a virtual MFA device. The first command creates the virtual device and returns the device's object representation in the variable $MFADevice. You can use the .Base32StringSeed or QRCodePng properties to configure the user's software application. The final command assigns the device to the user David, identifying the device by its serial number. The command also synchronizes the device with AWS by including the first two codes in sequence from the virtual MFA device.

```
$MFADevice = New-IAMVirtualMFADevice -VirtualMFADeviceName "MyMFADevice"
# see example for New-IAMVirtualMFADevice to see how to configure the software program with PNG or base32 seed code
```
Enable-IAMMFADevice -UserName "David" -SerialNumber -SerialNumber
$MFADevice.SerialNumber -AuthenticationCode1 "24681357" -AuthenticationCode2 "13572468"

- For API details, see EnableMfaDevice in AWS Tools for PowerShell Cmdlet Reference.

Get-IAMAccessKey

The following code example shows how to use Get-IAMAccessKey.

Tools for PowerShell

Example 1: This command lists the access keys for the IAM user named Bob. Note that you cannot list the secret access keys for IAM users. If the secret access keys are lost, you must create new access keys with the New-IAMAccessKey cmdlet.

Get-IAMAccessKey -UserName "Bob"

Output:

<table>
<thead>
<tr>
<th>AccessKeyId</th>
<th>CreateDate</th>
<th>Status</th>
<th>UserName</th>
</tr>
</thead>
<tbody>
<tr>
<td>AKIAIOSFODNN7EXAMPLE</td>
<td>12/3/2014 10:53:41 AM</td>
<td>Active</td>
<td>Bob</td>
</tr>
<tr>
<td>AKIAI44Q5H8DBHEXAMPLE</td>
<td>6/6/2013 8:42:26 PM</td>
<td>Inactive</td>
<td>Bob</td>
</tr>
</tbody>
</table>

- For API details, see ListAccessKeys in AWS Tools for PowerShell Cmdlet Reference.

Get-IAMAccessKeyLastUsed

The following code example shows how to use Get-IAMAccessKeyLastUsed.

Tools for PowerShell

Example 1: Returns the owning user name and last-usage information for the supplied access key.

Get-IAMAccessKeyLastUsed -AccessKeyId ABCDEXAMPLE

- For API details, see GetAccessKeyLastUsed in AWS Tools for PowerShell Cmdlet Reference.
Get-IAMAccountAlias

The following code example shows how to use Get-IAMAccountAlias.

Tools for PowerShell

Example 1: This command returns the account alias for the AWS account.

Get-IAMAccountAlias

Output:

ExampleCo

- For API details, see ListAccountAliases in AWS Tools for PowerShell Cmdlet Reference.

Get-IAMAccountAuthorizationDetail

The following code example shows how to use Get-IAMAccountAuthorizationDetail.

Tools for PowerShell

Example 1: This example gets authorization details about the identities in the AWS account, and displays the element list of the returned object, including users, groups, and roles. For example, the UserDetailList property displays details about the users. Similar information is available in the RoleDetailList and GroupDetailList properties.

$Details=Get-IAMAccountAuthorizationDetail
$Details

Output:

GroupDetailList : {Administrators, Developers, Testers, Backup}
IsTruncated     : False
Marker          :
RoleDetailList  : {TestRole1, AdminRole, TesterRole, clirole...}
UserDetailList  : {Administrator, Bob, BackupToS3, }

$Details.UserDetailList
### Output:

<table>
<thead>
<tr>
<th>Arn</th>
<th>arn:aws:iam::123456789012:user/Administrator</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateDate</td>
<td>10/16/2014 9:03:09 AM</td>
</tr>
<tr>
<td>GroupList</td>
<td>{Administrators}</td>
</tr>
<tr>
<td>Path</td>
<td>/</td>
</tr>
<tr>
<td>UserId</td>
<td>AIDACKCEVSQ6CEXAMPLE1</td>
</tr>
<tr>
<td>UserName</td>
<td>Administrator</td>
</tr>
<tr>
<td>UserPolicyList</td>
<td>{}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arn</th>
<th>arn:aws:iam::123456789012:user/Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateDate</td>
<td>4/6/2015 12:54:42 PM</td>
</tr>
<tr>
<td>GroupList</td>
<td>{Developers}</td>
</tr>
<tr>
<td>Path</td>
<td>/</td>
</tr>
<tr>
<td>UserId</td>
<td>AIDACKCEVSQ6CEXAMPLE2</td>
</tr>
<tr>
<td>UserName</td>
<td>bab</td>
</tr>
<tr>
<td>UserPolicyList</td>
<td>{}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arn</th>
<th>arn:aws:iam::123456789012:user/BackupToS3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateDate</td>
<td>1/27/2015 10:15:08 AM</td>
</tr>
<tr>
<td>GroupList</td>
<td>{Backup}</td>
</tr>
<tr>
<td>Path</td>
<td>/</td>
</tr>
<tr>
<td>UserId</td>
<td>AIDACKCEVSQ6CEXAMPLE3</td>
</tr>
<tr>
<td>UserName</td>
<td>BackupToS3</td>
</tr>
<tr>
<td>UserPolicyList</td>
<td>{BackupServicePermissionsToS3Buckets}</td>
</tr>
</tbody>
</table>

- For API details, see [GetAccountAuthorizationDetails](#) in *AWS Tools for PowerShell Cmdlet Reference*.

---

**Get-IAMAccountPasswordPolicy**

The following code example shows how to use Get-IAMAccountPasswordPolicy.

**Tools for PowerShell**

**Example 1:** This example returns details about the password policy for the current account. If no password policy is defined for the account, the command returns a **NoSuchEntity** error.

Get-IAMAccountPasswordPolicy

**Output:**
AllowUsersToChangePassword : True
ExpirePasswords : True
HardExpiry : False
MaxPasswordAge : 90
MinimumPasswordLength : 8
PasswordReusePrevention : 20
RequireLowercaseCharacters : True
RequireNumbers : True
RequireSymbols : False
RequireUppercaseCharacters : True

- For API details, see [GetAccountPasswordPolicy](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-IAMAccountSummary**

The following code example shows how to use `Get-IAMAccountSummary`.

**Tools for PowerShell**

**Example 1:** This example returns information about the current IAM entity usage and current IAM entity quotas in the AWS account.

```powershell
Get-IAMAccountSummary
```

**Output:**

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users</td>
<td>7</td>
</tr>
<tr>
<td>GroupPolicySizeQuota</td>
<td>5120</td>
</tr>
<tr>
<td>PolicyVersionsInUseQuota</td>
<td>10000</td>
</tr>
<tr>
<td>ServerCertificatesQuota</td>
<td>20</td>
</tr>
<tr>
<td>AccountSigningCertificatesPresent</td>
<td>0</td>
</tr>
<tr>
<td>AccountAccessKeysPresent</td>
<td>0</td>
</tr>
<tr>
<td>Groups</td>
<td>3</td>
</tr>
<tr>
<td>UsersQuota</td>
<td>5000</td>
</tr>
<tr>
<td>RolePolicySizeQuota</td>
<td>10240</td>
</tr>
<tr>
<td>UserPolicySizeQuota</td>
<td>2048</td>
</tr>
<tr>
<td>GroupsPerUserQuota</td>
<td>10</td>
</tr>
<tr>
<td>AssumeRolePolicySizeQuota</td>
<td>2048</td>
</tr>
<tr>
<td>AttachedPoliciesPerGroupQuota</td>
<td>2</td>
</tr>
</tbody>
</table>
Roles                                      9
VersionsPerPolicyQuota                    5
GroupsQuota                                100
PolicySizeQuota                            5120
Policies                                   5
RolesQuota                                 250
ServerCertificates                         0
AttachedPoliciesPerRoleQuota               2
MFADevicesInUse                            2
PoliciesQuota                              1000
AccountMFAEnabled                          1
Providers                                  2
InstanceProfilesQuota                      100
MFADevices                                 4
AccessKeysPerUserQuota                     2
AttachedPoliciesPerUserQuota               2
SigningCertificatesPerUserQuota            2
PolicyVersionsInUse                        4
InstanceProfiles                           1
...  

- For API details, see GetAccountSummary in AWS Tools for PowerShell Cmdlet Reference.

Get-IAMAttachedGroupPolicyList

The following code example shows how to use Get-IAMAttachedGroupPolicyList.

Tools for PowerShell

**Example 1:** This command returns the names and ARNs of the managed policies that are attached to the IAM group named Admins in the AWS account. To see the list of inline policies embedded in the group, use the Get-IAMGroupPolicyList command.

```bash
Get-IAMAttachedGroupPolicyList -GroupName "Admins"
```

**Output:**

<table>
<thead>
<tr>
<th>PolicyArn</th>
<th>PolicyName</th>
</tr>
</thead>
<tbody>
<tr>
<td>arn:aws:iam::aws:policy/SecurityAudit</td>
<td>SecurityAudit</td>
</tr>
<tr>
<td>arn:aws:iam::aws:policy/AdministratorAccess</td>
<td>AdministratorAccess</td>
</tr>
</tbody>
</table>
• For API details, see [ListAttachedGroupPolicies](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

**Get-IAMAttachedRolePolicyList**

The following code example shows how to use `Get-IAMAttachedRolePolicyList`.

**Tools for PowerShell**

**Example 1:** This command returns the names and ARNs of the managed policies attached to the IAM role named `SecurityAuditRole` in the AWS account. To see the list of inline policies that are embedded in the role, use the `Get-IAMRolePolicyList` command.

```
Get-IAMAttachedRolePolicyList -RoleName "SecurityAuditRole"
```

**Output:**

<table>
<thead>
<tr>
<th>PolicyArn</th>
<th>PolicyName</th>
</tr>
</thead>
<tbody>
<tr>
<td>arn:aws:iam::aws:policy/SecurityAudit</td>
<td>SecurityAudit</td>
</tr>
</tbody>
</table>

• For API details, see [ListAttachedRolePolicies](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

**Get-IAMAttachedUserPolicyList**

The following code example shows how to use `Get-IAMAttachedUserPolicyList`.

**Tools for PowerShell**

**Example 1:** This command returns the names and ARNs of the managed policies for the IAM user named `Bob` in the AWS account. To see the list of inline policies that are embedded in the IAM user, use the `Get-IAMUserPolicyList` command.

```
Get-IAMAttachedUserPolicyList -UserName "Bob"
```

**Output:**

<table>
<thead>
<tr>
<th>PolicyArn</th>
<th>PolicyName</th>
</tr>
</thead>
<tbody>
<tr>
<td>arn:aws:iam::aws:policy/TesterPolicy</td>
<td>TesterPolicy</td>
</tr>
</tbody>
</table>
• For API details, see ListAttachedUserPolicies in AWS Tools for PowerShell Cmdlet Reference.

Get-IAMContextKeysForCustomPolicy

The following code example shows how to use Get-IAMContextKeysForCustomPolicy.

Tools for PowerShell

Example 1: This example fetches all the context keys present in the provided policy json. In order to provide multiple policies you can provide as comma separated list of values.

```powershell
$policy1 = ' {
"Version":"2012-10-17","Statement":
{"Effect":"Allow","Action":"dynamodb:*","Resource":"arn:aws:dynamodb:us-west-2:123456789012:table/","Condition":{"DateGreaterThan":
{"aws:CurrentTime":"2015-08-16T12:00:00Z"}}}}
$policy2 = ' {
"Version":"2012-10-17","Statement":
Get-IAMContextKeysForCustomPolicy -PolicyInputList $policy1,$policy2
```

• For API details, see GetContextKeysForCustomPolicy in AWS Tools for PowerShell Cmdlet Reference.

Get-IAMContextKeysForPrincipalPolicy

The following code example shows how to use Get-IAMContextKeysForPrincipalPolicy.

Tools for PowerShell

Example 1: This example fetches all the context keys present in the provided policy json and the policies attached to IAM entity(user/role etc.). For -PolicyInputList you can provide multiple values list as comma separated values.

```powershell
$policy1 = ' {
"Version":"2012-10-17","Statement":
{"Effect":"Allow","Action":"dynamodb:*","Resource":"arn:aws:dynamodb:us-west-2:123456789012:table/","Condition":{"DateGreaterThan":
{"aws:CurrentTime":"2015-08-16T12:00:00Z"}}}}
$policy2 = ' {
"Version":"2012-10-17","Statement":
```

IAM
Get-IAMContextKeysForPrincipalPolicy -PolicyInputList $policy1,$policy2 -PolicySourceArn arn:aws:iam::852640994763:user/TestUser


### Get-IAMCredentialReport

The following code example shows how to use Get-IAMCredentialReport.

#### Tools for PowerShell

**Example 1:** This example opens the returned report and outputs it to the pipeline as an array of text lines. The first line is the header with comma-separated column names. Each successive row is the detail row for one user, with each field separated by commas. Before you can view the report, you must generate it with the `Request-IAMCredentialReport` cmdlet. To retrieve the report as a single string, use `-Raw` instead of `-AsTextArray`. The alias `-SplitLines` is also accepted for the `-AsTextArray` switch. For the full list of columns in the output consult the service API reference. Note that if you do not use `-AsTextArray` or `-SplitLines`, then you must extract the text from the `.Content` property using the .NET StreamReader class.

```powershell
Request-IAMCredentialReport

<table>
<thead>
<tr>
<th>Description</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>No report exists. Starting a new report generation task</td>
<td>STARTED</td>
</tr>
</tbody>
</table>

Get-IAMCredentialReport -AsTextArray

**Output:**

```
user,arn,user_creation_time,password_enabled,password_last_used,password_last_changed,password_next_rotation,password_rotation_next_rotation_mfa_active,ac...
root_account,arn:aws:iam::123456789012:root,2014-10-15T16:31:25+00:00,not_supported,2015-04-20T17:41:10+00:00,not_supported,not_supported,true,false,N/A,false,N/A,false,N/A
Administrator,arn:aws:iam::123456789012:user/Administrator,2014-10-16T16:03:09+00:00,true,2015-04-20T15:18:32+00:00,2014-10-16T16:06:00+00:00```
```
For API details, see [GetCredentialReport](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-IAMEntitiesForPolicy**

The following code example shows how to use Get-IAMEntitiesForPolicy.

**Tools for PowerShell**

**Example 1:** This example returns a list of IAM groups, roles, and users who have the policy `arn:aws:iam::123456789012:policy/TestPolicy` attached.

```
Get-IAMEntitiesForPolicy -PolicyArn "arn:aws:iam::123456789012:policy/TestPolicy"
```

**Output:**

```
IsTruncated : False
Marker      :
PolicyGroups : {}
PolicyRoles  : {testRole}
PolicyUsers  : {Bob, Theresa}
```

For API details, see [ListEntitiesForPolicy](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-IAMGroup**

The following code example shows how to use Get-IAMGroup.

**Tools for PowerShell**

**Example 1:** This example returns details about the IAM group *Testers*, including a collection of all the IAM users that belong to the group.

```
$results = Get-IAMGroup -GroupName "Testers"
$results
```

**Output:**
For API details, see [GetGroup](https://docs.aws.amazon.com/PowerShell/latest/userguide/iam-getgroup.html) in AWS Tools for PowerShell Cmdlet Reference.

**Get-IAMGroupForUser**

The following code example shows how to use Get-IAMGroupForUser.
Example 1: This example returns the list of IAM groups that the IAM user David belongs to.

Get-IAMGroupForUser -UserName David

Output:

<table>
<thead>
<tr>
<th>Arn</th>
<th>arn:aws:iam::123456789012:group/Administrators</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateDate</td>
<td>10/20/2014 10:06:24 AM</td>
</tr>
<tr>
<td>GroupId</td>
<td>6WCH4TRY3KIH1EXAMPLE1</td>
</tr>
<tr>
<td>GroupName</td>
<td>Administrators</td>
</tr>
<tr>
<td>Path</td>
<td>/</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arn</th>
<th>arn:aws:iam::123456789012:group/Testers</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateDate</td>
<td>12/10/2014 3:39:11 PM</td>
</tr>
<tr>
<td>GroupId</td>
<td>RHNZZGQJ7QHMA1EXAMPLE2</td>
</tr>
<tr>
<td>GroupName</td>
<td>Testers</td>
</tr>
<tr>
<td>Path</td>
<td>/</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arn</th>
<th>arn:aws:iam::123456789012:group/Developers</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateDate</td>
<td>12/10/2014 3:38:55 PM</td>
</tr>
<tr>
<td>GroupId</td>
<td>ZU2EOWMK6WBZ0EXAMPLE3</td>
</tr>
<tr>
<td>GroupName</td>
<td>Developers</td>
</tr>
<tr>
<td>Path</td>
<td>/</td>
</tr>
</tbody>
</table>

For API details, see [ListGroupsForUser](#) in AWS Tools for PowerShell Cmdlet Reference.

Get-IAMGroupList

The following code example shows how to use Get-IAMGroupList.

Example 1: This example returns a collection of all the IAM groups defined in the current AWS account.

Get-IAMGroupList

Output:
Arn        : arn:aws:iam::123456789012:group/Administrators
CreateDate : 10/20/2014 10:06:24 AM
GroupId    : 6WCH4TRY3KIHIEXAMPLE1
GroupName  : Administrators
Path       : /

Arn        : arn:aws:iam::123456789012:group/Developers
CreateDate : 12/10/2014 3:38:55 PM
GroupId    : ZU2EOWMK6WBZOEXAMPLE2
GroupName  : Developers
Path       : /

Arn        : arn:aws:iam::123456789012:group/Testers
CreateDate : 12/10/2014 3:39:11 PM
GroupId    : RHNZZGQJ7QHMAEXAMPLE3
GroupName  : Testers
Path       : /

• For API details, see ListGroups in AWS Tools for PowerShell Cmdlet Reference.

Get-IAMGroupPolicy

The following code example shows how to use Get-IAMGroupPolicy.

Tools for PowerShell

Example 1: This example returns details about the embedded inline policy named PowerUserAccess-Testers for the group Testers. The PolicyDocument property is URL encoded. It is decoded in this example with the UrlDecode .NET method.

```powershell
$results = Get-IAMGroupPolicy -GroupName Testers -PolicyName PowerUserAccess-Testers
$results
```

Output:

<table>
<thead>
<tr>
<th>GroupName</th>
<th>PolicyDocument</th>
<th>PolicyName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testers</td>
<td>%7B%0A%20%20%22Version%22%3A%20%222012-10-17%22%2C%0A%20...</td>
<td>PowerUserAccess-Testers</td>
</tr>
</tbody>
</table>

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "NotAction": "iam:*",
            "Resource": "*"
        }
    ]
}

• For API details, see GetGroupPolicy in AWS Tools for PowerShell Cmdlet Reference.

Get-IAMGroupPolicyList

The following code example shows how to use Get-IAMGroupPolicyList.

Tools for PowerShell

Example 1: This example returns a list of the inline policies that are embedded in the group Testers. To get the managed policies that are attached to the group, use the command Get-IAMAttachedGroupPolicyList.

Get-IAMGroupPolicyList -GroupName Testers

Output:

Deny-Assume-S3-Role-In-Production
PowerUserAccess-Testers

• For API details, see ListGroupPolicies in AWS Tools for PowerShell Cmdlet Reference.

Get-IAMInstanceProfile

The following code example shows how to use Get-IAMInstanceProfile.

Tools for PowerShell

Example 1: This example returns details of the instance profile named ec2instanceaccessrole that is defined in the current AWS account.
Get-IAMInstanceProfile -InstanceProfileName ec2instancerole

Output:

| Arn | arn:aws:iam::123456789012:instance-profile/ec2instancerole |
| CreateDate | 2/17/2015 2:49:04 PM |
| InstanceProfileId | HH36PTZQJUR32EXAMPLE1 |
| InstanceProfileName | ec2instancerole |
| Path | / |
| Roles | {ec2instancerole} |

- For API details, see [GetInstanceProfile](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-IAMInstanceProfileForRole

The following code example shows how to use Get-IAMInstanceProfileForRole.

Tools for PowerShell

**Example 1**: This example returns details of the instance profile associated with the role ec2instancerole.

Get-IAMInstanceProfileForRole -RoleName ec2instancerole

Output:

| Arn | arn:aws:iam::123456789012:instance-profile/ec2instancerole |
| CreateDate | 2/17/2015 2:49:04 PM |
| InstanceProfileId | HH36PTZQJUR32EXAMPLE1 |
| InstanceProfileName | ec2instancerole |
| Path | / |
| Roles | {ec2instancerole} |

- For API details, see [ListInstanceProfilesForRole](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-IAMInstanceProfileList

The following code example shows how to use Get-IAMInstanceProfileList.
Tools for PowerShell

Example 1: This example returns a collection of the instance profiles defined in the current AWS account.

Get-IAMInstanceProfileList

Output:

<table>
<thead>
<tr>
<th>Arn</th>
<th>arn:aws:iam::123456789012:instance-profile/ec2instancerole</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateDate</td>
<td>2/17/2015 2:49:04 PM</td>
</tr>
<tr>
<td>InstanceProfileId</td>
<td>HH36PTQJUR32EXAMPLE1</td>
</tr>
<tr>
<td>InstanceProfileName</td>
<td>ec2instancerole</td>
</tr>
<tr>
<td>Path</td>
<td>/</td>
</tr>
<tr>
<td>Roles</td>
<td>{ec2instancerole}</td>
</tr>
</tbody>
</table>

- For API details, see ListInstanceProfiles in AWS Tools for PowerShell Cmdlet Reference.

Get-IAMLoginProfile

The following code example shows how to use Get-IAMLoginProfile.

Tools for PowerShell

Example 1: This example returns the password creation date and whether a password reset is required for the IAM user David.

Get-IAMLoginProfile -UserName David

Output:

<table>
<thead>
<tr>
<th>CreateDate</th>
<th>PasswordResetRequired</th>
<th>UserName</th>
</tr>
</thead>
<tbody>
<tr>
<td>12/10/2014</td>
<td>False</td>
<td>David</td>
</tr>
</tbody>
</table>

- For API details, see GetLoginProfile in AWS Tools for PowerShell Cmdlet Reference.

Get-IAMMFADevice

The following code example shows how to use Get-IAMMFADevice.
Tools for PowerShell

Example 1: This example returns details about the MFA device assigned to the IAM user David. In this example you can tell that it is a virtual device because the SerialNumber is an ARN instead of a physical device's actual serial number.

Get-IAMMFADevice -UserName David

Output:

<table>
<thead>
<tr>
<th>EnableDate</th>
<th>SerialNumber</th>
<th>UserName</th>
</tr>
</thead>
</table>

For API details, see ListMfaDevices in AWS Tools for PowerShell Cmdlet Reference.

Get-IAMOpenIDConnectProvider

The following code example shows how to use Get-IAMOpenIDConnectProvider.

Tools for PowerShell

Example 1: This example returns details about the OpenID Connect provider whose ARN is arn:aws:iam::123456789012:oidc-provider/accounts.google.com. The ClientIDList property is a collection that contains all the Client IDs defined for this provider.

Get-IAMOpenIDConnectProvider -OpenIDConnectProviderArn
arn:aws:iam::123456789012:oidc-provider/oidc.example.com

Output:

<table>
<thead>
<tr>
<th>ClientIDList</th>
<th>CreateDate</th>
<th>ThumbprintList</th>
</tr>
</thead>
<tbody>
<tr>
<td>{MyOIDCApp}</td>
<td>2/3/2015 3:00:30 PM</td>
<td>{12345abcdefgijklmnopqrstuvwxyz}</td>
</tr>
</tbody>
</table>
For API details, see `GetOpenIdConnectProvider` in `AWS Tools for PowerShell Cmdlet Reference`.

**Get-IAMOpenIDConnectProviderList**

The following code example shows how to use Get-IAMOpenIDConnectProviderList.

**Tools for PowerShell**

**Example 1:** This example returns a list of ARNS of all the OpenID Connect providers that are defined in the current AWS account.

```
Get-IAMOpenIDConnectProviderList
```

**Output:**

```
Arn
---
arn:aws:iam::123456789012:oidc-provider/server.example.com
arn:aws:iam::123456789012:oidc-provider/another.provider.com
```

For API details, see `ListOpenIdConnectProviders` in `AWS Tools for PowerShell Cmdlet Reference`.

**Get-IAMPolicy**

The following code example shows how to use Get-IAMPolicy.

**Tools for PowerShell**

**Example 1:** This example returns details about the managed policy whose ARN is `arn:aws:iam::123456789012:policy/MySamplePolicy`.

```
Get-IAMPolicy -PolicyArn arn:aws:iam::123456789012:policy/MySamplePolicy
```

**Output:**

```
Arn              : arn:aws:iam::aws:policy/MySamplePolicy
AttachmentCount  : 0
CreateDate       : 2/6/2015 10:40:08 AM
```
For API details, see [GetPolicy](#) in AWS Tools for PowerShell Cmdlet Reference.

---

### Get-IAMPolicyList

The following code example shows how to use Get-IAMPolicyList.

**Tools for PowerShell**

*Example 1: This example returns a collection of the first three managed policies available in the current AWS account. Because `-scope` is not specified, it defaults to `all` and includes both AWS managed and customer managed policies.*

```
Get-IAMPolicyList -MaxItem 3
```

**Output:**

```
Arn              : arn:aws:iam::aws:policy/AWSDirectConnectReadOnlyAccess
AttachmentCount  : 0
CreateDate       : 2/6/2015 10:40:08 AM
DefaultVersionId : v1
Description      :
IsAttachable     : True
Path             : /
PolicyId         : Z27SI6FQMGNQ2EXAMPLE1
PolicyName       : AWSDirectConnectReadOnlyAccess
UpdateDate       : 2/6/2015 10:40:08 AM

Arn              : arn:aws:iam::aws:policy/AmazonGlacierReadOnlyAccess
AttachmentCount  : 0
CreateDate       : 2/6/2015 10:40:27 AM
DefaultVersionId : v1
Description      :
IsAttachable     : True
```
Example 2: This example returns a collection of the first two customer managed policies available in current AWS account. It uses -Scope local to limit the output to only customer managed policies.

```
Get-IAMPolicyList -Scope local -MaxItem 2
```

Output:

```
Arn              : arn:aws:iam::123456789012:policy/MyLocalPolicy
AttachmentCount  : 0
CreateDate       : 2/12/2015 9:39:09 AM
DefaultVersionId : v2
Description      :
IsAttachable     : True
Path             : /
PolicyId         : SQVCBL4VAQUCEXTERNALEXAMPLE4
PolicyName       : MyLocalPolicy
UpdateDate       : 2/12/2015 9:39:53 AM

Arn              : arn:aws:iam::123456789012:policy/policyforec2instancetype
AttachmentCount  : 1
CreateDate       : 2/17/2015 2:51:38 PM
DefaultVersionId : v11
Description      :
IsAttachable     : True
Path             : /
PolicyId         : X5JPBLJH2Z2S0EXAMPLE5
PolicyName       : policyforec2instancerole
UpdateDate       : 2/18/2015 8:52:31 AM

- For API details, see ListPolicies in AWS Tools for PowerShell Cmdlet Reference.

Get-IAMPolicyVersion

The following code example shows how to use Get-IAMPolicyVersion.

Tools for PowerShell

Example 1: This example returns the policy document for the v2 version of the policy whose ARN is arn:aws:iam::123456789012:policy/MyManagedPolicy. The policy document in the Document property is URL encoded and is decoded in this example with the UrlDecode .NET method.

```powershell
$results

Output:

<table>
<thead>
<tr>
<th>CreateDate</th>
<th>Document</th>
</tr>
</thead>
<tbody>
<tr>
<td>2/12/2015 9:39:53 AM</td>
<td>%7B%0A%20%20%22Version%22%3A%20%222012-10-17...</td>
</tr>
<tr>
<td>IsDefaultVersion</td>
<td>VersionId</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

$policy
{
    "Version": "2012-10-17",
    "Statement":
    {
        "Effect": "Allow",
        "Action": "*",
        "Resource": "*
    }
}
For API details, see [GetPolicyVersion](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-IAMPolicyVersionList

The following code example shows how to use Get-IAMPolicyVersionList.

**Tools for PowerShell**

**Example 1:** This example returns the list of available versions of the policy whose ARN is `arn:aws:iam::123456789012:policy/MyManagedPolicy`. To get the policy document for a specific version, use the `Get-IAMPolicyVersion` command and specify the `VersionId` of the one you want.

```
```

**Output:**

<table>
<thead>
<tr>
<th>CreateDate</th>
<th>Document</th>
<th>IsDefaultVersion</th>
</tr>
</thead>
<tbody>
<tr>
<td>2/12/2015 9:39:53 AM</td>
<td>v2</td>
<td>True</td>
</tr>
<tr>
<td>2/12/2015 9:39:09 AM</td>
<td>v1</td>
<td>False</td>
</tr>
</tbody>
</table>

For API details, see [ListPolicyVersions](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-IAMRole

The following code example shows how to use Get-IAMRole.

**Tools for PowerShell**

**Example 1:** This example returns the details of the `lambda_exec_role`. It includes the trust policy document that specifies who can assume this role. The policy document is URL encoded and can be decoded using the .NET `UrlDecode` method. In this example, the
original policy had all white space removed before it was uploaded to the policy. To see the permissions policy documents that determine what someone who assumes the role can do, use the `Get-IAMRolePolicy` for inline policies, and `Get-IAMPolicyVersion` for attached managed policies.

```powershell
$results = Get-IAMRole -RoleName lambda_exec_role
$results | Format-List
```

**Output:**

```
Arn                      : arn:aws:iam::123456789012:role/lambda_exec_role
AssumeRolePolicyDocument : %7B%22Version%22%3A%222012-10-17%22%2C%22Statement%22%3A%223A%22%2C%22Sid%22%22%3A%22%22%2C%22Effect%22%3A%22Allow%22%2C%22Principal%22%3A%7B%22Service%22%3A%22lambda.amazonaws.com%22%7D%22%2C%22Action%22%3A%222012-10-17%22%7D
CreateDate               : 4/2/2015 9:16:11 AM
Path                     : /
RoleId                   : 2YBIKAIBHNKB4EXAMPLE1
RoleName                 : lambda_exec_role
```

```
$policy
```

**Output:**

```
```

- For API details, see [GetRole](https://aws.amazon.com) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-IAMRoleList

The following code example shows how to use `Get-IAMRoleList`.

### Tools for PowerShell

**Example 1:** This example retrieves a list of all of the IAM roles in the AWS account.
Example 2: This example code snippet retrieves a list of IAM roles in the AWS account and displays them three at a time, and waits for you to press Enter between each group. It passes the Marker value from the previous call to specify where the next group should begin.

```powershell
$nextMarker = $null
Do
{
    $results = Get-IAMRoleList -MaxItem 3 -Marker $nextMarker
    $nextMarker = $AWSHistory.LastServiceResponse.Marker
    $results
    Read-Host
} while ($nextMarker -ne $null)
```

- For API details, see ListRoles in AWS Tools for PowerShell Cmdlet Reference.

Get-IAMRolePolicy

The following code example shows how to use Get-IAMRolePolicy.

Tools for PowerShell

Example 1: This example returns the permissions policy document for the policy named oneClick_lambda_exec_role_policy that is embedded in the IAM role lambda_exec_role. The resulting policy document is URL encoded. It is decoded in this example with the UrlDecode .NET method.

```powershell
$results = Get-IAMRolePolicy -RoleName lambda_exec_role -PolicyName oneClick_lambda_exec_role_policy
$results
```

Output:

<table>
<thead>
<tr>
<th>PolicyDocument</th>
<th>PolicyName</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserName</td>
<td>PolicyName</td>
</tr>
<tr>
<td>---------------</td>
<td>------------</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>IAM</td>
<td></td>
</tr>
</tbody>
</table>
For API details, see [GetRolePolicy](#) in AWS Tools for PowerShell Cmdlet Reference.

### Get-IAMRolePolicyList

The following code example shows how to use Get-IAMRolePolicyList.

#### Tools for PowerShell

**Example 1:** This example returns the list of names of inline policies that are embedded in the IAM role `lambda_exec_role`. To see the details of an inline policy, use the command `Get-IAMRolePolicy`.
Get-IAMRolePolicyList -RoleName lambda_exec_role

Output:

oneClick_lambda_exec_role_policy

- For API details, see ListRolePolicies in AWS Tools for PowerShell Cmdlet Reference.

Get-IAMRoleTagList

The following code example shows how to use Get-IAMRoleTagList.

Tools for PowerShell

Example 1: This example fetches the tag associated with the role..

Get-IAMRoleTagList -RoleName MyRoleName

- For API details, see ListRoleTags in AWS Tools for PowerShell Cmdlet Reference.

Get-IAMSAMLProvider

The following code example shows how to use Get-IAMSAMLProvider.

Tools for PowerShell

Example 1: This example retrieves the details about the SAML 2.0 provider whose ARM is arn:aws:iam::123456789012:saml-provider/SAMLADFS. The response includes the metadata document that you got from the identity provider to create the AWS SAML provider entity as well as the creation and expiration dates.

Get-IAMSAMLProvider -SAMLProviderArn arn:aws:iam::123456789012:saml-provider/SAMLADFS

Output:

<table>
<thead>
<tr>
<th>CreateDate</th>
<th>SAMLMetadataDocument</th>
</tr>
</thead>
<tbody>
<tr>
<td>ValidUntil</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------</td>
</tr>
<tr>
<td></td>
<td>----------------------</td>
</tr>
</tbody>
</table>
For API details, see `GetSamlProvider` in AWS Tools for PowerShell Cmdlet Reference.

Get-IAMSAMLProviderList

The following code example shows how to use Get-IAMSAMLProviderList.

Tools for PowerShell

Example 1: This example retrieves the list of SAML 2.0 providers created in the current AWS account. It returns the ARN, creation date, and expiration date for each SAML provider.

```
Get-IAMSAMLProviderList
```

Output:

<table>
<thead>
<tr>
<th>Arn</th>
<th>CreateDate</th>
</tr>
</thead>
</table>

For API details, see `ListSAMLProviders` in AWS Tools for PowerShell Cmdlet Reference.

Get-IAMServerCertificate

The following code example shows how to use Get-IAMServerCertificate.

Tools for PowerShell

Example 1: This example retrieves details about the server certificate named MyServerCertificate. You can find the certificate details in the CertificateBody and ServerCertificateMetadata properties.

```
$result = Get-IAMServerCertificate -ServerCertificateName MyServerCertificate
$result | format-list
```
AWS Tools for PowerShell User Guide

Output:

CertificateBody           : -----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uX0jANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBHMxCzAJ
BgNVBAgTAldBMRw0wDQQYDVR0TAQHqgSGZGIW5vb25lQGFtYXpvbi5jb20wgZ8w
+a4GmWIWJ
21uUSfwnEvxSVelC2XADZ4nB+BLyGIYk600pipsZ3G93vUEI03IyNoH/
f0wYK8m9T
rDHudUZgz3qX4waLGl543q7Wgc/
MbQITx0USQv7c7ugFfdzQGbzZswY6786m86gpE
Ibb30RhjNzcvQAaRbd1QWIMm2nrAgMBAAEwDQYJkgcWbE4QAQgDgY0AMIGJaooGBAMaK0dn
+auNkyExzyLwax1Aoo7TFHidbtS4J5iNmZgXL0Fkb
FFBjvSfpJlJ0zbhNYS5f6GuoEDmFj10ZxBHJJyyp3780D8uTs7fLvjx79LjSTb
NYiytVbZPQUQ5Yaxu2jXnimvw5rrszlaEXAMPLE=
-----END CERTIFICATE-----

CertificateChain          :
ServerCertificateMetadata :

$result.ServerCertificateMetadata

Output:

MyServerCertificate   :
Expiration            : 1/14/2018 9:52:36 AM
Path                  : /Org1/Org2/
ServerCertificateId   : ASCAJIFEXAMPLE17HQZYW
ServerCertificateName : MyServerCertificate
UploadDate            : 4/21/2015 11:14:16 AM

For API details, see [GetServerCertificate](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-IAMServerCertificateList**

The following code example shows how to use `Get-IAMServerCertificateList`.

**Tools for PowerShell**

**Example 1:** This example retrieves the list of server certificates that have been uploaded to the current AWS account.

```powershell
Get-IAMServerCertificateList
```

**Output:**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arn</td>
<td>arn:aws:iam::123456789012:server-certificate/Org1/Org2/</td>
</tr>
<tr>
<td>MyServerCertificate</td>
<td>MyServerCertificate</td>
</tr>
<tr>
<td>Expiration</td>
<td>1/14/2018 9:52:36 AM</td>
</tr>
<tr>
<td>Path</td>
<td>/Org1/Org2/</td>
</tr>
<tr>
<td>ServerCertificateId</td>
<td>ASCAJIFEXAMPLE17HQZYW</td>
</tr>
<tr>
<td>ServerCertificateName</td>
<td>MyServerCertificate</td>
</tr>
<tr>
<td>UploadDate</td>
<td>4/21/2015 11:14:16 AM</td>
</tr>
</tbody>
</table>

For API details, see [ListServerCertificates](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-IAMServiceLastAccessedDetail**

The following code example shows how to use `Get-IAMServiceLastAccessedDetail`.

**Tools for PowerShell**

**Example 1:** This example provides details of the service last accessed by the IAM entity (user, group, role or policy) associated in Request call.

```powershell
Request-IAMServiceLastAccessedDetail -Arn arn:aws:iam::123456789012:user/TestUser
```
AWS Tools for PowerShell

Get-IAMServiceLastAccessedDetail -JobId f0b7a819-eab0-929b-dc26-ca598911cb9f

- For API details, see [GetServiceLastAccessedDetails](#) in AWS Tools for PowerShell Cmdlet Reference.

**Get-IAMServiceLastAccessedDetailWithEntity**

The following code example shows how to use Get-IAMServiceLastAccessedDetailWithEntity.

**Tools for PowerShell**

**Example 1:** This example provides the last accessed timestamp for the service in the request by that respective IAM entity.

```
$results = Get-IAMServiceLastAccessedDetailWithEntity -JobId f0b7a819-eab0-929b-dc26-ca598911cb9f -ServiceNamespace ec2
$results
```

**Output:**

```
EntityDetailsList : {Amazon.IdentityManagement.Model.EntityDetails}
Error             : 
IsTruncated       : False
JobCompletionDate : 12/29/19 11:19:31 AM
JobCreationDate   : 12/29/19 11:19:31 AM
JobStatus         : COMPLETED
Marker            :

$results.EntityDetailsList
```

**Output:**

```
EntityInfo                                 LastAuthenticated
----------                                 -----------------
IAM                                    577
```

IAM
$results.EntityInfo

Output:

Arn : arn:aws:iam::123456789012:user/TestUser
Id   : AIDA4NBK5CF5TZHU1234
Name : TestUser
Path : /
Type : USER

- For API details, see [GetServiceLastAccessedDetailsWithEntities](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-IAMSigningCertificate

The following code example shows how to use `Get-IAMSigningCertificate`.

#### Tools for PowerShell

**Example 1: This example retrieves details about the signing certificate that is associated with the user named Bob.**

```powershell
Get-IAMSigningCertificate -UserName Bob
```

**Output:**

CertificateBody : -----BEGIN CERTIFICATE-----
MIICiTCCAfICCQDm7oRw0uXOjANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMCVVMx
CzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6b24x
FDASBnNVBAAsTC01BTSBDb25zb2x1MRIwEAYDVQQDEwUZXN0Q2lsYWMxHxAd
Bqkqhk1G9w0BCQEWE5v8vb251QGFrYXpvbi5jb20wHhcNMTEwNDI1MjA0NTIz
MTIwNDI0MjA0NTIwMjcyMjAxMTIwNDI0MjA0NTIz
YXpvbi5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMaK0dn+a4GmWlWJ
578
For API details, see [ListSigningCertificates](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-IAMUser

The following code example shows how to use Get-IAMUser.

#### Tools for PowerShell

**Example 1:** This example retrieves details about the user named David.

```powershell
Get-IAMUser -UserName David
```

**Output:**

- **Arn**: arn:aws:iam::123456789012:user/David
- **CreateDate**: 12/10/2014 3:39:27 PM
- **PasswordLastUsed**: 3/19/2015 8:44:04 AM
- **Path**: /
- **UserId**: Y4FKWQCXTA52QEXAMPLE1
- **UserName**: David

**Example 2:** This example retrieves details about the currently signed-in IAM user.

```powershell
Get-IAMUser
```

**Output:**

- **Arn**: arn:aws:iam::123456789012:user/Bob
- **CreateDate**: 10/16/2014 9:03:09 AM
- **PasswordLastUsed**: 3/4/2015 12:12:33 PM
- **Path**: /
- **UserId**: 7K3GJEANSKZF2EXAMPLE2
- **UserName**: Bob
Get-IAMUserList

The following code example shows how to use Get-IAMUserList.

Tools for PowerShell

Example 1: This example retrieves a collection of users in the current AWS account.

Get-IAMUserList

Output:

<table>
<thead>
<tr>
<th>Arn</th>
<th>arn:aws:iam::123456789012:user/Administrator</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateDate</td>
<td>10/16/2014 9:03:09 AM</td>
</tr>
<tr>
<td>PasswordLastUsed</td>
<td>3/4/2015 12:12:33 PM</td>
</tr>
<tr>
<td>Path</td>
<td>/</td>
</tr>
<tr>
<td>UserId</td>
<td>7K3GJEANSKZF2EXAMPLE1</td>
</tr>
<tr>
<td>UserName</td>
<td>Administrator</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arn</th>
<th>arn:aws:iam::123456789012:user/Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateDate</td>
<td>4/6/2015 12:54:42 PM</td>
</tr>
<tr>
<td>PasswordLastUsed</td>
<td>1/1/0001 12:00:00 AM</td>
</tr>
<tr>
<td>Path</td>
<td>/</td>
</tr>
<tr>
<td>UserId</td>
<td>L3EWNONDOM3YUEXAMPLE2</td>
</tr>
<tr>
<td>UserName</td>
<td>bab</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arn</th>
<th>arn:aws:iam::123456789012:user/David</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateDate</td>
<td>12/10/2014 3:39:27 PM</td>
</tr>
<tr>
<td>PasswordLastUsed</td>
<td>3/19/2015 8:44:04 AM</td>
</tr>
<tr>
<td>Path</td>
<td>/</td>
</tr>
<tr>
<td>UserId</td>
<td>Y4FKWQCTXA52QEXAMPLE3</td>
</tr>
<tr>
<td>UserName</td>
<td>David</td>
</tr>
</tbody>
</table>

Get-IAMUserPolicy

The following code example shows how to use Get-IAMUserPolicy.

• For API details, see [ListUsers](#) in AWS Tools for PowerShell Cmdlet Reference.
Example 1: This example retrieves the details of the inline policy named Davids_IAM_Admin_Policy that is embedded in the IAM user named David. The policy document is URL encoded.

```powershell
$results = Get-IAMUserPolicy -PolicyName Davids_IAM_Admin_Policy -UserName David
$results
```

**Output:**

<table>
<thead>
<tr>
<th>PolicyDocument</th>
<th>PolicyName</th>
</tr>
</thead>
<tbody>
<tr>
<td>---------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>UserName</td>
<td>Davids_IAM_Admin_Policy</td>
</tr>
<tr>
<td>David</td>
<td></td>
</tr>
</tbody>
</table>

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:*"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

- For API details, see [GetUserPolicy](#) in AWS Tools for PowerShell Cmdlet Reference.

**Get-IAMUserPolicyList**

The following code example shows how to use Get-IAMUserPolicyList.
Example 1: This example retrieves the list of names of the inline policies that are embedded in the IAM user named David.

```powershell
Get-IAMUserPolicyList -UserName David
```

Output:

```
Davids_IAM_Admin_Policy
```

- For API details, see [ListUserPolicies](#) in AWS Tools for PowerShell Cmdlet Reference.

Get-IAMUserTagList

The following code example shows how to use Get-IAMUserTagList.

Example 1: This example fetches the tag associated with the user.

```powershell
Get-IAMUserTagList -UserName joe
```

- For API details, see [ListUserTags](#) in AWS Tools for PowerShell Cmdlet Reference.

Get-IAMVirtualMFADevice

The following code example shows how to use Get-IAMVirtualMFADevice.

Example 1: This example retrieves a collection of the virtual MFA devices that are assigned to users in the AWS account. The User property of each is an object with details of the IAM user to which the device is assigned.

```powershell
Get-IAMVirtualMFADevice -AssignmentStatus Assigned
```

Output:
New-IAMAccessKey

The following code example shows how to use New-IAMAccessKey.

Tools for PowerShell

Example 1: This example creates a new access key and secret access key pair and assigns it to the user David. Ensure that you save the AccessKeyId and SecretAccessKey values to a file because this is the only time you can obtain the SecretAccessKey. You cannot retrieve it later. If you lose the secret key, you must create a new access key pair.

New-IAMAccessKey -UserName David

Output:

<table>
<thead>
<tr>
<th>AccessKeyId</th>
<th>AKIAIOSFODNN7EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateDate</td>
<td>4/13/2015 1:00:42 PM</td>
</tr>
<tr>
<td>SecretAccessKey</td>
<td>wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY</td>
</tr>
<tr>
<td>Status</td>
<td>Active</td>
</tr>
<tr>
<td>UserName</td>
<td>David</td>
</tr>
</tbody>
</table>

New-IAMAccountAlias

The following code example shows how to use New-IAMAccountAlias.
Example 1: This example changes the account alias for your AWS account to mycompanyaws. The address of the user logon page changes to https://mycompanyaws.signin.aws.amazon.com/console. The original URL using your account ID number instead of the alias (https://<accountidnumber>.signin.aws.amazon.com/console) continues to work. However, any previously defined alias-based URLs stop working.

```
New-IAMAccountAlias -AccountAlias mycompanyaws
```

For API details, see CreateAccountAlias in AWS Tools for PowerShell Cmdlet Reference.

New-IAMGroup

The following code example shows how to use New-IAMGroup.

Example 1: This example creates a new IAM group named Developers.

```
New-IAMGroup -GroupName Developers
```

Output:

```
Arn        : arn:aws:iam::123456789012:group/Developers
CreateDate : 4/14/2015 11:21:31 AM
GroupId    : QNEJ5PM4NFSQCEXAMPLE1
GroupName  : Developers
Path       : /
```

For API details, see CreateGroup in AWS Tools for PowerShell Cmdlet Reference.

New-IAMInstanceProfile

The following code example shows how to use New-IAMInstanceProfile.

Example 1: This example creates a new IAM instance profile named ProfileForDevEC2Instance. You must separately run the Add-
IAMRoleToInstanceProfile command to associate the instance profile with an existing IAM role that provides permissions to the instance. Finally, attach the instance profile to an EC2 instance when you launch it. To do that, use the New-EC2Instance cmdlet with either the InstanceProfile_Arn or InstanceProfile_Name parameter.

```powershell
New-IAMInstanceProfile -InstanceProfileName ProfileForDevEC2Instance
```

Output:

```
Arn                 : arn:aws:iam::123456789012:instance-profile/ProfileForDevEC2Instance
CreateDate          : 4/14/2015 11:31:39 AM
InstanceProfileId   : DYMFXL556EY46EXAMPLE1
InstanceProfileName : ProfileForDevEC2Instance
Path                : /
Roles               : {}
```

- For API details, see CreateInstanceProfile in AWS Tools for PowerShell Cmdlet Reference.

New-IAMLoginProfile

The following code example shows how to use New-IAMLoginProfile.

Tools for PowerShell

Example 1: This example creates a (temporary) password for the IAM user named Bob, and sets the flag that requires the user to change the password the next time Bob signs in.

```powershell
New-IAMLoginProfile -UserName Bob -Password P@ssw0rd -PasswordResetRequired $true
```

Output:

```
CreateDate                    PasswordResetRequired                UserName
----------                    ---------------------                --------
4/14/2015 12:26:30 PM         True                                 Bob
```

- For API details, see CreateLoginProfile in AWS Tools for PowerShell Cmdlet Reference.
**New-IAMOpenIDConnectProvider**

The following code example shows how to use `New-IAMOpenIDConnectProvider`.

**Tools for PowerShell**

**Example 1:** This example creates an IAM OIDC provider associated with the OIDC compatible provider service found at the URL `https://example.oidcprovider.com` and the client ID `my-testapp-1`. The OIDC provider supplies the thumbprint. To authenticate the thumbprint, follow the steps at [http://docs.aws.amazon.com/IAM/latest/UserGuide/identity-providers-oidc-obtain-thumbprint.html](http://docs.aws.amazon.com/IAM/latest/UserGuide/identity-providers-oidc-obtain-thumbprint.html).

```powershell
New-IAMOpenIDConnectProvider -Url https://example.oidcprovider.com -ClientIDList my-testapp-1 -ThumbprintList 990F419EXAMPLEECF12DDEDA5EXAMPLE52F20D9E
```

**Output:**

```
arn:aws:iam::123456789012:oidc-provider/example.oidcprovider.com
```

- For API details, see [CreateOpenIdConnectProvider](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**New-IAMPolicy**

The following code example shows how to use `New-IAMPolicy`.

**Tools for PowerShell**

**Example 1:** This example creates a new IAM policy in the current AWS account named `MySamplePolicy`. The file `MySamplePolicy.json` provides the policy content. Note that you must use the `-Raw` switch parameter to successfully process the JSON policy file.

```powershell
New-IAMPolicy -PolicyName MySamplePolicy -PolicyDocument (Get-Content -Raw MySamplePolicy.json)
```

**Output:**

```
Arn : arn:aws:iam::123456789012:policy/MySamplePolicy
```
For API details, see [CreatePolicy](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**New-IAMPolicyVersion**

The following code example shows how to use `New-IAMPolicyVersion`.

**Tools for PowerShell**

Example 1: This example creates a new "v2" version of the IAM policy whose ARN is `arn:aws:iam::123456789012:policy/MyPolicy` and makes it the default version. The `NewPolicyVersion.json` file provides the policy content. Note that you must use the `-Raw` switch parameter to successfully process the JSON policy file.

```powershell
```

**Output:**

<table>
<thead>
<tr>
<th>CreateDate</th>
<th>Document</th>
<th>IsDefaultVersion</th>
</tr>
</thead>
<tbody>
<tr>
<td>4/15/2015 10:54:54 AM</td>
<td>v2</td>
<td>True</td>
</tr>
</tbody>
</table>

For API details, see [CreatePolicyVersion](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**New-IAMRole**

The following code example shows how to use `New-IAMRole`.
Example 1: This example creates a new role named MyNewRole and attaches to it the policy found in the file NewRoleTrustPolicy.json. Note that you must use the -Raw switch parameter to successfully process the JSON policy file. The policy document displayed in the output is URL encoded. It is decoded in this example with the UrlDecode .NET method.

```
$results = New-IAMRole -AssumeRolePolicyDocument (Get-Content -raw NewRoleTrustPolicy.json) -RoleName MyNewRole
$results
```

**Output:**

```
Arn                      : arn:aws:iam::123456789012:role/MyNewRole
AssumeRolePolicyDocument : %7B%0D%0A%20%20%22Version%22%3A%222012-10-17%22%0D%0A%20%20%22Statement%22%0D%0A%20%20%20%20%7B%0D%0A%20%20%20%20%20%20%22Sid%22%3A%22%22%0D%0A%20%20%20%20%20%20%22Effect%22%3A%22Allow%22%0D%0A%20%20%20%20%20%20%22Principal%22%3A%22arn%3Aaws%3Aiam%3A%3A123456789012%3ADavid%22%0D%0A%20%20%20%22Action%22%3A%22sts%3AAssumeRole%22%0D%0A%20%20%22Principal%22%3A%22arn%3Aaws%3Aiam%3A%3A123456789012%3ADavid%22%0D%0A%20%20%22Action%22%3A%22sts%3AAssumeRole%22%0D%0A%20%20%22Principal%22%3A%22arn%3Aaws%3Aiam%3A%3A123456789012%3ADavid%22%0D%0A%20%20%22Action%22%3A%22sts%3AAssumeRole%22%0D%0A
CreateDate               : 4/15/2015 11:04:23 AM
Path                     : /
RoleId                   : V5PAJI2KPN4EAEXAMPLE1
RoleName                 : MyNewRole

{
    "Version": "2012-10-17",
    "Statement": [
    {
        "Sid": "",
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::123456789012:David"
        },
```
AWS Tools for PowerShell

"Action": "sts:AssumeRole"
}
]
}

- For API details, see CreateRole in AWS Tools for PowerShell Cmdlet Reference.

New-IAMSAMLProvider

The following code example shows how to use New-IAMSAMLProvider.

Tools for PowerShell

Example 1: This example creates a new SAML provider entity in IAM. It is named MySAMLProvider and is described by the SAML metadata document found in the file SAMLMetaData.xml, which was separately downloaded from the SAML service provider's web site.

```
New-IAMSAMLProvider -Name MySAMLProvider -SAMLMetadataDocument (Get-Content -Raw SAMLMetaData.xml)
```

Output:

```
arn:aws:iam::123456789012:saml-provider/MySAMLProvider
```

- For API details, see CreateSAMLProvider in AWS Tools for PowerShell Cmdlet Reference.

New-IAMServiceLinkedRole

The following code example shows how to use New-IAMServiceLinkedRole.

Tools for PowerShell

Example 1: This example creates a servicelinked role for autoscaling service.

```
New-IAMServiceLinkedRole -AWSServiceName autoscaling.amazonaws.com -CustomSuffix RoleNameEndsWithThis -Description "My service-linked role to support autoscaling"
```

- For API details, see CreateServiceLinkedRole in AWS Tools for PowerShell Cmdlet Reference.
**New-IAMUser**

The following code example shows how to use New-IAMUser.

**Tools for PowerShell**

**Example 1:** This example creates an IAM user named Bob. If Bob needs to sign in to the AWS console, then you must separately run the command New-IAMLoginProfile to create a sign-in profile with a password. If Bob needs to run AWS PowerShell or cross-platform CLI commands or make AWS API calls, then you must separately run the New-IAMAccessKey command to create access keys.

```
New-IAMUser -UserName Bob
```

**Output:**

```
Arn              : arn:aws:iam::123456789012:user/Bob
CreateDate       : 4/22/2015 12:02:11 PM
PasswordLastUsed : 1/1/0001 12:00:00 AM
Path             : /
UserId           : AIDAJWGEFDMEMEXAMPLE1
UserName         : Bob
```

- For API details, see [CreateUser](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**New-IAMVirtualMFADevice**

The following code example shows how to use New-IAMVirtualMFADevice.

**Tools for PowerShell**

**Example 1:** This example creates a new virtual MFA device. Lines 2 and 3 extract the Base32StringSeed value that the virtual MFA software program needs to create an account (as an alternative to the QR code). After you configure the program with the value, get two sequential authentication codes from the program. Finally, use the last command to link the virtual MFA device to the IAM user Bob and synchronize the account with the two authentication codes.

```
$Device = New-IAMVirtualMFADevice -VirtualMFADeviceName BobsMFADevice
$SR = New-Object System.IO.StreamReader($Device.Base32StringSeed)
$base32stringseed = $SR.ReadToEnd()
```
$base32stringseed
CZWZMCQNW4DEXAMPLE3V0UGXJFZYSUW7EXAMPLECR4NJFD65GX2SLUDW2EXAMPLE

Output:

-- Pause here to enter base-32 string seed code into virtual MFA program to register account. --

Enable-IAMMFADevice -SerialNumber $Device.SerialNumber -UserName Bob -AuthenticationCode1 123456 -AuthenticationCode2 789012

Example 2: This example creates a new virtual MFA device. Lines 2 and 3 extract the QRCodePNG value and write it to a file. This image can be scanned by the virtual MFA software program to create an account (as an alternative to manually entering the Base32StringSeed value). After you create the account in your virtual MFA program, get two sequential authentication codes and enter them in the last commands to link the virtual MFA device to the IAM user Bob and synchronize the account.

$Device = New-IAMVirtualMFADevice -VirtualMFADeviceName BobsMFADevice
$BR = New-Object System.IO.BinaryReader($Device.QRCodePNG)
$BR.ReadBytes($BR.BaseStream.Length) | Set-Content -Encoding Byte -Path QRCode.png

Output:

-- Pause here to scan PNG with virtual MFA program to register account. --

Enable-IAMMFADevice -SerialNumber $Device.SerialNumber -UserName Bob -AuthenticationCode1 123456 -AuthenticationCode2 789012

• For API details, see CreateVirtualMfaDevice in AWS Tools for PowerShell Cmdlet Reference.

Publish-IAMServerCertificate

The following code example shows how to use Publish-IAMServerCertificate.

Tools for PowerShell

Example 1: This example uploads a new server certificate to the IAM account. The files containing the certificate body, the private key, and (optionally) the certificate chain must all be PEM encoded. Note that the parameters require the actual content of the files rather
than the file names. You must use the -Raw switch parameter to successfully process the file contents.

```
Publish-IAMServerCertificate -ServerCertificateName MyTestCert -CertificateBody (Get-Content -Raw server.crt) -PrivateKey (Get-Content -Raw server.key)
```

**Output:**

<table>
<thead>
<tr>
<th>Arn</th>
<th>arn:aws:iam::123456789012:server-certificate/MyTestCert</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expiration</td>
<td>1/14/2018 9:52:36 AM</td>
</tr>
<tr>
<td>Path</td>
<td>/</td>
</tr>
<tr>
<td>ServerCertificateId</td>
<td>ASCAJIEXAMPLE7J7HQZYW</td>
</tr>
<tr>
<td>ServerCertificateName</td>
<td>MyTestCert</td>
</tr>
<tr>
<td>UploadDate</td>
<td>4/21/2015 11:14:16 AM</td>
</tr>
</tbody>
</table>

- For API details, see [UploadServerCertificate](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Publish-IAMSigningCertificate**

The following code example shows how to use Publish-IAMSigningCertificate.

**Tools for PowerShell**

**Example 1:** This example uploads a new X.509 signing certificate and associates it with the IAM user named Bob. The file containing the certificate body is PEM encoded. The CertificateBody parameter requires the actual contents of the certificate file rather than the file name. You must use the -Raw switch parameter to successfully process the file.

```
Publish-IAMSigningCertificate -UserName Bob -CertificateBody (Get-Content -Raw SampleSigningCert.pem)
```

**Output:**

```
CertificateBody : -----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMCVVMx
zA1BgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMDQwDQYDVQQKEwZBbWF6b24x
b24xFDASBgNVBAoTC01BTSBd25zb24x
```

---
CertificateId   : Y3EK7RMEXAMPLESV33FCEXAMPLEHMJLU
Status          : Active
UploadDate      : 4/20/2015 1:26:01 PM
UserName        : Bob

- For API details, see [UploadSigningCertificate](#) in AWS Tools for PowerShell Cmdlet Reference.

**Register-IAMGroupPolicy**

The following code example shows how to use Register-IAMGroupPolicy.

**Tools for PowerShell**

**Example 1:** This example attaches the customer managed policy named TesterPolicy to the IAM group Testers. The users in that group are immediately affected by the permissions defined in the default version of that policy.

```powershell
Register-IAMGroupPolicy -GroupName Testers -PolicyArn arn:aws:iam::123456789012:policy/TesterPolicy
```

**Example 2:** This example attaches the AWS managed policy named AdministratorAccess to the IAM group Admins. The users in that group are immediately affected by the permissions defined in the latest version of that policy.

```powershell
Register-IAMGroupPolicy -GroupName Admins -PolicyArn arn:aws:iam::aws:policy/AdministratorAccess
```

- For API details, see [AttachGroupPolicy](#) in AWS Tools for PowerShell Cmdlet Reference.

**Register-IAMRolePolicy**

The following code example shows how to use Register-IAMRolePolicy.

**Example 1:** This example attaches the customer managed policy named TesterPolicy to the IAM group Testers. The users in that group are immediately affected by the permissions defined in the default version of that policy.

```powershell
Register-IAMRolePolicy -GroupName Testers -PolicyArn arn:aws:iam::123456789012:policy/TesterPolicy
```

**Example 2:** This example attaches the AWS managed policy named AdministratorAccess to the IAM group Admins. The users in that group are immediately affected by the permissions defined in the latest version of that policy.

```powershell
Register-IAMRolePolicy -GroupName Admins -PolicyArn arn:aws:iam::aws:policy/AdministratorAccess
```
Tools for PowerShell

Example 1: This example attaches the AWS managed policy named SecurityAudit to the IAM role CoSecurityAuditors. The users who assume that role are immediately affected by the permissions defined in the latest version of that policy.

```powershell
```

• For API details, see AttachRolePolicy in AWS Tools for PowerShell Cmdlet Reference.

Register-IAMUserPolicy

The following code example shows how to use Register-IAMUserPolicy.

Tools for PowerShell

Example 1: This example attaches the AWS managed policy named AmazonCognitoPowerUser to the IAM user Bob. The user is immediately affected by the permissions defined in the latest version of that policy.

```powershell
Register-IAMUserPolicy -UserName Bob -PolicyArn arn:aws:iam::aws:policy/AmazonCognitoPowerUser
```

• For API details, see AttachUserPolicy in AWS Tools for PowerShell Cmdlet Reference.

Remove-IAMAccessKey

The following code example shows how to use Remove-IAMAccessKey.

Tools for PowerShell

Example 1: This example deletes the AWS access key pair with the key ID AKIAI0SFO0DDNN7EXAMPLE from the user named Bob.

```powershell
Remove-IAMAccessKey -AccessKeyId AKIAI0SFO0DDNN7EXAMPLE -UserName Bob -Force
```

• For API details, see DeleteAccessKey in AWS Tools for PowerShell Cmdlet Reference.
Remove-IAMAccountAlias

The following code example shows how to use Remove-IAMAccountAlias.

**Tools for PowerShell**

Example 1: This example removes the account alias from your AWS account. The user sign in page with the alias at https://mycompanyaws.signin.aws.amazon.com/console no longer works. You must instead use the original URL with your AWS account ID number at https://<accountidnumber>.signin.aws.amazon.com/console.

```
Remove-IAMAccountAlias -AccountAlias mycompanyaws
```

- For API details, see [DeleteAccountAlias](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Remove-IAMAccountPasswordPolicy

The following code example shows how to use Remove-IAMAccountPasswordPolicy.

**Tools for PowerShell**

Example 1: This example deletes the password policy for the AWS account and resets all values to their original defaults. If a password policy does not currently exist, the following error message appears: The account policy with name PasswordPolicy cannot be found.

```
Remove-IAMAccountPasswordPolicy
```

- For API details, see [DeleteAccountPasswordPolicy](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Remove-IAMClientIDFromOpenIDConnectProvider

The following code example shows how to use Remove-IAMClientIDFromOpenIDConnectProvider.

**Tools for PowerShell**

Example 1: This example removes the client ID My-TestApp-3 from the list of client IDs associated with the IAM OIDC provider whose ARN is arn:aws:iam::123456789012:oidc-provider/example.oidcprovider.com.
Remove-IAMClientIDFromOpenIDConnectProvider -ClientID My-TestApp-3 -OpenIDConnectProviderArn arn:aws:iam::123456789012:oidc-provider/example.oidcprovider.com

- For API details, see RemoveClientIDFromOpenIdConnectProvider in AWS Tools for PowerShell Cmdlet Reference.

Remove-IAMGroup

The following code example shows how to use Remove-IAMGroup.

Tools for PowerShell

Example 1: This example deletes the IAM group named MyTestGroup. The first command removes any IAM users that are members of the group, and the second command deletes the IAM group. Both commands work without any prompts for confirmation.

(Get-IAMGroup -GroupName MyTestGroup).Users | Remove-IAMUserFromGroup -GroupName MyTestGroup -Force
Remove-IAMGroup -GroupName MyTestGroup -Force

- For API details, see DeleteGroup in AWS Tools for PowerShell Cmdlet Reference.

Remove-IAMGroupPolicy

The following code example shows how to use Remove-IAMGroupPolicy.

Tools for PowerShell

Example 1: This example removes the inline policy named TesterPolicy from the IAM group Testers. The users in that group immediately lose the permissions defined in that policy.

Remove-IAMGroupPolicy -GroupName Testers -PolicyName TestPolicy

- For API details, see DeleteGroupPolicy in AWS Tools for PowerShell Cmdlet Reference.

Remove-IAMInstanceProfile

The following code example shows how to use Remove-IAMInstanceProfile.
Tools for PowerShell

Example 1: This example deletes the EC2 instance profile named MyAppInstanceProfile. The first command detaches any roles from the instance profile, and then the second command deletes the instance profile.

```powershell
(Get-IAMInstanceProfile -InstanceProfileName MyAppInstanceProfile).Roles | Remove-IAMRoleFromInstanceProfile -InstanceProfileName MyAppInstanceProfile
Remove-IAMInstanceProfile -InstanceProfileName MyAppInstanceProfile
```

- For API details, see [DeleteInstanceProfile](https://docs.aws.amazon.com/powershell/latest/reference/deleteinstanceprofile.html) in *AWS Tools for PowerShell Cmdlet Reference*.

Remove-IAMLoginProfile

The following code example shows how to use Remove-IAMLoginProfile.

Tools for PowerShell

Example 1: This example deletes the login profile from the IAM user named Bob. This prevents the user from signing-in to the AWS console. It does not prevent the user from running any AWS CLI, PowerShell, or API calls using AWS access keys that might still be attached to the user account.

```powershell
Remove-IAMLoginProfile -UserName Bob
```

- For API details, see [DeleteLoginProfile](https://docs.aws.amazon.com/powershell/latest/reference/deleteloginprofile.html) in *AWS Tools for PowerShell Cmdlet Reference*.

Remove-IAMOpenIDConnectProvider

The following code example shows how to use Remove-IAMOpenIDConnectProvider.

Tools for PowerShell

Example 1: This example deletes the IAM OIDC provider that connects to the provider example.oidcprovider.com. Ensure that you update or delete any roles that reference this provider in the Principal element of the role’s trust policy.

```powershell
Remove-IAMOpenIDConnectProvider -OpenIDConnectProviderArn arn:aws:iam::123456789012:oidc-provider/example.oidcprovider.com
```
• For API details, see `DeleteOpenIdConnectProvider` in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-IAMPolicy**

The following code example shows how to use `Remove-IAMPolicy`.

**Tools for PowerShell**

Example 1: This example deletes the policy whose ARN is `arn:aws:iam::123456789012:policy/MySamplePolicy`. Before you can delete the policy, you must first delete all versions except the default by running `Remove-IAMPolicyVersion`. You must also detach the policy from any IAM users, groups, or roles.

```
Remove-IAMPolicy -PolicyArn arn:aws:iam::123456789012:policy/MySamplePolicy
```

Example 2: This example deletes a policy by first deleting all the non-default policy versions, detaching it from all attached IAM entities, and finally deleting the policy itself. The first line retrieves the policy object. The second line retrieves all the policy versions that are not flagged as the default version into a collection and then deletes each policy in the collection. The third line retrieves all of the IAM users, groups, and roles to which the policy is attached. Lines four through six detach the policy from each attached entity. The last line uses this command to remove the managed policy as well as the remaining default version. The example includes the `-Force` switch parameter on any line that needs it to suppress prompts for confirmation.

```
$pol = Get-IAMPolicy -PolicyArn arn:aws:iam::123456789012:policy/MySamplePolicy
$attached = Get-IAMEntitiesForPolicy -PolicyArn $pol.Arnn
$attached.PolicyGroups | Unregister-IAMGroupPolicy -PolicyArn $pol.arn
$attached.PolicyRoles | Unregister-IAMRolePolicy -PolicyArn $pol.arn
$attached.PolicyUsers | Unregister-IAMUserPolicy -PolicyArn $pol.arn
Remove-IAMPolicy $pol.Arnn -Force
```

• For API details, see `DeletePolicy` in *AWS Tools for PowerShell Cmdlet Reference*. 
Remove-IAMPolicyVersion

The following code example shows how to use Remove-IAMPolicyVersion.

Tools for PowerShell

Example 1: This example deletes the version identified as v2 from the policy whose ARN is `arn:aws:iam::123456789012:policy/MySamplePolicy`.

```
```

Example 2: This example deletes a policy by first deleting all non-default policy versions and then deleting the policy itself. The first line retrieves the policy object. The second line retrieves all of the policy versions that are not flagged as the default into a collection and then uses this command to delete each policy in the collection. The last line removes the policy itself as well as the remaining default version. Note that to successfully delete a managed policy, you must also detach the policy from any users, groups, or roles by using the Unregister-IAMUserPolicy, Unregister-IAMGroupPolicy, and Unregister-IAMRolePolicy commands. See the example for the Remove-IAMPolicy cmdlet.

```
$pol = Get-IAMPolicy -PolicyArn arn:aws:iam::123456789012:policy/MySamplePolicy
Remove-IAMPolicy -PolicyArn $pol.Arn -force
```

- For API details, see [DeletePolicyVersion](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Remove-IAMRole

The following code example shows how to use Remove-IAMRole.

Tools for PowerShell

Example 1: This example deletes the role named MyNewRole from the current IAM account. Before you can delete the role you must first use the Unregister-IAMRolePolicy command to detach any managed policies. Inline policies are deleted with the role.

```
Remove-IAMRole -RoleName MyNewRole
```
Example 2: This example detaches any managed policies from the role named `MyNewRole` and then deletes the role. The first line retrieves any managed policies attached to the role as a collection and then detaches each policy in the collection from the role. The second line deletes the role itself. Inline policies are deleted along with the role.

```powershell
Get-IAMAttachedRolePolicyList -RoleName MyNewRole | Unregister-IAMRolePolicy -RoleName MyNewRole
Remove-IAMRole -RoleName MyNewRole
```

- For API details, see [DeleteRole](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Remove-IAMRoleFromInstanceProfile

The following code example shows how to use `Remove-IAMRoleFromInstanceProfile`.

#### Tools for PowerShell

Example 1: This example deletes the role named `MyNewRole` from the EC2 instance profile named `MyNewRole`. An instance profile that is created in the IAM console always has the same name as the role, as in this example. If you create them in the API or CLI, then they can have different names.

```powershell
Remove-IAMRoleFromInstanceProfile -InstanceProfileName MyNewRole -RoleName MyNewRole -Force
```

- For API details, see [RemoveRoleFromInstanceProfile](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Remove-IAMRolePermissionsBoundary

The following code example shows how to use `Remove-IAMRolePermissionsBoundary`.

#### Tools for PowerShell

Example 1: This example shows how to remove the permission boundary attached to an IAM role.

```powershell
Remove-IAMRolePermissionsBoundary -RoleName MyRoleName
```
Remove-IAMRolePolicy

The following code example shows how to use Remove-IAMRolePolicy.

Tools for PowerShell

Example 1: This example deletes the inline policy S3AccessPolicy that is embedded in the IAM role S3BackupRole.

Remove-IAMRolePolicy -PolicyName S3AccessPolicy -RoleName S3BackupRole

• For API details, see DeleteRolePermissionsBoundary in AWS Tools for PowerShell Cmdlet Reference.

Remove-IAMRoleTag

The following code example shows how to use Remove-IAMRoleTag.

Tools for PowerShell

Example 1: This example removes the tag from the role named "MyRoleName" with tag key as "abac". To remove multiple tags, provide a comma separated tag keys list.

Remove-IAMRoleTag -RoleName MyRoleName -TagKey "abac","xyzw"

• For API details, see UntagRole in AWS Tools for PowerShell Cmdlet Reference.

Remove-IAMSAMLProvider

The following code example shows how to use Remove-IAMSAMLProvider.

Tools for PowerShell

Example 1: This example deletes the IAM SAML 2.0 provider whose ARN is arn:aws:iam::123456789012:saml-provider/SAMLADFSProvider.

Remove-IAMSAMLProvider -SAMLProviderArn arn:aws:iam::123456789012:saml-provider/SAMLADFSProvider
For API details, see [DeleteSAMLProvider](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-IAMServerCertificate**

The following code example shows how to use Remove-IAMServerCertificate.

**Tools for PowerShell**

**Example 1:** This example deletes the server certificate named *MyServerCert*.

```powershell
Remove-IAMServerCertificate -ServerCertificateName MyServerCert
```

For API details, see [DeleteServerCertificate](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-IAMServiceLinkedRole**

The following code example shows how to use Remove-IAMServiceLinkedRole.

**Tools for PowerShell**

**Example 1:** This example deleted the service linked role. Please note that if the service is still using this role, then this command results in a failure.

```powershell
Remove-IAMServiceLinkedRole -RoleName AWSServiceRoleForAutoScaling_RoleNameEndsWithThis
```

For API details, see [DeleteServiceLinkedRole](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-IAMSigningCertificate**

The following code example shows how to use Remove-IAMSigningCertificate.

**Tools for PowerShell**

**Example 1:** This example deletes the signing certificate with the ID *Y3EK7RMEXAMPLESV33FCREXAMPLEMJLU* from the IAM user named *Bob*.

```powershell
Remove-IAMSigningCertificate -UserName Bob -CertificateId Y3EK7RMEXAMPLESV33FCREXAMPLEMJLU
```
Remove-IAMUser

The following code example shows how to use Remove-IAMUser.

Tools for PowerShell

Example 1: This example deletes the IAM user named Bob.

Remove-IAMUser -UserName Bob

Example 2: This example deletes the IAM user named Theresa along with any elements that must be deleted first.

$name = "Theresa"

# find any groups and remove user from them
$groups = Get-IAMGroupForUser -UserName $name
foreach ($group in $groups) { Remove-IAMUserFromGroup -GroupName $group.GroupName -UserName $name -Force }

# find any inline policies and delete them
$inlinepols = Get-IAMUserPolicies -UserName $name
foreach ($pol in $inlinepols) { Remove-IAMUserPolicy -PolicyName $pol -UserName $name -Force }

# find any managed polices and detach them
$managedpols = Get-IAMAttachedUserPolicies -UserName $name
foreach ($pol in $managedpols) { Unregister-IAMUserPolicy -PolicyArn $pol.PolicyArn -UserName $name }

# find any signing certificates and delete them
$certs = Get-IAMSigningCertificate -UserName $name
foreach ($cert in $certs) { Remove-IAMSigningCertificate -CertificateId $cert.CertificateId -UserName $name -Force }

# find any access keys and delete them
$keys = Get-IAMAccessKey -UserName $name
foreach ($key in $keys) { Remove-IAMAccessKey -AccessKeyId $key.AccessKeyId -UserName $name -Force }
# delete the user's login profile, if one exists - note: need to use try/catch to suppress not found error
try { $prof = Get-IAMLoginProfile -UserName $name -ea 0 } catch { out-null }
if ($prof) { Remove-IAMLoginProfile -UserName $name -Force }

# find any MFA device, detach it, and if virtual, delete it.
$mfa = Get-IAMMFADevice -UserName $name
if ($mfa) {
    Disable-IAMMFADevice -SerialNumber $mfa.SerialNumber -UserName $name
    if ($mfa.SerialNumber -like "arn:*") { Remove-IAMVirtualMFADevice -SerialNumber $mfa.SerialNumber }
}

# finally, remove the user
Remove-IAMUser -UserName $name -Force

- For API details, see [DeleteUser](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Remove-IAMUserFromGroup

The following code example shows how to use Remove-IAMUserFromGroup.

#### Tools for PowerShell

**Example 1:** This example removes the IAM user Bob from the group Testers.

```powershell
Remove-IAMUserFromGroup -GroupName Testers -UserName Bob
```

**Example 2:** This example finds any groups of which IAM user Theresa is a member, and then removes Theresa from those groups.

```powershell
$groups = Get-IAMGroupForUser -UserName Theresa
foreach ($group in $groups) { Remove-IAMUserFromGroup -GroupName $group.GroupName -UserName Theresa -Force }
```

**Example 3:** This example shows an alternate way of removing the IAM user Bob from the Testers group.

```powershell
Get-IAMGroupForUser -UserName Bob | Remove-IAMUserFromGroup -UserName Bob -GroupName Testers -Force
```
- For API details, see `RemoveUserFromGroup` in `AWS Tools for PowerShell Cmdlet Reference`.

### Remove-IAMUserPermissionsBoundary

The following code example shows how to use `Remove-IAMUserPermissionsBoundary`.

**Tools for PowerShell**

**Example 1:** This example shows how to remove the permission boundary attached to an IAM user.

```
Remove-IAMUserPermissionsBoundary -UserName joe
```

- For API details, see `DeleteUserPermissionsBoundary` in `AWS Tools for PowerShell Cmdlet Reference`.

### Remove-IAMUserPolicy

The following code example shows how to use `Remove-IAMUserPolicy`.

**Tools for PowerShell**

**Example 1:** This example deletes the inline policy named `AccessToEC2Policy` that is embedded in the IAM user named `Bob`.

```
Remove-IAMUserPolicy -PolicyName AccessToEC2Policy -UserName Bob
```

**Example 2:** This example finds all of the inline policies that are embedded in the IAM user named `Theresa` and then deletes them.

```
$inlinepols = Get-IAMUserPolicies -UserName Theresa
foreach ($pol in $inlinepols) { Remove-IAMUserPolicy -PolicyName $pol -UserName Theresa -Force}
```

- For API details, see `DeleteUserPolicy` in `AWS Tools for PowerShell Cmdlet Reference`.

### Remove-IAMUserTag

The following code example shows how to use `Remove-IAMUserTag`.

**Tools for PowerShell**
Tools for PowerShell

Example 1: This example removes the tag from the user named "joe" with tag key as "abac" and "xyzw". To remove multiple tags, provide a comma separated tag keys list.

Remove-IAMUserTag -UserName joe -TagKey "abac","xyzw"

- For API details, see UntagUser in AWS Tools for PowerShell Cmdlet Reference.

Remove-IAMVirtualMFADevice

The following code example shows how to use Remove-IAMVirtualMFADevice.

Tools for PowerShell

Example 1: This example deletes the IAM virtual MFA device whose ARN is arn:aws:iam::123456789012:mfa/bob.

Remove-IAMVirtualMFADevice -SerialNumber arn:aws:iam::123456789012:mfa/bob

Example 2: This example checks to see whether the IAM user Theresa has an MFA device assigned. If one is found, the device is disabled for the IAM user. If the device is virtual, then it is also deleted.

$mfa = Get-IAMMFADevice -UserName Theresa
if ($mfa) {
    Disable-IAMMFADevice -SerialNumber $mfa.SerialNumber -UserName $name
    if ($mfa.SerialNumber -like "arn:*") { Remove-IAMVirtualMFADevice -SerialNumber $mfa.SerialNumber }
}

- For API details, see DeleteVirtualMfaDevice in AWS Tools for PowerShell Cmdlet Reference.

Request-IAMCredentialReport

The following code example shows how to use Request-IAMCredentialReport.
Example 1: This example requests generation of a new report, which can be done every four hours. If the last report is still recent the State field reads COMPLETE. Use `Get-IAMCredentialReport` to view the completed report.

```powershell
Request-IAMCredentialReport
```

**Output:**

<table>
<thead>
<tr>
<th>Description</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>No report exists. Starting a new report generation task</td>
<td>STARTED</td>
</tr>
</tbody>
</table>

- For API details, see [GenerateCredentialReport](#) in *AWS Tools for PowerShell Cmdlet Reference*.

---

**Request-IAMServiceLastAccessedDetail**

The following code example shows how to use `Request-IAMServiceLastAccessedDetail`.

**Tools for PowerShell**

Example 1: This example is equivalent cmdlet of `GenerateServiceLastAccessedDetails` API. This provides with a job id which can be used in `Get-IAMServiceLastAccessedDetail` and `Get-IAMServiceLastAccessedDetailWithEntity`.

```powershell
Request-IAMServiceLastAccessedDetail -Arn arn:aws:iam::123456789012:user/TestUser
```

- For API details, see [GenerateServiceLastAccessedDetails](#) in *AWS Tools for PowerShell Cmdlet Reference*.

---

**Set-IAMDefaultPolicyVersion**

The following code example shows how to use `Set-IAMDefaultPolicyVersion`.

**Tools for PowerShell**

Example 1: This example sets the v2 version of the policy whose ARN is `arn:aws:iam::123456789012:policy/MyPolicy` as the default active version.

- For API details, see SetDefaultPolicyVersion in AWS Tools for PowerShell Cmdlet Reference.

Set-IAMRolePermissionsBoundary

The following code example shows how to use Set-IAMRolePermissionsBoundary.

Tools for PowerShell

Example 1: This example shows how to set the Permission boundary for a IAM Role. You can set AWS Managed policies or Custom policies as permission boundary.

Set-IAMRolePermissionsBoundary -RoleName MyRoleName -PermissionsBoundary arn:aws:iam::123456789012:policy/intern-boundary

- For API details, see PutRolePermissionsBoundary in AWS Tools for PowerShell Cmdlet Reference.

Set-IAMUserPermissionsBoundary

The following code example shows how to use Set-IAMUserPermissionsBoundary.

Tools for PowerShell

Example 1: This example shows how to set the Permission boundary for the user. You can set AWS Managed policies or Custom policies as permission boundary.

Set-IAMUserPermissionsBoundary -UserName joe -PermissionsBoundary arn:aws:iam::123456789012:policy/intern-boundary

- For API details, see PutUserPermissionsBoundary in AWS Tools for PowerShell Cmdlet Reference.

Sync-IAMMFADevice

The following code example shows how to use Sync-IAMMFADevice.
Tools for PowerShell

Example 1: This example synchronizes the MFA device that is associated with the IAM user Bob and whose ARN is `arn:aws:iam::123456789012:mfa/bob` with an authenticator program that provided the two authentication codes.

```
Sync-IAMMFADevice -SerialNumber arn:aws:iam::123456789012:mfa/theresa -AuthenticationCode1 123456 -AuthenticationCode2 987654 -UserName Bob
```

Example 2: This example synchronizes the IAM MFA device that is associated with the IAM user Theresa with a physical device that has the serial number ABCD12345678 and that provided the two authentication codes.

```
Sync-IAMMFADevice -SerialNumber ABCD12345678 -AuthenticationCode1 123456 -AuthenticationCode2 987654 -UserName Theresa
```

- For API details, see ResyncMfaDevice in AWS Tools for PowerShell Cmdlet Reference.

Unregister-IAMGroupPolicy

The following code example shows how to use Unregister-IAMGroupPolicy.

Tools for PowerShell

Example 1: This example detaches the managed group policy whose ARN is `arn:aws:iam::123456789012:policy/TesterAccessPolicy` from the group named Testers.

```
```

Example 2: This example finds all the managed policies that are attached to the group named Testers and detaches them from the group.

```
Get-IAMAttachedGroupPolicies -GroupName Testers | Unregister-IAMGroupPolicy -GroupName Testers
```

- For API details, see DetachGroupPolicy in AWS Tools for PowerShell Cmdlet Reference.
Unregister-IAMRolePolicy

The following code example shows how to use Unregister-IAMRolePolicy.

**Tools for PowerShell**

**Example 1:** This example detaches the managed group policy whose ARN is `arn:aws:iam::123456789012:policy/FederatedTesterAccessPolicy` from the role named FedTesterRole.

```
```

**Example 2:** This example finds all of the managed policies that are attached to the role named FedTesterRole and detaches them from the role.

```
Get-IAMAttachedRolePolicyList -RoleName FedTesterRole | Unregister-IAMRolePolicy -Rolename FedTesterRole
```

- For API details, see [DetachRolePolicy](#) in AWS Tools for PowerShell Cmdlet Reference.

Unregister-IAMUserPolicy

The following code example shows how to use Unregister-IAMUserPolicy.

**Tools for PowerShell**

**Example 1:** This example detaches the managed policy whose ARN is `arn:aws:iam::123456789012:policy/TesterPolicy` from the IAM user named Bob.

```
Unregister-IAMUserPolicy -UserName Bob -PolicyArn arn:aws:iam::123456789012:policy/TesterPolicy
```

**Example 2:** This example finds all the managed policies that are attached to the IAM user named Theresa and detaches those policies from the user.

```
Get-IAMAttachedUserPolicyList -UserName Theresa | Unregister-IAMUserPolicy -Username Theresa
```

- For API details, see [DetachUserPolicy](#) in AWS Tools for PowerShell Cmdlet Reference.
Update-IAMAccessKey

The following code example shows how to use Update-IAMAccessKey.

**Tools for PowerShell**

**Example 1:** This example changes the status of the access key AKIAIOSFODNN7EXAMPLE for the IAM user named Bob to Inactive.

```
Update-IAMAccessKey -UserName Bob -AccessKeyId AKIAIOSFODNN7EXAMPLE -Status Inactive
```

- For API details, see [UpdateAccessKey](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Update-IAMAccountPasswordPolicy

The following code example shows how to use Update-IAMAccountPasswordPolicy.

**Tools for PowerShell**

**Example 1:** This example updates the password policy for the account with the specified settings. Note that any parameters that are not included in the command are not left unmodified. Instead, they are reset to default values.

```
Update-IAMAccountPasswordPolicy -AllowUsersToChangePasswords $true -HardExpiry $false -MaxPasswordAge 90 -MinimumPasswordLength 8 -PasswordReusePrevention 20 -RequireLowercaseCharacters $true -RequireNumbers $true -RequireSymbols $true -RequireUppercaseCharacters $true
```

- For API details, see [UpdateAccountPasswordPolicy](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Update-IAMAssumeRolePolicy

The following code example shows how to use Update-IAMAssumeRolePolicy.

**Tools for PowerShell**

**Example 1:** This example updates the IAM role named ClientRole with a new trust policy, the contents of which come from the file ClientRolePolicy.json. Note that you must use the `-Raw` switch parameter to successfully process the contents of the JSON file.
Update-IAMAssumeRolePolicy -RoleName ClientRole -PolicyDocument (Get-Content -raw ClientRolePolicy.json)

- For API details, see [UpdateAssumeRolePolicy](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Update-IAMGroup**

The following code example shows how to use Update-IAMGroup.

**Tools for PowerShell**

*Example 1:* This example renames the IAM group Testers to AppTesters.

```
Update-IAMGroup -GroupName Testers -NewGroupName AppTesters
```

*Example 2:* This example changes the path of the IAM group AppTesters to /Org1/Org2/. This changes the ARN for the group to `arn:aws:iam::123456789012:group/Org1/Org2/AppTesters`.

```
Update-IAMGroup -GroupName AppTesters -NewPath /Org1/Org2/
```

- For API details, see [UpdateGroup](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Update-IAMLoginProfile**

The following code example shows how to use Update-IAMLoginProfile.

**Tools for PowerShell**

*Example 1:* This example sets a new temporary password for the IAM user Bob, and requires the user to change the password the next time the user signs in.

```
Update-IAMLoginProfile -UserName Bob -Password "P@ssw0rd1234" -PasswordResetRequired $true
```

- For API details, see [UpdateLoginProfile](#) in *AWS Tools for PowerShell Cmdlet Reference*. 
Update-IAMOpenIDConnectProviderThumbprint

The following code example shows how to use Update-IAMOpenIDConnectProviderThumbprint.

Tools for PowerShell

Example 1: This example updates the certificate thumbprint list for the OIDC provider whose ARN is arn:aws:iam::123456789012:oidc-provider/example.oidcprovider.com to use a new thumbprint. The OIDC provider shares the new value when the certificate that is associated with the provider changes.

```
Update-IAMOpenIDConnectProviderThumbprint -OpenIDConnectProviderArn arn:aws:iam::123456789012:oidc-provider/example.oidcprovider.com -ThumbprintList 7359755EXAMPLEabc3060bce3EXAMPLEec4542a3
```

- For API details, see UpdateOpenIdConnectProviderThumbprint in AWS Tools for PowerShell Cmdlet Reference.

Update-IAMRole

The following code example shows how to use Update-IAMRole.

Tools for PowerShell

Example 1: This example updates the role description and the maximum session duration value (in seconds) for which a role's session can be requested.

```
Update-IAMRole -RoleName MyRoleName -Description "My testing role" -MaxSessionDuration 43200
```

- For API details, see UpdateRole in AWS Tools for PowerShell Cmdlet Reference.

Update-IAMRoleDescription

The following code example shows how to use Update-IAMRoleDescription.

Tools for PowerShell

Example 1: This example updates the description of an IAM role in your account.
Update-IAMRoleDescription -RoleName MyRoleName -Description "My testing role"

- For API details, see UpdateRoleDescription in AWS Tools for PowerShell Cmdlet Reference.

**Update-IAMSAMLProvider**

The following code example shows how to use Update-IAMSAMLProvider.

**Tools for PowerShell**

**Example 1:** This example updates the SAML provider in IAM whose ARN is arn:aws:iam::123456789012:saml-provider/SAMLADFS with a new SAML metadata document from the file SAMLMetaData.xml. Note that you must use the -Raw switch parameter to successfully process the contents of the JSON file.

```
Update-IAMSAMLProvider -SAMLProviderArn arn:aws:iam::123456789012:saml-provider/SAMLADFS -SAMLMetadataDocument (Get-Content -Raw SAMLMetaData.xml)
```

- For API details, see UpdateSamlProvider in AWS Tools for PowerShell Cmdlet Reference.

**Update-IAMServerCertificate**

The following code example shows how to use Update-IAMServerCertificate.

**Tools for PowerShell**

**Example 1:** This example renames the certificate named MyServerCertificate to MyRenamedServerCertificate.

```
```

**Example 2:** This example moves the certificate named MyServerCertificate to the path /Org1/Org2/. This changes the ARN for the resource to arn:aws:iam::123456789012:server-certificate/Org1/Org2/MyServerCertificate.

```
```
• For API details, see UpdateServerCertificate in AWS Tools for PowerShell Cmdlet Reference.

Update-IAMSigningCertificate

The following code example shows how to use Update-IAMSigningCertificate.

Tools for PowerShell

Example 1: This example updates the certificate that is associated with the IAM user named Bob and whose certificate ID is Y3EK7RMEXAMPELSV33FCREXAMPEMJLU to mark it as inactive.

```
Update-IAMSigningCertificate -CertificateId Y3EK7RMEXAMPELSV33FCREXAMPEMJLU -UserName Bob -Status Inactive
```

• For API details, see UpdateSigningCertificate in AWS Tools for PowerShell Cmdlet Reference.

Update-IAMUser

The following code example shows how to use Update-IAMUser.

Tools for PowerShell

Example 1: This example renames the IAM user Bob to Robert.

```
Update-IAMUser -UserName Bob -NewUserName Robert
```

Example 2: This example changes the path of the IAM User Bob to /Org1/Org2/, which effectively changes the ARN for the user to arn:aws:iam::123456789012:user/Org1/Org2/bob.

```
Update-IAMUser -UserName Bob -NewPath /Org1/Org2/
```

• For API details, see UpdateUser in AWS Tools for PowerShell Cmdlet Reference.

Write-IAMGroupPolicy

The following code example shows how to use Write-IAMGroupPolicy.
Tools for PowerShell

Example 1: This example creates an inline policy named AppTesterPolicy and embeds it in the IAM group AppTesters. If an inline policy with the same name already exists, then it is overwritten. The JSON policy content comes from the file apptesterpolicy.json. Note that you must use the -Raw parameter to successfully process the content of the JSON file.

```
Write-IAMGroupPolicy -GroupName AppTesters -PolicyName AppTesterPolicy -PolicyDocument (Get-Content -Raw apptesterpolicy.json)
```

- For API details, see PutGroupPolicy in AWS Tools for PowerShell Cmdlet Reference.

Write-IAMRolePolicy

The following code example shows how to use Write-IAMRolePolicy.

Tools for PowerShell

Example 1: This example creates an inline policy named FedTesterRolePolicy and embeds it in the IAM role FedTesterRole. If an inline policy with the same name already exists, then it is overwritten. The JSON policy content comes from the file FedTesterPolicy.json. Note that you must use the -Raw parameter to successfully process the content of the JSON file.

```
Write-IAMRolePolicy -RoleName FedTesterRole -PolicyName FedTesterRolePolicy -PolicyDocument (Get-Content -Raw FedTesterPolicy.json)
```

- For API details, see PutRolePolicy in AWS Tools for PowerShell Cmdlet Reference.

Write-IAMUserPolicy

The following code example shows how to use Write-IAMUserPolicy.

Tools for PowerShell

Example 1: This example creates an inline policy named EC2AccessPolicy and embeds it in the IAM user Bob. If an inline policy with the same name already exists, then it is overwritten. The JSON policy content comes from the file EC2AccessPolicy.json. Note that you must use the -Raw parameter to successfully process the content of the JSON file.
Write-IAMUserPolicy -UserName Bob -PolicyName EC2AccessPolicy -PolicyDocument (Get-Content -Raw EC2AccessPolicy.json)

- For API details, see PutUserPolicy in AWS Tools for PowerShell Cmdlet Reference.

Kinesis examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with Kinesis.

*Actions* are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

*Scenarios* are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

**Topics**

- **Actions**

**Actions**

**Get-KINRecord**

The following code example shows how to use Get-KINRecord.

**Tools for PowerShell**

*Example 1:* This example shows how to return and extract data from a series of one or more records. The iterator supplied to Get-KINRecord determines the starting position of the records to return which in this example are captured into a variable, $records. Each individual record can then be accessed by indexing the $records collection. Assuming the data in the record is UTF-8 encoded text, the final command shows how you can extract the data from the MemoryStream in the object and return it as text to the console.

```powershell
$records
```
Get-KINRecord -ShardIterator "AAAAAAAAAAAGIc....9VnbiRNAP"

Output:

<table>
<thead>
<tr>
<th>MillisBehindLatest</th>
<th>NextShardIterator</th>
<th>Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>AAAAAAAAAAAAAERNIq...uDn11HuUs</td>
<td>{Key1, Key2}</td>
</tr>
</tbody>
</table>

$records.Records[0]

Output:

<table>
<thead>
<tr>
<th>ApproximateArrivalTimestamp</th>
<th>Data</th>
<th>PartitionKey</th>
<th>SequenceNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>3/7/2016 5:14:33 PM</td>
<td>System.IO.MemoryStream Key1</td>
<td></td>
<td>4955986459776...931586</td>
</tr>
</tbody>
</table>

[Text.Encoding]::UTF8.GetString($records.Records[0].Data.ToArray())

Output:

test data from string

- For API details, see GetRecords in AWS Tools for PowerShell Cmdlet Reference.

Get-KINShardIterator

The following code example shows how to use Get-KINShardIterator.

Tools for PowerShell

Example 1: Returns a shard iterator for the specified shard and starting position. Details of the shard identifiers and sequence numbers can be obtained from the output of the Get-KINStream cmdlet, by referencing the Shards collection of the returned stream object. The returned iterator can be used with the Get-KINRecord cmdlet to pull data records in the shard.

Get-KINShardIterator -StreamName "mystream" -ShardId "shardId-000000000000" -ShardIteratorType AT_SEQUENCE_NUMBER -StartingSequenceNumber "495598645..."
Output:

```
AAAIAAAAADg....9Nvb1RNaP
```

- For API details, see `GetShardIterator` in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-KINStream**

The following code example shows how to use `Get-KINStream`.

**Tools for PowerShell**

**Example 1: Returns details of the specified stream.**

```
Get-KINStream -StreamName "mystream"
```

**Output:**

```
HasMoreShards : False
RetentionPeriodHours : 24
Shards : {}
StreamName : mystream
StreamStatus : ACTIVE
```

- For API details, see `DescribeStream` in *AWS Tools for PowerShell Cmdlet Reference*.

**New-KINStream**

The following code example shows how to use `New-KINStream`.

**Tools for PowerShell**

**Example 1: Creates a new stream.** By default this cmdlet returns no output so the -PassThru switch is added to return the value supplied to the -StreamName parameter for subsequent use.

```
$streamName = New-KINStream -StreamName "mystream" -ShardCount 1 -PassThru
```

- For API details, see `CreateStream` in *AWS Tools for PowerShell Cmdlet Reference*. 
Remove-KINStream

The following code example shows how to use Remove-KINStream.

Tools for PowerShell

Example 1: Deletes the specified stream. You are prompted for confirmation before the command executes. To suppress confirmation prompting use the -Force switch.

```
Remove-KINStream -StreamName "mystream"
```

- For API details, see [DeleteStream](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Write-KINRecord

The following code example shows how to use Write-KINRecord.

Tools for PowerShell

Example 1: Writes a record containing the string supplied to the -Text parameter.

```
Write-KINRecord -Text "test data from string" -StreamName "mystream" -PartitionKey "Key1"
```

Example 2: Writes a record containing the data contained in the specified file. The file is treated as a sequence of bytes so if it contains text, it should be written with any necessary encoding before using it with this cmdlet.

```
Write-KINRecord -FilePath "C:\TestData.txt" -StreamName "mystream" -PartitionKey "Key2"
```

- For API details, see [PutRecord](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Lambda examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with Lambda.
Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

• Actions

Actions

Add-LMResourceTag

The following code example shows how to use Add-LMResourceTag.

Tools for PowerShell

Example 1: Adds the three tags (Washington, Oregon and California) and their associated values to the specified function identified by its ARN.


• For API details, see TagResource in AWS Tools for PowerShell Cmdlet Reference.

Get-LMAccountSetting

The following code example shows how to use Get-LMAccountSetting.

Tools for PowerShell

Example 1: This sample displays to compare the Account Limit and Account Usage
Get-LMAccountSetting | Select-Object
@{Name="TotalCodeSizeLimit";Expression={$_.AccountLimit.TotalCodeSize}},
@{Name="TotalCodeSizeUsed";Expression={$_.AccountUsage.TotalCodeSize}}

**Output:**

<table>
<thead>
<tr>
<th>TotalCodeSizeLimit</th>
<th>TotalCodeSizeUsed</th>
</tr>
</thead>
<tbody>
<tr>
<td>8053063680</td>
<td>15078795</td>
</tr>
</tbody>
</table>

- For API details, see [GetAccountSettings](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-LMAlias**

The following code example shows how to use Get-LMAlias.

**Tools for PowerShell**

**Example 1:** This example retrieves the Routing Config weights for a specific Lambda Function Alias.

```powershell
Get-LMAlias -FunctionName "MylambdaFunction123" -Name "newlabel1" -Select RoutingConfig
```

**Output:**

<table>
<thead>
<tr>
<th>AdditionalVersionWeights</th>
</tr>
</thead>
<tbody>
<tr>
<td>{[1, 0.6]}</td>
</tr>
</tbody>
</table>

- For API details, see [GetAlias](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-LMFunctionConcurrency**

The following code example shows how to use Get-LMFunctionConcurrency.

**Tools for PowerShell**

**Example 1:** This examples gets the Reserved concurrency for the Lambda Function
Get-LMFunctionConcurrency -FunctionName "MylambdaFunction123" -Select *

Output:

ReservedConcurrentExecutions
-----------------------------
100

- For API details, see [GetFunctionConcurrency](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-LMFunctionConfiguration**

The following code example shows how to use Get-LMFunctionConfiguration.

**Tools for PowerShell**

**Example 1:** This example returns the version specific configuration of a Lambda Function.

Get-LMFunctionConfiguration -FunctionName "MylambdaFunction123" -Qualifier "PowershellAlias"

Output:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CodeSha256</td>
<td>uW0W0R7z+f0VyLuUg7+/D08hKFsq0SF4seuyUZJ/R8=</td>
</tr>
<tr>
<td>CodeSize</td>
<td>1426</td>
</tr>
<tr>
<td>DeadLetterConfig</td>
<td>Amazon.Lambda.Model.DeadLetterConfig</td>
</tr>
<tr>
<td>Description</td>
<td>Version 3 to test Aliases</td>
</tr>
<tr>
<td>Environment</td>
<td>Amazon.Lambda.Model.EnvironmentResponse</td>
</tr>
<tr>
<td>FunctionName</td>
<td>MylambdaFunction123</td>
</tr>
<tr>
<td>Handler</td>
<td>lambda_function.launch_instance</td>
</tr>
<tr>
<td>KMSKeyArn</td>
<td></td>
</tr>
<tr>
<td>LastModified</td>
<td>2019-12-25T09:52:59.872+0000</td>
</tr>
<tr>
<td>LastUpdateStatus</td>
<td>Successful</td>
</tr>
<tr>
<td>LastUpdateStatusReason</td>
<td></td>
</tr>
<tr>
<td>LastUpdateStatusReasonCode</td>
<td></td>
</tr>
<tr>
<td>Layers</td>
<td>{}</td>
</tr>
<tr>
<td>MasterArn</td>
<td></td>
</tr>
<tr>
<td>MemorySize</td>
<td>128</td>
</tr>
</tbody>
</table>
RevisionId                 : 5d7de38b-87f2-4260-8f8a-e87280e10c33
Role                       : arn:aws:iam::123456789012:role/service-role/lambda
Runtime                    : python3.8
State                      : Active
StateReason                :
StateReasonCode            :
Timeout                    : 600
TracingConfig              : Amazon.Lambda.Model.TracingConfigResponse
Version                    : 4
VpcConfig                  : Amazon.Lambda.Model.VpcConfigDetail

- For API details, see [GetFunctionConfiguration](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-LMFunctionList

The following code example shows how to use Get-LMFunctionList.

**Tools for PowerShell**

**Example 1: This sample displays all the Lambda functions with sorted code size**

```
Get-LMFunctionList | Sort-Object -Property CodeSize | Select-Object FunctionName, RunTime, Timeout, CodeSize
```

**Output:**

<table>
<thead>
<tr>
<th>FunctionName</th>
<th>Runtime</th>
<th>Timeout</th>
</tr>
</thead>
<tbody>
<tr>
<td>test</td>
<td>python2.7</td>
<td>3</td>
</tr>
<tr>
<td>243</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MylambdaFunction123</td>
<td>python3.8</td>
<td>600</td>
</tr>
<tr>
<td>659</td>
<td></td>
<td></td>
</tr>
<tr>
<td>myfuncpython1</td>
<td>python3.8</td>
<td>303</td>
</tr>
<tr>
<td>675</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- For API details, see [ListFunctions](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-LMPolicy

The following code example shows how to use Get-LMPolicy.
Tools for PowerShell

**Example 1: This sample displays the Function policy of the Lambda function**

Get-LMPolicy -FunctionName test -Select Policy

Output:

```json
{"Version":"2012-10-17","Id":"default","Statement":
{"Sid":"xxxx","Effect":"Allow","Principal":
```

- For API details, see [GetPolicy](https://aws.amazonaws.com) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-LMProvisionedConcurrencyConfig**

The following code example shows how to use Get-LMProvisionedConcurrencyConfig.

Tools for PowerShell

**Example 1: This example gets the provisioned Concurrency Configuration for the specified Alias of the Lambda Function.**

C:\>Get-LMProvisionedConcurrencyConfig -FunctionName "MylambdaFunction123" -Qualifier "NewAlias1"

Output:

```
AllocatedProvisionedConcurrentExecutions : 0
AvailableProvisionedConcurrentExecutions : 0
LastModified                             : 2020-01-15T03:21:26+0000
RequestedProvisionedConcurrentExecutions : 70
Status                                   : IN_PROGRESS
StatusReason                             :
```

- For API details, see [GetProvisionedConcurrencyConfig](https://aws.amazonaws.com) in *AWS Tools for PowerShell Cmdlet Reference*. 
Get-LMProvisionedConcurrencyConfigList

The following code example shows how to use Get-LMProvisionedConcurrencyConfigList.

Tools for PowerShell

Example 1: This example retrieves the list of provisioned concurrency configurations for a Lambda function.

```
Get-LMProvisionedConcurrencyConfigList -FunctionName "MylambdaFunction123"
```

• For API details, see ListProvisionedConcurrentyConfigs in AWS Tools for PowerShell Cmdlet Reference.

Get-LMResourceTag

The following code example shows how to use Get-LMResourceTag.

Tools for PowerShell

Example 1: Retrieves the tags and their values currently set on the specified function.

```
```

Output:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>California</td>
<td>Sacramento</td>
</tr>
<tr>
<td>Oregon</td>
<td>Salem</td>
</tr>
<tr>
<td>Washington</td>
<td>Olympia</td>
</tr>
</tbody>
</table>

• For API details, see ListTags in AWS Tools for PowerShell Cmdlet Reference.

Get-LMVersionsByFunction

The following code example shows how to use Get-LMVersionsByFunction.
Example 1: This example returns the list of version specific configurations for each version of the Lambda Function.

```
Get-LMVersionsByFunction -FunctionName "MylambdaFunction123"
```

Output:

<table>
<thead>
<tr>
<th>FunctionName</th>
<th>RoleName</th>
<th>Runtime</th>
<th>MemorySize</th>
<th>Timeout</th>
<th>CodeSize</th>
<th>LastModified</th>
</tr>
</thead>
<tbody>
<tr>
<td>MylambdaFunction123</td>
<td></td>
<td>python3.8</td>
<td>128</td>
<td>600</td>
<td>659</td>
<td>2020-01-10T03:20:56.390+0000 lambda</td>
</tr>
<tr>
<td>MylambdaFunction123</td>
<td></td>
<td>python3.8</td>
<td>128</td>
<td>5</td>
<td>1426</td>
<td>2019-12-25T09:19:02.238+0000 lambda</td>
</tr>
<tr>
<td>MylambdaFunction123</td>
<td></td>
<td>python3.8</td>
<td>128</td>
<td>5</td>
<td>1426</td>
<td>2019-12-25T09:39:36.779+0000 lambda</td>
</tr>
<tr>
<td>MylambdaFunction123</td>
<td></td>
<td>python3.8</td>
<td>128</td>
<td>600</td>
<td>1426</td>
<td>2019-12-25T09:52:59.872+0000 lambda</td>
</tr>
</tbody>
</table>

- For API details, see [ListVersionsByFunction](#) in *AWS Tools for PowerShell Cmdlet Reference*.

New-LMAlias

The following code example shows how to use New-LMAlias.

Example 1: This example creates a New Lambda Alias for specified version and routing configuration to specify the percentage of invocation requests that it receives.

```
New-LMAlias -FunctionName "MylambdaFunction123" -RoutingConfig_AdditionalVersionWeight @{Name="1";Value="0.6"} -Description "Alias for version 4" -FunctionVersion 4 -Name "PowershellAlias"
```

- For API details, see [CreateAlias](#) in *AWS Tools for PowerShell Cmdlet Reference*.  

Lambda
Publish-LMFunction

The following code example shows how to use Publish-LMFunction.

Tools for PowerShell

Example 1: This example creates a new C# (dotnetcore1.0 runtime) function named MyFunction in AWS Lambda, providing the compiled binaries for the function from a zip file on the local file system (relative or absolute paths may be used). C# Lambda functions specify the handler for the function using the designation AssemblyName::Namespace.ClassName::MethodName. You should replace the assembly name (without .dll suffix), namespace, class name and method name parts of the handler spec appropriately. The new function will have environment variables 'envvar1' and 'envvar2' set up from the provided values.

```
Publish-LMFunction -Description "My C# Lambda Function" ` 
  -FunctionName MyFunction ` 
  -ZipFilename .\MyFunctionBinaries.zip ` 
  -Handler "AssemblyName::Namespace.ClassName::MethodName" ` 
  -Role "arn:aws:iam::123456789012:role/LambdaFullExecRole" ` 
  -Runtime dotnetcore1.0 ` 
  -Environment_Variable @{ "envvar1"="value";"envvar2"="value" }
```

Output:

<table>
<thead>
<tr>
<th>CodeSha256</th>
<th>/NgBMd...gq71I=</th>
</tr>
</thead>
<tbody>
<tr>
<td>CodeSize</td>
<td>214784</td>
</tr>
<tr>
<td>DeadLetterConfig :</td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>My C# Lambda Function</td>
</tr>
<tr>
<td>Environment</td>
<td>Amazon.Lambda.Model.EnvironmentResponse</td>
</tr>
<tr>
<td>FunctionArn</td>
<td>arn:aws:lambda:us-west-2:123456789012:function:ToUpper</td>
</tr>
<tr>
<td>FunctionName</td>
<td>MyFunction</td>
</tr>
<tr>
<td>Handler</td>
<td>AssemblyName::Namespace.ClassName::MethodName</td>
</tr>
<tr>
<td>KMSKeyArn</td>
<td>:</td>
</tr>
<tr>
<td>LastModified</td>
<td>2016-12-29T23:50:14.207+0000</td>
</tr>
<tr>
<td>MemorySize</td>
<td>128</td>
</tr>
<tr>
<td>Role</td>
<td>arn:aws:iam::123456789012:role/LambdaFullExecRole</td>
</tr>
<tr>
<td>Runtime</td>
<td>dotnetcore1.0</td>
</tr>
<tr>
<td>Timeout</td>
<td>3</td>
</tr>
<tr>
<td>Version</td>
<td>$LATEST</td>
</tr>
<tr>
<td>VpcConfig</td>
<td>:</td>
</tr>
</tbody>
</table>
Example 2: This example is similar to the previous one except the function binaries are first uploaded to an Amazon S3 bucket (which must be in the same region as the intended Lambda function) and the resulting S3 object is then referenced when creating the function.

```powershell
Write-S3Object -BucketName mybucket -Key MyFunctionBinaries.zip -File .\MyFunctionBinaries.zip
Publish-LMFunction -Description "My C# Lambda Function" ` -FunctionName MyFunction ` -BucketName mybucket ` -Key MyFunctionBinaries.zip ` -Handler "AssemblyName::Namespace.ClassName::MethodName" ` -Role "arn:aws:iam::123456789012:role/LambdaFullExecRole" ` -Runtime dotnetcore1.0 ` -Environment_Variable @{ "envvar1"="value";"envvar2"="value" }
```

- For API details, see [CreateFunction](https://aws.amazon.com/tools/powershell-cmdlet-reference/) in *AWS Tools for PowerShell Cmdlet Reference*.

**Publish-LMVersion**

The following code example shows how to use `Publish-LMVersion`.

**Tools for PowerShell**

**Example 1: This example creates a version for the existing snapshot of Lambda Function Code**

```powershell
Publish-LMVersion -FunctionName "MylambdaFunction123" -Description "Publishing Existing Snapshot of function code as a new version through Powershell"
```


**Remove-LMAlias**

The following code example shows how to use `Remove-LMAlias`.

**Tools for PowerShell**

**Example 1: This example deletes the Lambda function Alias mentioned in the command.**

```powershell
Remove-LMAlias -FunctionName "MylambdaFunction123" -Name "NewAlias"
```
For API details, see [DeleteAlias](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-LMFunction**

The following code example shows how to use `Remove-LMFunction`.

**Tools for PowerShell**

**Example 1:** This example deletes a specific version of a Lambda function

```powershell
Remove-LMFunction -FunctionName "MylambdaFunction123" -Qualifier '3'
```

For API details, see [DeleteFunction](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-LMFunctionConcurrency**

The following code example shows how to use `Remove-LMFunctionConcurrency`.

**Tools for PowerShell**

**Example 1:** This example removes the Function Concurrency of the Lambda Function.

```powershell
Remove-LMFunctionConcurrency -FunctionName "MylambdaFunction123"
```

For API details, see [DeleteFunctionConcurrency](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-LMPermission**

The following code example shows how to use `Remove-LMPermission`.

**Tools for PowerShell**

**Example 1:** This example removes the function policy for the specified StatementId of a Lambda Function.

```powershell
$policy = Get-LMPolicy -FunctionName "MylambdaFunction123" -Select Policy | ConvertFrom-Json| Select-Object -ExpandProperty Statement
Remove-LMPermission -FunctionName "MylambdaFunction123" -StatementId $policy[0].Sid
```

For API details, see [RemovePermission](#) in *AWS Tools for PowerShell Cmdlet Reference*.  

Lambda
Remove-LMProvisionedConcurrencyConfig

The following code example shows how to use Remove-LMProvisionedConcurrencyConfig.

Tools for PowerShell

Example 1: This example removes the Provisioned Concurrency Configuration for a specific Alias.

Remove-LMProvisionedConcurrencyConfig -FunctionName "MylambdaFunction123" -Qualifier "NewAlias1"

• For API details, see DeleteProvisionedConcurrencyConfig in AWS Tools for PowerShell Cmdlet Reference.

Remove-LMResourceTag

The following code example shows how to use Remove-LMResourceTag.

Tools for PowerShell

Example 1: Removes the supplied tags from a function. The cmdlet will prompt for confirmation before proceeding unless the -Force switch is specified. A single call is made to the service to remove the tags.


Example 2: Removes the supplied tags from a function. The cmdlet will prompt for confirmation before proceeding unless the -Force switch is specified. Once call to the service is made per supplied tag.


• For API details, see UntagResource in AWS Tools for PowerShell Cmdlet Reference.

Update-LMAlias

The following code example shows how to use Update-LMAlias.
Tools for PowerShell

Example 1: This example updates the Configuration of an existing Lambda function Alias. It updates the RoutingConfiguration value to shift 60% (0.6) of traffic to version 1

Update-LMAlias -FunctionName "MylambdaFunction123" -Description "Alias for version 2" -FunctionVersion 2 -Name "newlabel1" -RoutingConfig_AdditionalVersionWeight @{Name="1";Value="0.6"}

- For API details, see UpdateAlias in AWS Tools for PowerShell Cmdlet Reference.

Update-LMFunctionCode

The following code example shows how to use Update-LMFunctionCode.

Tools for PowerShell

Example 1: Updates the function named 'MyFunction' with new content contained in the specified zip file. For a C# .NET Core Lambda function the zip file should contain the compiled assembly.

Update-LMFunctionCode -FunctionName MyFunction -ZipFilename .\UpdatedCode.zip

Example 2: This example is similar to the previous one but uses an Amazon S3 object containing the updated code to update the function.

Update-LMFunctionCode -FunctionName MyFunction -BucketName mybucket -Key UpdatedCode.zip

- For API details, see UpdateFunctionCode in AWS Tools for PowerShell Cmdlet Reference.

Update-LMFunctionConfiguration

The following code example shows how to use Update-LMFunctionConfiguration.

Tools for PowerShell

Example 1: This example updates the existing Lambda Function Configuration

Update-LMFunctionConfiguration -FunctionName "MylambdaFunction123" -Handler "lambda_function.launch_instance" -Timeout 600 -Environment_Variable
Tools for PowerShell

Example 1: This example applies the concurrency settings for the Function as a whole.

```powershell
Write-LMFunctionConcurrency -FunctionName "MylambdaFunction123" -
ReservedConcurrentExecution 100
```

For API details, see `PutFunctionConcurrency` in AWS Tools for PowerShell Cmdlet Reference.

Tools for PowerShell

Example 1: This example adds a provisioned concurrency configuration to a Function's Alias

```powershell
Write-LMProvisionedConcurrencyConfig -FunctionName "MylambdaFunction123" -
ProvisionedConcurrentExecution 20 -Qualifier "NewAlias1"
```

For API details, see `PutProvisionedConcurrencyConfig` in AWS Tools for PowerShell Cmdlet Reference.

Amazon ML examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with Amazon ML.
*Actions* are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

*Scenarios* are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

**Topics**
- **Actions**

**Actions**

**Get-MLBatchPrediction**

The following code example shows how to use `Get-MLBatchPrediction`.

**Tools for PowerShell**

**Example 1:** Returns the detailed metadata for a batch prediction with id ID.

```powershell
Get-MLBatchPrediction -BatchPredictionId ID
```

- For API details, see [GetBatchPrediction](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-MLBatchPredictionList**

The following code example shows how to use `Get-MLBatchPredictionList`.

**Tools for PowerShell**

**Example 1:** Returns a list of all BatchPredictions and their associated data records that match the search criterion given in the request.

```powershell
Get-MLBatchPredictionList
```

**Example 2:** Returns a list of all BatchPredictions with a status of COMPLETED.
Get-MLBatchPredictionList -FilterVariable Status -EQ COMPLETED

- For API details, see [DescribeBatchPredictions](https://aws.amazon.com/tools-for-powershell-cmdlet-reference/) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-MLDataSource**

The following code example shows how to use Get-MLDataSource.

**Tools for PowerShell**

**Example 1:** Returns the metadata, status, and data file information for a DataSource with the id ID.

```
Get-MLDataSource -DataSourceId ID
```

- For API details, see [GetDataSource](https://aws.amazon.com/tools-for-powershell-cmdlet-reference/) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-MLDataSourceList**

The following code example shows how to use Get-MLDataSourceList.

**Tools for PowerShell**

**Example 1:** Returns a list of all DataSources and their associated data records.

```
Get-MLDataSourceList
```

**Example 2:** Returns a list of all DataSources with a status of COMPLETED.

```
Get-MLDataSourceList -FilterVariable Status -EQ COMPLETED
```

- For API details, see [DescribeDataSources](https://aws.amazon.com/tools-for-powershell-cmdlet-reference/) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-MLEvaluation**

The following code example shows how to use Get-MLEvaluation.

**Tools for PowerShell**

**Example 1:** Returns metadata and status for an Evaluation with id ID.
Get-MLEvaluation -EvaluationId ID

- For API details, see `GetEvaluation` in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-MLEvaluationList**

The following code example shows how to use Get-MLEvaluationList.

**Tools for PowerShell**

**Example 1: Returns a list of all Evaluation resources**

```
Get-MLEvaluationList
```

**Example 2: Returns a list of all Evaluations with a status of COMPLETED.**

```
Get-MLEvaluationList -FilterVariable Status -EQ COMPLETED
```

- For API details, see `DescribeEvaluations` in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-MLModel**

The following code example shows how to use Get-MLModel.

**Tools for PowerShell**

**Example 1: Returns the detail metadata, status, schema, and data file information for a MLModel with id ID.**

```
Get-MLModel -ModelId ID
```

- For API details, see `GetMLModel` in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-MLModelList**

The following code example shows how to use Get-MLModelList.

**Tools for PowerShell**

**Example 1: Returns a list of all Models and their associated data records.**
Get-MLModelList

Example 2: Returns a list of all Models with a status of COMPLETED.

Get-MLModelList -FilterVariable Status -EQ COMPLETED

• For API details, see DescribeMLModels in AWS Tools for PowerShell Cmdlet Reference.

Get-MLPrediction

The following code example shows how to use Get-MLPrediction.

Tools for PowerShell

Example 1: Send a record to the realtime prediction endpoint URL for Model with id ID.

Get-MLPrediction -ModelId ID -PredictEndpoint URL -Record @{"A" = "B"; "C" = "D";}

• For API details, see Predict in AWS Tools for PowerShell Cmdlet Reference.

New-MLBatchPrediction

The following code example shows how to use New-MLBatchPrediction.

Tools for PowerShell

Example 1: Create a new batch prediction request for model with id ID and put the output at the specified S3 location.

New-MLBatchPrediction -ModelId ID -Name NAME -OutputURI s3://...

• For API details, see CreateBatchPrediction in AWS Tools for PowerShell Cmdlet Reference.

New-MLDataSourceFromS3

The following code example shows how to use New-MLDataSourceFromS3.
Tools for PowerShell

Example 1: Create a data source with data for an S3 location, with a name of NAME and a schema of SCHEMA.

```
New-MLDataSourceFromS3 -Name NAME -ComputeStatistics $true -DataSpec_DataLocationS3 "s3://BUCKET/KEY" -DataSchema SCHEMA
```

- For API details, see [CreateDataSourceFromS3](#) in *AWS Tools for PowerShell Cmdlet Reference*.

New-MLEvaluation

The following code example shows how to use New-MLEvaluation.

Tools for PowerShell

Example 1: Create an evaluation for a given data source id and model id

```
New-MLEvaluation -Name NAME -DataSourceId DSID -ModelId MID
```

- For API details, see [CreateEvaluation](#) in *AWS Tools for PowerShell Cmdlet Reference*.

New-MLModel

The following code example shows how to use New-MLModel.

Tools for PowerShell

Example 1: Create a new model with training data.

```
New-MLModel -Name NAME -ModelType BINARY -Parameter @{} -TrainingDataSourceId ID
```

- For API details, see [CreateMLModel](#) in *AWS Tools for PowerShell Cmdlet Reference*.

New-MLRealtimeEndpoint

The following code example shows how to use New-MLRealtimeEndpoint.

Tools for PowerShell

Example 1: Create a new realtime prediction endpoint for the given model id.
Macie examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with Macie.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

- Actions

Actions

Get-MAC2FindingList

The following code example shows how to use Get-MAC2FindingList.

Tools for PowerShell

Example 1: Returns list of FindingIds for Findings containing a sensitive data detection with type "CREDIT_CARD_NUMBER" or "US_SOCIAL_SECURITY_NUMBER"

```powershell
$criterionAddProperties.Eq = @("CREDIT_CARD_NUMBER" "US_SOCIAL_SECURITY_NUMBER")
```
$FindingCriterion = @{
    'classificationDetails.result.sensitiveData.detections.type' = 
}


• For API details, see ListFindings in AWS Tools for PowerShell Cmdlet Reference.

AWS OpsWorks examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with AWS OpsWorks.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics
• Actions

Actions

New-OPSDeployment

The following code example shows how to use New-OPSDeployment.

Tools for PowerShell

Example 1: This command creates a new app deployment on all of the Linux-based instances in a layer in AWS OpsWorks Stacks. Even if you specify a layer ID, you must specify a stack ID, too. The command lets the deployment restart the instances if required.
Example 2: This command deploys the appsetup recipe from the phpapp cookbook, and the secbaseline recipe from the testcookbook cookbook. The deployment target is one instance, but the stack ID and layer ID are also required. The Command_Arg parameter allow_reboot attribute is set to true, which lets the deployment restart the instances if required.

```powershell
$commandArgs = '{ "Name":"execute_recipes", "Args":{ "recipes": ["phpapp::appsetup","testcookbook::secbaseline"] } }
New-OPSDeployment -StackID "724z93zz-zz78-4zzz-8z9z-1290123zzz1z" -LayerId "511b99c5-ec78-4caa-8a9d-1440116ffd1b" -AppId "0f7a109c-bf68-4336-8cb9-d37fe0b8c61d" -Command_Name deploy -Command_Arg @{Name="allow_reboot";Value="true"}
```

For API details, see CreateDeployment in AWS Tools for PowerShell Cmdlet Reference.

### AWS Price List examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with AWS Price List.

**Actions** are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

**Scenarios** are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

**Topics**

- **Actions**
Actions

**Get-PLSAttributeValue**

The following code example shows how to use `Get-PLSAttributeValue`.

**Tools for PowerShell**

**Example 1:** Returns the values for the attribute 'volumeType' for Amazon EC2 in the us-east-1 region.

```powershell
Get-PLSAttributeValue -ServiceCode AmazonEC2 -AttributeName "volumeType" -region us-east-1
```

**Output:**

Value
-----
Cold HDD
General Purpose
Magnetic
Provisioned IOPS
Throughput Optimized HDD

- For API details, see [GetAttributeValues](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-PLSPProduct**

The following code example shows how to use `Get-PLSPProduct`.

**Tools for PowerShell**

**Example 1:** Returns details of all products for Amazon EC2.

```powershell
Get-PLSPProduct -ServiceCode AmazonEC2 -Region us-east-1
```

**Output:**

```json
{"product":{"productFamily":"Compute Instance","attributes":
{"enhancedNetworkingSupported":"Yes","memory":"30.5
```
Example 2: Returns data for Amazon EC2 in the us-east-1 region filtered by volume types of 'General Purpose' that are SSD-backed.

```
Get-PLSProduct -ServiceCode AmazonEC2 -Filter @{{Type="TERM_MATCH";Field="volumeType";Value="General Purpose"},@{{Type="TERM_MATCH";Field="storageMedia";Value="SSD-backed"}} -Region us-east-1
```

Output:

```
{"product":{"productFamily":"Storage","attributes":{"storageMedia":"SSD-backed","maxThroughputvolume":"160 MB/sec","volumeType":"General Purpose","maxIopsvolume":"10000"},...
```

- For API details, see [GetProducts](https://aws.amazon.com) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-PLSService**

The following code example shows how to use Get-PLSService.

**Tools for PowerShell**

**Example 1: Returns the metadata for all available service codes in the us-east-1 region.**

```
Get-PLSService -Region us-east-1
```

Output:

```
AttributeNames  ServiceCode
AWS Price List  643
```
Example 2: Returns the metadata for the Amazon EC2 service in the us-east-1 region.

```powershell
Get-PLSService -ServiceCode AmazonEC2 -Region us-east-1
```

Output:

```
AttributeNames         ServiceCode
----------------------  -----------
{volumeType, maxIopsvolume, instanceCapacity10xlarge, locationType...} AmazonEC2
```

- For API details, see [DescribeServices](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Resource Groups examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with Resource Groups.

**Actions** are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

**Scenarios** are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

**Topics**
- **Actions**
Actions

Add-RGResourceTag

The following code example shows how to use Add-RGResourceTag.

Example 1: This example adds tag key 'Instances' with value 'workboxes' to the given resource group arn

```
Add-RGResourceTag -Tag @{$Instances="workboxes"} -Arn arn:aws:resource-groups:eu-west-1:123456789012:group/workboxes
```

Output:

```
Arn                                         Tags
---                                         ----
arn:aws:resource-groups:eu-west-1:123456789012:group/workboxes {[Instances, workboxes]}
```

• For API details, see Tag in AWS Tools for PowerShell Cmdlet Reference.

Find-RGResource

The following code example shows how to use Find-RGResource.

Example 1: This example creates a ResourceQuery for Instance resource types with tag filters and finds resources.

```
$query.Type = [Amazon.ResourceGroups.QueryType]::TAG_FILTERS_1_0
$query.Query = ConvertTo-Json -Compress -Depth 4 -InputObject @{
    ResourceTypeFilters = @('AWS::EC2::Instance')
    TagFilters = @(@{
        Key = 'auto'
        Values = @('no')
    })
}
```
Find-RGResource -ResourceQuery $query | Select-Object -ExpandProperty ResourceIdentifiers

Output:

<table>
<thead>
<tr>
<th>ResourceArn</th>
<th>ResourceType</th>
</tr>
</thead>
<tbody>
<tr>
<td>arn:aws:ec2:eu-west-1:123456789012:instance/i-0123445b6cb7bd67b</td>
<td>AWS::EC2::Instance</td>
</tr>
</tbody>
</table>

- For API details, see [SearchResources](#) in **AWS Tools for PowerShell Cmdlet Reference**.

Get-RGGroup

The following code example shows how to use Get-RGGroup.

**Tools for PowerShell**

**Example 1: This example retrieves resource group as per the group name**

Get-RGGroup -GroupName auto-no

Output:

<table>
<thead>
<tr>
<th>Description</th>
<th>GroupArn</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>arn:aws:resource-groups:eu-west-1:123456789012:group/auto-no</td>
<td>auto-no</td>
</tr>
</tbody>
</table>

- For API details, see [GetGroup](#) in **AWS Tools for PowerShell Cmdlet Reference**.

Get-RGGroupList

The following code example shows how to use Get-RGGroupList.

**Tools for PowerShell**

**Example 1: This example lists resource group already created.**

Get-RGGroupList

Output:
For API details, see [ListGroups](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-RGGroupQuery

The following code example shows how to use Get-RGGroupQuery.

**Tools for PowerShell**

**Example 1:** This example fetches the resource query for the given resource group

```
Get-RGGroupQuery -GroupName auto-no | Select-Object -ExpandProperty ResourceQuery
```

**Output:**

```
Query     Type
-----     ----
{"ResourceTypeFilters":["AWS::EC2::Instance"],"TagFilters":[{"Key":"auto","Values":
["no"]}]}
```

For API details, see [GetGroupQuery](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-RGGroupResourceList

The following code example shows how to use Get-RGGroupResourceList.

**Tools for PowerShell**

**Example 1:** This example lists group resources on the basis of filtered by resource type

```
Get-RGGroupResourceList -Filter @{Name="resource-type";Values="AWS::EC2::Instance"}
-GroupName auto-yes | Select-Object -ExpandProperty ResourceIdentifiers
```

**Output:**
<table>
<thead>
<tr>
<th>ResourceArn</th>
<th>ResourceType</th>
</tr>
</thead>
<tbody>
<tr>
<td>arn:aws:ec2:eu-west-1:123456789012:instance/i-0123bc45b567890e1</td>
<td>AWS::EC2::Instance</td>
</tr>
<tr>
<td>arn:aws:ec2:eu-west-1:123456789012:instance/i-0a1caf2345f67d8dc</td>
<td>AWS::EC2::Instance</td>
</tr>
<tr>
<td>arn:aws:ec2:eu-west-1:123456789012:instance/i-012e3cb4df567e8aa</td>
<td>AWS::EC2::Instance</td>
</tr>
<tr>
<td>arn:aws:ec2:eu-west-1:123456789012:instance/i-0fd12dd3456789012</td>
<td>AWS::EC2::Instance</td>
</tr>
</tbody>
</table>

- For API details, see [ListGroupResources](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-RGResourceTag

The following code example shows how to use `Get-RGResourceTag`.

**Tools for PowerShell**

**Example 1:** This example lists tags for the given resource group arn

```powershell
```

**Output:**

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instances</td>
<td>workboxes</td>
</tr>
</tbody>
</table>

- For API details, see [GetTags](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### New-RGGroup

The following code example shows how to use `New-RGGroup`.

**Tools for PowerShell**

**Example 1:** This example creates a new tag-based AWS Resource Groups resource group named `TestPowerShellGroup`. The group includes Amazon EC2 instances in the current region that are tagged with the tag key "Name", and tag value "test2". The command returns the query and type of group, and the results of the operation.

```powershell
$ResourceQuery = New-Object -TypeName Amazon.ResourceGroups.Model.ResourceQuery
```
$ResourceQuery.Type = "TAG_FILTERS_1_0"
$ResourceQuery.Query = '{"ResourceTypeFilters": ["AWS::EC2::Instance"], "TagFilters": [{"Key": "Name", "Values": ["test2"]}]}
$ResourceQuery

New-RGGroup -Name TestPowerShellGroup -ResourceQuery $ResourceQuery -Description "Test resource group."

Output:

<table>
<thead>
<tr>
<th>Query</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ResourceTypeFilters&quot;: [&quot;AWS::EC2::Instance&quot;], &quot;TagFilters&quot;: [{&quot;Key&quot;: &quot;Name&quot;, &quot;Values&quot;: [&quot;test2&quot;]}]</td>
<td>TAG_FILTERS_1_0</td>
</tr>
</tbody>
</table>

LoggedAt   : 11/20/2018 2:40:59 PM
Tags       : {}
ResponseMetadata : Amazon.Runtime.ResponseMetadata
ContentLength : 338
HttpStatusCode : OK

- For API details, see [CreateGroup](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-RGGroup**

The following code example shows how to use Remove-RGGroup.

**Tools for PowerShell**

**Example 1: This example removes the named resource group**

Remove-RGGroup -GroupName non-tag-cfn-elbv2

**Output:**

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-RGGroup (DeleteGroup)" on target "non-tag-cfn-elbv2".

[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): Y

Description GroupArn
Name
----------- --------
----

- For API details, see [DeleteGroup](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Remove-RGResourceTag

The following code example shows how to use Remove-RGResourceTag.

#### Tools for PowerShell

**Example 1:** This example removes mentioned tag from the resource group

```
Remove-RGResourceTag -Arn arn:aws:resource-groups:eu-west-1:123456789012:group/workboxes -Key Instances
```

**Output:**

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-RGResourceTag (Untag)" on target "arn:aws:resource-groups:eu-west-1:933303704102:group/workboxes".

[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): Y

Arn                      Keys
---                      ----

arn:aws:resource-groups:eu-west-1:123456789012:group/workboxes {Instances}

- For API details, see [Untag](#) in *AWS Tools for PowerShell Cmdlet Reference*.
**Update-RGGroup**

The following code example shows how to use Update-RGGroup.

**Tools for PowerShell**

**Example 1: This example updates the description of the group**

```
Update-RGGroup -GroupName auto-yes -Description "Instances auto-remove"
```

**Output:**

<table>
<thead>
<tr>
<th>Description</th>
<th>GroupArn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>----</td>
</tr>
<tr>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>Instances to be cleaned</td>
<td>arn:aws:resource-groups:eu-west-1:123456789012:group/auto-yes auto-yes</td>
</tr>
</tbody>
</table>

- For API details, see [UpdateGroup](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Update-RGGroupQuery**

The following code example shows how to use Update-RGGroupQuery.

**Tools for PowerShell**

**Example 1: This example creates a query object and updates the query for the group.**

```
$query.Type = [Amazon.ResourceGroups.QueryType]::TAG_FILTERS_1_0
$query.Query = @{
    ResourceTypeFilters = @('AWS::EC2::Instance')
    TagFilters = @@{
        Key='Environment'
        Values='Build600.11'
    )
}
| ConvertTo-Json -Compress -Depth 4
```

**Output:**

Resource Groups

651
Resource Groups Tagging API examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with Resource Groups Tagging API.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics
• Actions

Actions

Add-RGTResourceTag

The following code example shows how to use Add-RGTResourceTag.

Tools for PowerShell

Example 1: This example adds the tag keys "stage" and "version" with values "beta" and "preprod_test" to an Amazon S3 bucket and an Amazon DynamoDB table. A single call is made to the service to apply the tags.

```powershell
$arn1 = "arn:aws:s3:::mybucket"
```
Add-RGTResourceTag -ResourceARNList $arn1,$arn2 -Tag @{ "stage"="beta"; "version"="preprod_test" }

Example 2: This example adds the specified tags and values to an Amazon S3 bucket and an Amazon DynamoDB table. Two calls are made to the service, one for each resource ARN piped into the cmdlet.

$arn1 = "arn:aws:s3:::mybucket"

$arn1,$arn2 | Add-RGTResourceTag -Tag @{ "stage"="beta"; "version"="preprod_test" }

- For API details, see TagResources in AWS Tools for PowerShell Cmdlet Reference.

Get-RGTResource

The following code example shows how to use Get-RGTResource.

Tools for PowerShell

Example 1: Returns all the tagged resources in a region and the tag keys associated with the resource. If no -Region parameter is supplied to the cmdlet it will attempt to infer region from the shell or EC2 instance metadata.

Get-RGTResource

Output:

<table>
<thead>
<tr>
<th>ResourceARN</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>arn:aws:dynamodb:us-west-2:123456789012:table/mytable</td>
<td>{stage, version}</td>
</tr>
<tr>
<td>arn:aws:s3:::mybucket</td>
<td>{stage, version, othertag}</td>
</tr>
</tbody>
</table>

Example 2: Returns all the tagged resources of the specified type in a region. The string for each service name and resource type is the same as that embedded in a resource’s Amazon Resource Name (ARN).

Get-RGTResource -ResourceType "s3"
Output:

<table>
<thead>
<tr>
<th>ResourceARN</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>arn:aws:s3:::mybucket</td>
<td>{stage, version, othertag}</td>
</tr>
</tbody>
</table>

Example 3: Returns all the tagged resources of the specified type in a region. Note that when the resource types are piped into the cmdlet, one call to the service is made for each supplied resource type.

"dynamodb","s3" | Get-RGTResource

Output:

<table>
<thead>
<tr>
<th>ResourceARN</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>arn:aws:dynamodb:us-west-2:123456789012:table/mytable</td>
<td>{stage, version}</td>
</tr>
<tr>
<td>arn:aws:s3:::mybucket</td>
<td>{stage, version, othertag}</td>
</tr>
</tbody>
</table>

Example 4: Returns all the tagged resources that match the specified filter.

Get-RGTResource -TagFilter @{ Key="stage" }  

Output:

<table>
<thead>
<tr>
<th>ResourceARN</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>arn:aws:s3:::mybucket</td>
<td>{stage, version, othertag}</td>
</tr>
</tbody>
</table>

Example 5: Returns all the tagged resources that match the specified filter and resource type.

Get-RGTResource -TagFilter @{ Key="stage" } -ResourceType "dynamodb"

Output:
Example 6: Returns all the tagged resources that match the specified filter.

Get-RGTResource -TagFilter @{ Key="stage"; Values=(@("beta","gamma") ) }

Output:

<table>
<thead>
<tr>
<th>ResourceARN</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>arn:aws:dynamodb:us-west-2:123456789012:table/mytable</td>
<td>{stage, version}</td>
</tr>
</tbody>
</table>

• For API details, see [GetResources](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-RGTagKey

The following code example shows how to use Get-RGTagKey.

Tools for PowerShell

Example 1: Returns all tag keys in the specified region. If the -Region parameter is not specified the cmdlet will attempt to infer the region from the default shell region or EC2 instance metadata. Note that the tag keys are not returned in any specific order.

Get-RGTagKey -region us-west-2

Output:

<table>
<thead>
<tr>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>stage</td>
</tr>
</tbody>
</table>

• For API details, see [GetTagKeys](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-RGTagValue

The following code example shows how to use Get-RGTagValue.
Tools for PowerShell

Example 1: Returns the value for the specified tag in a region. If the -Region parameter is not specified the cmdlet will attempt to infer the region from the default shell region or EC2 instance metadata.

```
Get-RGTTagValue -Key "stage" -Region us-west-2
```

Output:

```
beta
```

- For API details, see [GetTagValues](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

Remove-RGTResourceTag

The following code example shows how to use Remove-RGTResourceTag.

Tools for PowerShell

Example 1: Removes the tag keys "stage" and "version", and the associated values, from an Amazon S3 bucket and an Amazon DynamoDB table. A single call is made to the service to remove the tags. Before the tags are removed the cmdlet will prompt for confirmation. To bypass confirmation add the -Force parameter.

```
$arn1 = "arn:aws:s3:::mybucket"
Remove-RGTResourceTag -ResourceARNList $arn1,$arn2 -TagKey "stage","version"
```

Example 2: Removes the tag keys "stage" and "version", and the associated values, from an Amazon S3 bucket and an Amazon DynamoDB table. Two calls are made to the service, one for each resource ARN piped into the cmdlet. Before each call the cmdlet will prompt for confirmation. To bypass confirmation add the -Force parameter.

```
$arn1 = "arn:aws:s3:::mybucket"

$arn1,$arn2 | Remove-RGTResourceTag -TagKey "stage","version"
```
Route 53 examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with Route 53.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics
• Actions

Actions

Edit-R53ResourceRecordSet

The following code example shows how to use Edit-R53ResourceRecordSet.

Tools for PowerShell

Example 1: This example creates an A record for www.example.com and changes the A record for test.example.com from 192.0.2.3 to 192.0.2.1. Note that values for changes TXT-type records must be in double quotes. See the Amazon Route 53 documentation for more details. You can use the Get-R53Change cmdlet to poll to determine when the changes are complete.

```powershell
$change1 = New-Object Amazon.Route53.Model.Change
$change1.Action = "CREATE"
$change1.ResourceRecordSet = New-Object Amazon.Route53.Model.ResourceRecordSet
$change1.ResourceRecordSet.Name = "www.example.com"
$change1.ResourceRecordSet.Type = "TXT"
```
$change1.ResourceRecordSet.TTL = 600
$change1.ResourceRecordSet.ResourceRecords.Add(@{Value="item 1 item 2 item 3"})

$change2 = New-Object Amazon.Route53.Model.Change
$change2.Action = "DELETE"
$change2.ResourceRecordSet = New-Object Amazon.Route53.Model.ResourceRecordSet
$change2.ResourceRecordSet.Name = "test.example.com"
$change2.ResourceRecordSet.Type = "A"
$change2.ResourceRecordSet.TTL = 600
$change2.ResourceRecordSet.ResourceRecords.Add(@{Value="192.0.2.3"})

$change3 = New-Object Amazon.Route53.Model.Change
$change3.Action = "CREATE"
$change3.ResourceRecordSet.Name = "test.example.com"
$change3.ResourceRecordSet.Type = "A"
$change3.ResourceRecordSet.TTL = 600
$change3.ResourceRecordSet.ResourceRecords.Add(@{Value="192.0.2.1"})

$params = @{
    HostedZoneId="Z1PA6795UKMFR9"
    ChangeBatch_Comment="This change batch creates a TXT record for www.example.com and changes the A record for test.example.com. from 192.0.2.3 to 192.0.2.1."
    ChangeBatch_Change=$change1,$change2,$change3
}

Edit-R53ResourceRecordSet @params

Example 2: This example shows how to create alias resource record sets. 'Z2222222222' is the ID of the Amazon Route 53 hosted zone in which you're creating the alias resource record set. 'example.com' is the zone apex for which you want to create an alias and 'www.example.com' is a subdomain for which you also want to create an alias. 'Z11111111111111' is an example of a hosted zone ID for the load balancer and 'example-load-balancer-1111111111.us-east-1.elb.amazonaws.com' is an example of a load balancer domain name with which Amazon Route 53 responds to queries for example.com and www.example.com. See the Amazon Route 53 documentation for more details. You can use the Get-R53Change cmdlet to poll to determine when the changes are complete.
Example 3: This example creates two A records for www.example.com. One-fourth of the time \((1/(1+3))\), Amazon Route 53 responds to queries for www.example.com with the two values for the first resource record set (192.0.2.9 and 192.0.2.10). Three-fourths of the time \((3/(1+3))\) Amazon Route 53 responds to queries for www.example.com with the two values for the second resource record set (192.0.2.11 and 192.0.2.12). See the Amazon Route 53 documentation for more details. You can use the Get-R53Change cmdlet to poll to determine when the changes are complete.

```powershell
$change1 = New-Object Amazon.Route53.Model.Change
$change1.Action = "CREATE"
$change1.ResourceRecordSet = New-Object Amazon.Route53.Model.ResourceRecordSet
$change1.ResourceRecordSet.Name = "www.example.com"
$change1.ResourceRecordSet.Type = "A"
$change1.ResourceRecordSet.SetIdentifier = "Rack 2, Positions 4 and 5"

$change2 = New-Object Amazon.Route53.Model.Change
$change2.Action = "CREATE"
$change2.ResourceRecordSet = New-Object Amazon.Route53.Model.ResourceRecordSet
$change1.ResourceRecordSet.Name = "www.example.com"
$change1.ResourceRecordSet.Type = "A"
$change1.ResourceRecordSet.SetIdentifier = "Rack 2, Positions 4 and 5"
```

Example 4: This example shows how to create weighted alias resource record sets assuming that example.com is the domain for which you want to create weighted alias resource record sets. SetIdentifier differentiates the two weighted alias resource record sets from one another. This element is required because the Name and Type elements have the same values for both resource record sets. Z1111111111111 and Z3333333333333 are examples of hosted zone IDs for the ELB load balancer specified by the value of DNSName. example-load-balancer-2222222222.us-east-1.elb.amazonaws.com and example-load-balancer-4444444444.us-east-1.elb.amazonaws.com are examples of Elastic Load Balancing domains from which Amazon Route 53 responds to queries for example.com. See the Amazon Route 53 documentation for more details. You can use the Get-R53Change cmdlet to poll to determine when the changes are complete.
$change1.ResourceRecordSet.Type = "A"
$change1.ResourceRecordSet.SetIdentifier = "1"
$change1.ResourceRecordSet.Weight = 3
$change1.ResourceRecordSet.AliasTarget.HostedZoneId = "Z1111111111111"
$change1.ResourceRecordSet.AliasTarget.DNSName = "example-load-balancer-2222222222.us-east-1.elb.amazonaws.com."
$change1.ResourceRecordSet.AliasTarget.EvaluateTargetHealth = $true

$change2 = New-Object Amazon.Route53.Model.Change
$change2.Action = "CREATE"
$change2.ResourceRecordSet = New-Object Amazon.Route53.Model.ResourceRecordSet
$change2.ResourceRecordSet.Name = "example.com"
$change2.ResourceRecordSet.Type = "A"
$change2.ResourceRecordSet.SetIdentifier = "2"
$change2.ResourceRecordSet.Weight = 1
$change2.ResourceRecordSet.AliasTarget.HostedZoneId = "Z3333333333333"
$change2.ResourceRecordSet.AliasTarget.DNSName = "example-load-balancer-4444444444.us-east-1.elb.amazonaws.com."
$change2.ResourceRecordSet.AliasTarget.EvaluateTargetHealth = $false

$params = @{
    HostedZoneId="Z55555555555"
    ChangeBatch_Comment="This change batch creates two weighted alias resource record sets. Amazon Route 53 responds to queries for example.com with the first ELB domain 3/4ths of the times and the second one 1/4th of the time."
    ChangeBatch_Change=$change1,$change2
}

Edit-R53ResourceRecordSet @params

Example 5: This example creates two latency alias resource record sets, one for an ELB load balancer in the US West (Oregon) region (us-west-2), and another for a load balancer in the Asia Pacific (Singapore) region (ap-southeast-1). See the Amazon Route 53 documentation for more details. You can use the Get-R53Change cmdlet to poll to determine when the changes are complete.

$change1 = New-Object Amazon.Route53.Model.Change
$change1.Action = "CREATE"
$change1.ResourceRecordSet = New-Object Amazon.Route53.Model.ResourceRecordSet
$change1.ResourceRecordSet.Name = "example.com"
$change1.ResourceRecordSet.Type = "A"
$change1.ResourceRecordSet.SetIdentifier = "Oregon load balancer 1"
$change1.ResourceRecordSet.Region = us-west-2
$change1.ResourceRecordSet.AliasTarget.HostedZoneId = "Z1111111111111"
$change1.ResourceRecordSet.AliasTarget.DNSName = "example-load-balancer-2222222222.us-west-2.elb.amazonaws.com"
$change1.ResourceRecordSet.AliasTarget.EvaluateTargetHealth = $true

$change2 = New-Object Amazon.Route53.Model.Change
$change2.Action = "CREATE"
$change2.ResourceRecordSet = New-Object Amazon.Route53.Model.ResourceRecordSet
$change2.ResourceRecordSet.Name = "example.com"
$change2.ResourceRecordSet.Type = "A"
$change2.ResourceRecordSet.SetIdentifier = "Singapore load balancer 1"
$change2.ResourceRecordSet.Region = ap-southeast-1
$change2.ResourceRecordSet.AliasTarget.HostedZoneId = "Z2222222222222"
$change2.ResourceRecordSet.AliasTarget.DNSName = "example-load-balancer-1111111111.ap-southeast-1.elb.amazonaws.com"
$change2.ResourceRecordSet.AliasTarget.EvaluateTargetHealth = $true

$params = @{
    HostedZoneId="Z5555555555"
    ChangeBatch_Comment="This change batch creates two latency resource record sets, one for the US West (Oregon) region and one for the Asia Pacific (Singapore) region."
    ChangeBatch_Change=$change1,$change2
}

Edit-R53ResourceRecordSet @params

- For API details, see ChangeResourceRecordSets in AWS Tools for PowerShell Cmdlet Reference.

Get-R53AccountLimit

The following code example shows how to use Get-R53AccountLimit.

Tools for PowerShell

Example 1: This example returns the maximum number of hosted zones that can be created using the current account.
Get-R53AccountLimit -Type MAX_HOSTED_ZONES_BY_OWNER

Output:

15

- For API details, see GetAccountLimit in AWS Tools for PowerShell Cmdlet Reference.

Get-R53CheckerIpRanges

The following code example shows how to use Get-R53CheckerIpRanges.

Tools for PowerShell

Example 1: This example returns the CIDRs for the Route53 health checkers

Get-R53CheckerIpRanges

Output:

15.177.2.0/23
15.177.6.0/23
15.177.10.0/23
15.177.14.0/23
15.177.18.0/23
15.177.22.0/23
15.177.26.0/23
15.177.30.0/23
15.177.34.0/23
15.177.38.0/23
15.177.42.0/23
15.177.46.0/23
15.177.50.0/23
15.177.54.0/23
15.177.58.0/23
15.177.62.0/23
54.183.255.128/26
54.228.16.0/26
54.232.40.64/26
54.241.32.64/26
• For API details, see GetCheckerIpRanges in AWS Tools for PowerShell Cmdlet Reference.

Get-R53HostedZone

The following code example shows how to use Get-R53HostedZone.

Tools for PowerShell

Example 1: Returns details of the hosted zone with ID Z1D633PJN98FT9.

Get-R53HostedZone -Id Z1D633PJN98FT9

• For API details, see GetHostedZone in AWS Tools for PowerShell Cmdlet Reference.

Get-R53HostedZoneCount

The following code example shows how to use Get-R53HostedZoneCount.

Tools for PowerShell

Example 1: Returns the total number of public and private hosted zones for the current AWS account.

Get-R53HostedZoneCount

• For API details, see GetHostedZoneCount in AWS Tools for PowerShell Cmdlet Reference.
Get-R53HostedZoneLimit

The following code example shows how to use Get-R53HostedZoneLimit.

Tools for PowerShell

Example 1: This example returns the limit on the maximum number of records that can be created in the specified hosted zone.

```
Get-R53HostedZoneLimit -HostedZoneId Z3MEQ8T7HAAAAF -Type MAX_RRSETS_BY_ZONE
```

Output:

```
5
```

- For API details, see GetHostedZoneLimit in AWS Tools for PowerShell Cmdlet Reference.

Get-R53HostedZoneList

The following code example shows how to use Get-R53HostedZoneList.

Tools for PowerShell

Example 1: Outputs all of your public and private hosted zones.

```
Get-R53HostedZoneList
```

Example 2: Outputs all of the hosted zones that are associated with the reusable delegation set that has the ID NZ8X2CISAMPLE

```
Get-R53HostedZoneList -DelegationSetId NZ8X2CISAMPLE
```

- For API details, see ListHostedZones in AWS Tools for PowerShell Cmdlet Reference.

Get-R53HostedZonesByName

The following code example shows how to use Get-R53HostedZonesByName.
Tools for PowerShell

Example 1: Returns all of your public and private hosted zones in ASCII order by domain name.

```
Get-R53HostedZonesByName
```

Example 2: Returns your public and private hosted zones, in ASCII order by domain name, starting at the specified DNS name.

```
Get-R53HostedZonesByName -DnsName example2.com
```

Example 3: This example shows how to manually enumerate the hosted zones by first retrieving a single item and then iterating two at a time until all zones have been returned, using marker properties attached to the service response in the $AWSHistory stack after each call.

```
Get-R53HostedZonesByName -MaxItem 1
while ($LastServiceResponse.IsTruncated)
{
    $nextPageParams = @{
        DnsName=$LastServiceResponse.NextDNSName
        HostedZoneId=$LastServiceResponse.NextHostedZoneId
    }
    Get-R53HostedZonesByName -MaxItem 2 @nextPageParams
}
```

- For API details, see Get-R53QueryLoggingConfigList in AWS Tools for PowerShell Cmdlet Reference.

Get-R53QueryLoggingConfigList

The following code example shows how to use Get-R53QueryLoggingConfigList.

Tools for PowerShell

Example 1: This example returns all the configurations for DNS query logging that are associated with the current AWS account.

```
Get-R53QueryLoggingConfigList
```
Output:

<table>
<thead>
<tr>
<th>Id</th>
<th>HostedZoneId</th>
<th>CloudWatchLogsLogGroupArn</th>
</tr>
</thead>
<tbody>
<tr>
<td>59b0fa33-4fe9-4471-a88c-926476aaa4d</td>
<td>Z385PDS6EAAAZR</td>
<td>arn:aws:logs:us-east-1:111111111112:log-group:/aws/route53/example1.com:*</td>
</tr>
<tr>
<td>ee528e95-4e03-4fde-9d28-9e24ddaaa063</td>
<td>Z94SJHBV1AAAAZ</td>
<td>arn:aws:logs:us-east-1:111111111112:log-group:/aws/route53/example2.com:*</td>
</tr>
<tr>
<td>e38ddda-ceb6-45c1-8cb7-f0ae56aaaa2b</td>
<td>Z3MEQ8T7AAABF</td>
<td>arn:aws:logs:us-east-1:111111111112:log-group:/aws/route53/example3.com:*</td>
</tr>
</tbody>
</table>

• For API details, see ListQueryLoggingConfigs in AWS Tools for PowerShell Cmdlet Reference.

Get-R53ReusableDelegationSet

The following code example shows how to use Get-R53ReusableDelegationSet.

Tools for PowerShell

Example 1: This example retrieves information about the specified delegation set including the four name servers that are assigned to the delegation set.

Get-R53ReusableDelegationSet -Id N23DS9X4AYEAAA

Output:

<table>
<thead>
<tr>
<th>Id</th>
<th>CallerReference</th>
<th>NameServers</th>
</tr>
</thead>
<tbody>
<tr>
<td>/delegationset/N23DS9X4AYEAAA testcaller</td>
<td>{ns-545.awsdns-04.net, ns-1264.awsdns-30.org, ns-2004.awsdns-58.co.uk, ns-240.awsdns-30.com}</td>
<td></td>
</tr>
</tbody>
</table>

• For API details, see GetReusableDelegationSet in AWS Tools for PowerShell Cmdlet Reference.

New-R53HostedZone

The following code example shows how to use New-R53HostedZone.

Tools for PowerShell

Example 1: Creates a new hosted zone named 'example.com', associated with a reusable delegation set. Note that you must supply a value for the CallerReference parameter so that
requests that need to be retried if necessary without the risk of executing the operation twice. Because the hosted zone is being created in a VPC it is automatically private and you should not set the `-HostedZoneConfig_PrivateZone` parameter.

```powershell
$params = @{
    Name="example.com"
    CallerReference="myUniqueIdentifier"
    HostedZoneConfig_Comment="This is my first hosted zone"
    DelegationSetId="NZ8X2CISAMPLE"
    VPC_VPCId="vpc-1a2b3c4d"
    VPC_VPCRegion="us-east-1"
}
New-R53HostedZone @params
```

- For API details, see [CreateHostedZone](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### New-R53QueryLoggingConfig

The following code example shows how to use `New-R53QueryLoggingConfig`.

#### Tools for PowerShell

**Example 1**: This example creates a new Route53 DNS query logging configuration for the specified hosted zone. Amazon Route53 will publish DNS query logs to the specified Cloudwatch log group.

```powershell
```

**Output:**

```
<table>
<thead>
<tr>
<th>QueryLoggingConfig</th>
<th>Location</th>
</tr>
</thead>
</table>
```

- For API details, see [CreateQueryLoggingConfig](#) in *AWS Tools for PowerShell Cmdlet Reference*.
New-R53ReusableDelegationSet

The following code example shows how to use New-R53ReusableDelegationSet.

**Tools for PowerShell**

**Example 1:** This example creates a reusable delegation set of 4 name servers that can be resused by multiple hosted zones.

```
New-R53ReusableDelegationSet -CallerReference testcallerreference
```

**Output:**

```
<table>
<thead>
<tr>
<th>DelegationSet</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>delegationset/N23DS9XAAAAAXM</td>
</tr>
</tbody>
</table>
```

- For API details, see [CreateReusableDelegationSet](https://aws.amazon.com) in *AWS Tools for PowerShell Cmdlet Reference*.

Register-R53VPCWithHostedZone

The following code example shows how to use Register-R53VPCWithHostedZone.

**Tools for PowerShell**

**Example 1:** This example associates the specified VPC with the private hosted zone.

```
Register-R53VPCWithHostedZone -HostedZoneId Z3MEQ8T7HAAAAAF -VPC_VPCId vpc-f1b9aaaaa -VPC_VPCRegion us-east-1
```

**Output:**

```
<table>
<thead>
<tr>
<th>Id</th>
<th>Status</th>
<th>SubmittedAt</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>/change/C3SCAAA633Z6DX</td>
<td>PENDING</td>
<td>01/28/2020 19:32:02</td>
<td></td>
</tr>
</tbody>
</table>
```

- For API details, see [AssociateVPCWithHostedZone](https://aws.amazon.com) in *AWS Tools for PowerShell Cmdlet Reference*.
Remove-R53HostedZone

The following code example shows how to use Remove-R53HostedZone.

**Tools for PowerShell**

**Example 1:** Deletes the hosted zone with the specified ID. You will be prompted for confirmation before the command proceeds unless you add the -Force switch parameter.

```
Remove-R53HostedZone -Id Z1PA6795UKMFR9
```

- For API details, see [DeleteHostedZone](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Remove-R53QueryLoggingConfig

The following code example shows how to use Remove-R53QueryLoggingConfig.

**Tools for PowerShell**

**Example 1:** This example removes the specified configuration for DNS query logging.

```
Remove-R53QueryLoggingConfig -Id ee528e95-4e03-4fdc-9d28-9e24daaa20063
```

- For API details, see [DeleteQueryLoggingConfig](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Remove-R53ReusableDelegationSet

The following code example shows how to use Remove-R53ReusableDelegationSet.

**Tools for PowerShell**

**Example 1:** This example deletes the specified reusable delegation set.

```
Remove-R53ReusableDelegationSet -Id N23DS9X4AYAAAM
```

- For API details, see [DeleteReusableDelegationSet](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Unregister-R53VPCFromHostedZone

The following code example shows how to use Unregister-R53VPCFromHostedZone.
Tools for PowerShell

Example 1: This example disassociates the specified VPC from the private hosted zone.

```
Unregister-R53VPCFromHostedZone -HostedZoneId Z3MEQ8T7HAAAAAF -VPC_VPCId vpc-f1b9aaaa -VPC_VPCRegion us-east-1
```

Output:

```
<table>
<thead>
<tr>
<th>Id</th>
<th>Status</th>
<th>SubmittedAt</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>/change/C2XFCAAAA9HKZG</td>
<td>PENDING</td>
<td>01/28/2020 10:35:55</td>
<td></td>
</tr>
</tbody>
</table>
```

- For API details, see [DisassociateVPCFromHostedZone](https://docs.aws.amazon.com/tools/latest/dwCmdlets/DisassociateVPCFromHostedZone) in *AWS Tools for PowerShell Cmdlet Reference*.

Update-R53HostedZoneComment

The following code example shows how to use `Update-R53HostedZoneComment`.

Tools for PowerShell

Example 1: This command updates the comment for the specified hosted zone.

```
Update-R53HostedZoneComment -Id Z385PDS6AAAAAR -Comment "This is my first hosted zone"
```

Output:

```
Id                     : /hostedzone/Z385PDS6AAAAAR
Name                   : example.com.
CallerReference        : 5B5555-7147-EF04-8341-69131E805C89
Config                 : Amazon.Route53.Model.HostedZoneConfig
ResourceRecordSetCount : 9
LinkedService          :
```

- For API details, see [UpdateHostedZoneComment](https://docs.aws.amazon.com/tools/latest/dwCmdlets/UpdateHostedZoneComment) in *AWS Tools for PowerShell Cmdlet Reference*.
Amazon S3 examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with Amazon S3.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

- Actions

Actions

Copy-S3Object

The following code example shows how to use Copy-S3Object.

Tools for PowerShell

Example 1: This command copies the object "sample.txt" from bucket "test-files" to the same bucket but with a new key of "sample-copy.txt".

Copy-S3Object -BucketName test-files -Key sample.txt -DestinationKey sample-copy.txt

Example 2: This command copies the object "sample.txt" from bucket "test-files" to the bucket "backup-files" with a key of "sample-copy.txt".

Copy-S3Object -BucketName test-files -Key sample.txt -DestinationKey sample-copy.txt
   -DestinationBucket backup-files

Example 3: This command downloads the object "sample.txt" from bucket "test-files" to a local file with name "local-sample.txt".
Example 4: Downloads the single object to the specified file. The downloaded file will be found at c:\downloads\data\archive.zip

Example 5: Downloads all objects that match the specified key prefix to the local folder. The relative key hierarchy will be preserved as subfolders in the overall download location.

• For API details, see CopyObject in AWS Tools for PowerShell Cmdlet Reference.

Get-S3ACL

The following code example shows how to use Get-S3ACL.

Example 1: The command gets the details of the object owner of the S3 object.

```
Get-S3ACL -BucketName 's3casetestbucket' -key 'initialize.ps1' -Select AccessControlList.Owner
```

Output:

```
DisplayName  Id
------------- -----
testusername  9988776a6554433d22f1100112e334acb45566778899009e9887bd7f66c5f544
```

• For API details, see GetACL in AWS Tools for PowerShell Cmdlet Reference.

Get-S3Bucket

The following code example shows how to use Get-S3Bucket.

Example 1: This command returns all S3 buckets.
Get-S3Bucket

**Example 2: This command returns bucket named "test-files"**

Get-S3Bucket -BucketName test-files

- For API details, see [ListBuckets](AWS Tools for PowerShell Cmdlet Reference) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-S3BucketAccelerateConfiguration

The following code example shows how to use Get-S3BucketAccelerateConfiguration.

**Tools for PowerShell**

**Example 1: This command returns the value Enabled, if the transfer acceleration settings is enabled for the bucket specified.**

Get-S3BucketAccelerateConfiguration -BucketName 's3testbucket'

**Output:**

<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
</tr>
<tr>
<td>Enabled</td>
</tr>
</tbody>
</table>

- For API details, see [GetBucketAccelerateConfiguration](AWS Tools for PowerShell Cmdlet Reference) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-S3BucketAnalyticsConfiguration

The following code example shows how to use Get-S3BucketAnalyticsConfiguration.

**Tools for PowerShell**

**Example 1: This command returns the details of the analytics filter with the name 'testfilter' in the given S3 bucket.**

Get-S3BucketAnalyticsConfiguration -BucketName 's3testbucket' -AnalyticsId 'testfilter'
• For API details, see `GetBucketAnalyticsConfiguration` in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-S3BucketAnalyticsConfigurationList**

The following code example shows how to use `Get-S3BucketAnalyticsConfigurationList`.

**Tools for PowerShell**

**Example 1:** This command returns the first 100 analytics configurations of the given S3 bucket.

```powershell
Get-S3BucketAnalyticsConfigurationList -BucketName 's3casetestbucket'
```

• For API details, see `ListBucketAnalyticsConfigurations` in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-S3BucketEncryption**

The following code example shows how to use `Get-S3BucketEncryption`.

**Tools for PowerShell**

**Example 1:** This command returns all the server side encryption rules associated with the given bucket.

```powershell
Get-S3BucketEncryption -BucketName 's3casetestbucket'
```

• For API details, see `GetBucketEncryption` in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-S3BucketInventoryConfiguration**

The following code example shows how to use `Get-S3BucketInventoryConfiguration`.

**Tools for PowerShell**

**Example 1:** This command returns the details of the inventory named 'testinventory' for the given S3 bucket.
Get-S3BucketInventoryConfiguration -BucketName 's3testbucket' -InventoryId 'testinventory'

- For API details, see [GetBucketInventoryConfiguration](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-S3BucketInventoryConfigurationList

The following code example shows how to use Get-S3BucketInventoryConfigurationList.

**Tools for PowerShell**

**Example 1:** This command returns the first 100 inventory configurations of the given S3 bucket.

```powershell
Get-S3BucketInventoryConfigurationList -BucketName 's3testbucket'
```

- For API details, see [ListBucketInventoryConfigurations](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-S3BucketLocation

The following code example shows how to use Get-S3BucketLocation.

**Tools for PowerShell**

**Example 1:** This command returns the location constraint for the bucket 's3testbucket', if a constraint exists.

```powershell
Get-S3BucketLocation -BucketName 's3testbucket'
```

**Output:**

```
Value
-----
ap-south-1
```

- For API details, see [GetBucketLocation](#) in *AWS Tools for PowerShell Cmdlet Reference*.
Get-S3BucketLogging

The following code example shows how to use Get-S3BucketLogging.

Tools for PowerShell

Example 1: This command returns the logging status for the specified bucket.

```
Get-S3BucketLogging -BucketName 's3testbucket'
```

Output:

```
TargetBucketName  Grants  TargetPrefix
----------------  ------  ------------
testbucket1       {}      testprefix
```

- For API details, see GetBucketLogging in AWS Tools for PowerShell Cmdlet Reference.

Get-S3BucketMetricsConfiguration

The following code example shows how to use Get-S3BucketMetricsConfiguration.

Tools for PowerShell

Example 1: This command returns the details about the metrics filter named 'testfilter' for the given S3 bucket.

```
Get-S3BucketMetricsConfiguration -BucketName 's3testbucket' -MetricsId 'testfilter'
```

- For API details, see GetBucketMetricsConfiguration in AWS Tools for PowerShell Cmdlet Reference.

Get-S3BucketNotification

The following code example shows how to use Get-S3BucketNotification.

Tools for PowerShell

Example 1: This example retrieves notification configuration of the given bucket
Get-S3BucketNotification -BucketName kt-tools | select -ExpandProperty TopicConfigurations

Output:

<table>
<thead>
<tr>
<th>Id</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>mimo</td>
<td>arn:aws:sns:eu-west-1:123456789012:topic-1</td>
</tr>
</tbody>
</table>

- For API details, see [GetBucketNotification](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-S3BucketPolicy

The following code example shows how to use Get-S3BucketPolicy.

**Tools for PowerShell**

**Example 1:** This command outputs the bucket policy associated with the given S3 bucket.

Get-S3BucketPolicy -BucketName 's3testbucket'

- For API details, see [GetBucketPolicy](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-S3BucketPolicyStatus

The following code example shows how to use Get-S3BucketPolicyStatus.

**Tools for PowerShell**

**Example 1:** This command returns policy status for the given S3 bucket, indicating whether the bucket is public.

Get-S3BucketPolicyStatus -BucketName 's3casetestbucket'

- For API details, see [GetBucketPolicyStatus](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-S3BucketReplication

The following code example shows how to use Get-S3BucketReplication.
Tools for PowerShell

Example 1: Returns the replication configuration information set on the bucket named 'mybucket'.

```
Get-S3BucketReplication -BucketName mybucket
```


Get-S3BucketRequestPayment

The following code example shows how to use Get-S3BucketRequestPayment.

Tools for PowerShell

Example 1: Returns the request payment configuration for the bucket named 'mybucket'. By default, the bucket owner pays for downloads from the bucket.

```
Get-S3BucketRequestPayment -BucketName mybucket
```

- For API details, see [GetBucketRequestPayment](https://awsdocs.amazonwebservices.com/AWSToolsForPowerShell/latest/LatestCmdletReference/G_GetBucketRequestPayment.html) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-S3BucketTagging

The following code example shows how to use Get-S3BucketTagging.

Tools for PowerShell

Example 1: This command returns all the tags associated with the given bucket.

```
Get-S3BucketTagging -BucketName 's3casetestbucket'
```


Get-S3BucketVersioning

The following code example shows how to use Get-S3BucketVersioning.

Tools for PowerShell

Example 1: This command returns the status of versioning with respect to the given bucket.
Get-S3BucketVersioning -BucketName 's3testbucket'

- For API details, see [GetBucketVersioning](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-S3BucketWebsite

The following code example shows how to use Get-S3BucketWebsite.

**Tools for PowerShell**

**Example 1:** This command returns the details of the static website configurations of the given S3 bucket.

Get-S3BucketWebsite -BucketName 's3testbucket'

- For API details, see [GetBucketWebsite](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-S3CORSConfiguration

The following code example shows how to use Get-S3CORSConfiguration.

**Tools for PowerShell**

**Example 1:** This command returns an object that contains all the CORS configuration rules corresponding to the given S3 Bucket.

Get-S3CORSConfiguration -BucketName 's3testbucket' -Select Configuration.Rules

**Output:**

```
AllowedMethods : {PUT, POST, DELETE}
AllowedOrigins : {http://www.example1.com}
Id             :
ExposeHeaders  : {}
MaxAgeSeconds  : 0
AllowedHeaders : {*}

AllowedMethods : {PUT, POST, DELETE}
AllowedOrigins : {http://www.example2.com}
```
### Get-S3LifecycleConfiguration

The following code example shows how to use Get-S3LifecycleConfiguration.

**Tools for PowerShell**

**Example 1:** This example retrieves lifecycle configuration for the bucket.

```powershell
Get-S3LifecycleConfiguration -BucketName test-bla
```

**Output:**

```
Rules
-----
{Remove-in-150-days, Archive-to-Glacier-in-30-days}
```

**For API details, see** [GetLifecycleConfiguration](https://aws.amazon.com/tools政权) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-S3Object

The following code example shows how to use Get-S3Object.

**Tools for PowerShell**

**Example 1:** This command retrieves the information about all of the items in the bucket "test-files".

```powershell
Get-S3Object
```
Get-S3Object -BucketName test-files

Example 2: This command retrieves the information about the item "sample.txt" from bucket "test-files".

Get-S3Object -BucketName test-files -Key sample.txt

Example 3: This command retrieves the information about all items with the prefix "sample" from bucket "test-files".

Get-S3Object -BucketName test-files -KeyPrefix sample

- For API details, see ListObjects in AWS Tools for PowerShell Cmdlet Reference.

Get-S3ObjectLockConfiguration

The following code example shows how to use Get-S3ObjectLockConfiguration.

Tools for PowerShell

Example 1: This command returns the value 'Enabled' if Object lock configuration is enabled for the given S3 bucket.

Get-S3ObjectLockConfiguration -BucketName 's3buckettesting' -Select ObjectLockConfiguration.ObjectLockEnabled

Output:

<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
</tr>
</tbody>
</table>

- For API details, see GetObjectLockConfiguration in AWS Tools for PowerShell Cmdlet Reference.

Get-S3ObjectMetadata

The following code example shows how to use Get-S3ObjectMetadata.
Example 1: This command returns the metadata of the object with key 'ListTrusts.txt' in the given S3 bucket.

```
Get-S3ObjectMetadata -BucketName 's3testbucket' -Key 'ListTrusts.txt'
```

Output:

<table>
<thead>
<tr>
<th>Headers</th>
<th>Amazon.S3.Model.HeadersCollection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metadata</td>
<td>Amazon.S3.Model.MetadataCollection</td>
</tr>
<tr>
<td>DeleteMarker</td>
<td>:</td>
</tr>
<tr>
<td>AcceptRanges</td>
<td>: bytes</td>
</tr>
<tr>
<td>ContentRange</td>
<td>:</td>
</tr>
<tr>
<td>Expiration</td>
<td>:</td>
</tr>
<tr>
<td>RestoreExpiration</td>
<td>:</td>
</tr>
<tr>
<td>RestoreInProgress</td>
<td>: False</td>
</tr>
<tr>
<td>LastModified</td>
<td>: 01/01/2020 08:02:05</td>
</tr>
<tr>
<td>ETag</td>
<td>: &quot;d000011112a222e333e3bb4ee5d43d21&quot;</td>
</tr>
<tr>
<td>MissingMeta</td>
<td>: 0</td>
</tr>
<tr>
<td>VersionId</td>
<td>: null</td>
</tr>
<tr>
<td>Expires</td>
<td>: 01/01/0001 00:00:00</td>
</tr>
<tr>
<td>WebsiteRedirectLocation</td>
<td>:</td>
</tr>
<tr>
<td>ServerSideEncryptionMethod</td>
<td>: AES256</td>
</tr>
<tr>
<td>ServerSideEncryptionCustomerMethod</td>
<td>:</td>
</tr>
<tr>
<td>ServerSideEncryptionKeyManagementServiceKeyId</td>
<td>:</td>
</tr>
<tr>
<td>ReplicationStatus</td>
<td>:</td>
</tr>
<tr>
<td>PartsCount</td>
<td>:</td>
</tr>
<tr>
<td>ObjectLockLegalHoldStatus</td>
<td>:</td>
</tr>
<tr>
<td>ObjectLockMode</td>
<td>:</td>
</tr>
<tr>
<td>ObjectLockRetainUntilDate</td>
<td>: 01/01/0001 00:00:00</td>
</tr>
<tr>
<td>StorageClass</td>
<td>:</td>
</tr>
<tr>
<td>RequestCharged</td>
<td>:</td>
</tr>
</tbody>
</table>

- For API details, see GetObjectMetadata in AWS Tools for PowerShell Cmdlet Reference.

Get-S3ObjectRetention

The following code example shows how to use Get-S3ObjectRetention.
Tools for PowerShell

Example 1: The command returns the mode and date till the object would be retained.

```
Get-S3ObjectRetention -BucketName 's3buckettesting' -Key 'testfile.txt'
```

For API details, see [GetObjectRetention](#) in AWS Tools for PowerShell Cmdlet Reference.

Get-S3ObjectTagSet

The following code example shows how to use Get-S3ObjectTagSet.

```
Get-S3ObjectTagSet -Key 'testfile.txt' -BucketName 'testbucket123'
```

Output:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>test</td>
<td>value</td>
</tr>
</tbody>
</table>

For API details, see [GetObjectTagging](#) in AWS Tools for PowerShell Cmdlet Reference.

Get-S3PreSignedURL

The following code example shows how to use Get-S3PreSignedURL.

Tools for PowerShell

Example 1: The command returns pre-signed URL for a specified key and an expiration date.

```
Get-S3PreSignedURL -BucketName 's3testbucket' -Key 'testkey' -Expires '2023-11-16'
```

Example 2: The command returns pre-signed URL for a Directory Bucket with specified key and an expiration date.
[Amazon.AWSConfigsS3]::UseSignatureVersion4 = $true
Get-S3PreSignedURL -BucketName sampledirectorybucket--use1-az5--x-s3 -Key 'testkey' -Expire '2023-11-17'


**Get-S3PublicAccessBlock**

The following code example shows how to use Get-S3PublicAccessBlock.

**Tools for PowerShell**

**Example 1:** The command returns the public access block configuration of the given S3 bucket.

```
Get-S3PublicAccessBlock -BucketName 's3testbucket'
```


**Get-S3Version**

The following code example shows how to use Get-S3Version.

**Tools for PowerShell**

**Example 1:** This command returns the metadata about all of the versions of objects in the given S3 bucket.

```
Get-S3Version -BucketName 's3testbucket'
```

**Output:**

```
IsTruncated : False
KeyMarker   :
VersionIdMarker :
NextKeyMarker :
NextVersionIdMarker :
Versions : {EC2.txt, EC2MicrosoftWindowsGuide.txt, ListDirectories.json, ListTrusts.json}
Name : s3testbucket
```
Prefix              :  
MaxKeys             : 1000
CommonPrefixes      : {}
Delimiter           :

- For API details, see [ListVersions](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**New-S3Bucket**

The following code example shows how to use New-S3Bucket.

**Tools for PowerShell**

**Example 1:** This command creates a new private bucket named "sample-bucket".

```
New-S3Bucket -BucketName sample-bucket
```

**Example 2:** This command creates a new bucket named "sample-bucket" with read-write permissions.

```
New-S3Bucket -BucketName sample-bucket -PublicReadWrite
```

**Example 3:** This command creates a new bucket named "sample-bucket" with read-only permissions.

```
New-S3Bucket -BucketName sample-bucket -PublicReadOnly
```

**Example 4:** This command creates a new Directory bucket named "samplebucket--use1-az5--x-s3" with PutBucketConfiguration.

```powershell
$bucketConfiguration = @{
    BucketInfo = @{
        DataRedundancy = 'SingleAvailabilityZone'
        Type = 'Directory'
    }
    Location = @{
        Name = 'use1-az5'
        Type = 'AvailabilityZone'
    }
}
```

Amazon S3 686
New-S3Bucket -BucketName samplebucket --use1-az5 --x-s3 -BucketConfiguration $bucketConfiguration -Region us-east-1

- For API details, see PutBucket in AWS Tools for PowerShell Cmdlet Reference.

Read-S3Object

The following code example shows how to use Read-S3Object.

Tools for PowerShell

Example 1: This command retrieves item "sample.txt" from bucket "test-files" and saves it to a file named "local-sample.txt" in the current location. The file "local-sample.txt" does not have to exist before this command is called.

Read-S3Object -BucketName test-files -Key sample.txt -File local-sample.txt

Example 2: This command retrieves virtual directory "DIR" from bucket "test-files" and saves it to a folder named "Local-DIR" in the current location. The folder "Local-DIR" does not have to exist before this command is called.

Read-S3Object -BucketName test-files -KeyPrefix DIR -Folder Local-DIR

Example 3: Downloads all objects with keys ending in '.json' from buckets with 'config' in the bucket name to files in the specified folder. The object keys are used to set the filenames.

Get-S3Bucket | ? {$_ .BucketName -like '*config*' } | Get-S3Object | ? {$_ .Key -like '*json' } | Read-S3Object -Folder C:\ConfigObjects

- For API details, see GetObject in AWS Tools for PowerShell Cmdlet Reference.

Remove-S3Bucket

The following code example shows how to use Remove-S3Bucket.

Tools for PowerShell

Example 1: This command removes all objects and object versions from the bucket 'test-files' and then deletes the bucket. The command will prompt for confirmation before
proceeding. Add the -Force switch to suppress confirmation. Note that buckets that are not empty cannot be deleted.

Remove-S3Bucket -BucketName test-files -DeleteBucketContent

- For API details, see DeleteBucket in AWS Tools for PowerShell Cmdlet Reference.

Remove-S3BucketAnalyticsConfiguration

The following code example shows how to use Remove-S3BucketAnalyticsConfiguration.

Tools for PowerShell

Example 1: The command removes the analytics filter with name 'testfilter' in the given S3 bucket.

Remove-S3BucketAnalyticsConfiguration -BucketName 's3testbucket' -AnalyticsId 'testfilter'

- For API details, see DeleteBucketAnalyticsConfiguration in AWS Tools for PowerShell Cmdlet Reference.

Remove-S3BucketEncryption

The following code example shows how to use Remove-S3BucketEncryption.

Tools for PowerShell

Example 1: This disables the encryption enabled for the S3 bucket provided.

Remove-S3BucketEncryption -BucketName 's3casetestbucket'

Output:

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-S3BucketEncryption (DeleteBucketEncryption)" on target "s3casetestbucket".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): Y
• For API details, see [DeleteBucketEncryption](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-S3BucketInventoryConfiguration**

The following code example shows how to use Remove-S3BucketInventoryConfiguration.

**Tools for PowerShell**

*Example 1: This command removes the inventory named 'testInventoryName' corresponding to the given S3 bucket.*

```powershell
Remove-S3BucketInventoryConfiguration -BucketName 's3testbucket' -InventoryId 'testInventoryName'
```

**Output:**

```
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-S3BucketInventoryConfiguration" on target "s3testbucket".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): Y
```

• For API details, see [DeleteBucketInventoryConfiguration](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-S3BucketMetricsConfiguration**

The following code example shows how to use Remove-S3BucketMetricsConfiguration.

**Tools for PowerShell**

*Example 1: The command removes the metrics filter with name 'testmetrics' in the given S3 bucket.*

```powershell
Remove-S3BucketMetricsConfiguration -BucketName 's3testbucket' -MetricsId 'testmetrics'
```

• For API details, see [DeleteBucketMetricsConfiguration](#) in *AWS Tools for PowerShell Cmdlet Reference*.
Remove-S3BucketPolicy

The following code example shows how to use Remove-S3BucketPolicy.

**Tools for PowerShell**

**Example 1:** The command removes the bucket policy associated with the given S3 bucket.

Remove-S3BucketPolicy -BucketName 's3testbucket'

- For API details, see [DeleteBucketPolicy](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Remove-S3BucketReplication

The following code example shows how to use Remove-S3BucketReplication.

**Tools for PowerShell**

**Example 1:** Deletes the replication configuration associated with the bucket named 'mybucket'. Note that this operation requires permission for the s3:DeleteReplicationConfiguration action. You will be prompted for confirmation before the operation proceeds - to suppress confirmation, use the -Force switch.

Remove-S3BucketReplication -BucketName mybucket

- For API details, see [DeleteBucketReplication](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Remove-S3BucketTagging

The following code example shows how to use Remove-S3BucketTagging.

**Tools for PowerShell**

**Example 1:** This command removes all the tags associated with the given S3 bucket.

Remove-S3BucketTagging -BucketName 's3testbucket'

**Output:**

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-S3BucketTagging (DeleteBucketTagging)" on target "s3testbucket".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): Y

- For API details, see [DeleteBucketTagging](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-S3BucketWebsite**

The following code example shows how to use Remove-S3BucketWebsite.

**Tools for PowerShell**

**Example 1:** This command disables the static website hosting property of the given S3 bucket.

Remove-S3BucketWebsite -BucketName 's3testbucket'

**Output:**

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-S3BucketWebsite (DeleteBucketWebsite)" on target "s3testbucket".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): Y

- For API details, see [DeleteBucketWebsite](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-S3CORSConfiguration**

The following code example shows how to use Remove-S3CORSConfiguration.

**Tools for PowerShell**

**Example 1:** This command removes the CORS configuration for the given S3 bucket.

Remove-S3CORSConfiguration -BucketName 's3testbucket'

**Output:**
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-S3CORSConfiguration (DeleteCORSConfiguration)" on target "s3testbucket".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y") : Y

- For API details, see [DeleteCORSConfiguration](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

Remove-S3LifecycleConfiguration

The following code example shows how to use Remove-S3LifecycleConfiguration.

Tools for PowerShell

**Example 1:** The command removes all the lifecycle rules for the given S3 bucket.

Remove-S3LifecycleConfiguration -BucketName 's3testbucket'

- For API details, see [DeleteLifecycleConfiguration](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

Remove-S3MultipartUpload

The following code example shows how to use Remove-S3MultipartUpload.

Tools for PowerShell

**Example 1:** This command aborts multipart uploads created earlier than 5 days ago.

Remove-S3MultipartUpload -BucketName test-files -DaysBefore 5

**Example 2:** This command aborts multipart uploads created earlier than January 2nd, 2014.

Remove-S3MultipartUpload -BucketName test-files -InitiatedDate "Thursday, January 02, 2014"

**Example 3:** This command aborts multipart uploads created earlier than January 2nd, 2014, 10:45:37.
Remove-S3MultipartUpload -BucketName test-files -InitiatedDate "2014/01/02 10:45:37"

- For API details, see [AbortMultipartUpload](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-S3Object**

The following code example shows how to use Remove-S3Object.

**Tools for PowerShell**

**Example 1:** This command removes the object "sample.txt" from bucket "test-files". You are prompted for confirmation before the command executes; to suppress the prompt use the -Force switch.

```powershell
Remove-S3Object -BucketName test-files -Key sample.txt
```

**Example 2:** This command removes the specified version of object "sample.txt" from bucket "test-files", assuming the bucket has been configured to enable object versions.

```powershell
Remove-S3Object -BucketName test-files -Key sample.txt -VersionId HLbxnx6V9omT6AQYVpks8mmFKQcejpqt
```

**Example 3:** This command removes objects "sample1.txt", "sample2.txt" and "sample3.txt" from bucket "test-files" as a single batch operation. The service response will list all keys processed, regardless of the success or error status of the deletion. To obtain only errors for keys that were not able to be processed by the service add the -ReportErrorsOnly parameter (this parameter can also be specified with the alias -Quiet).

```powershell
Remove-S3Object -BucketName test-files -KeyCollection @("sample1.txt", "sample2.txt", "sample3.txt")
```

**Example 4:** This example uses an inline expression with the -KeyCollection parameter to obtain the keys of the objects to delete. Get-S3Object returns a collection of Amazon.S3.Model.S3Object instances, each of which has a Key member of type string identifying the object.

```powershell
Remove-S3Object -BucketName "test-files" -KeyCollection (Get-S3Object "test-files" -KeyPrefix "prefix/subprefix" | select -ExpandProperty Key)
Example 5: This example obtains all objects that have a key prefix "prefix/subprefix" in the bucket and deletes them. Note that the incoming objects are processed one at a time. For large collections consider passing the collection to the cmdlet's -InputObject (alias -S3ObjectCollection) parameter to enable the deletion to occur as a batch with a single call to the service.

Get-S3Object -BucketName "test-files" -KeyPrefix "prefix/subprefix" | Remove-S3Object -Force

Example 6: This example pipes a collection of Amazon.S3.Model.S3ObjectVersion instances that represent delete markers to the cmdlet for deletion. Note that the incoming objects are processed one at a time. For large collections consider passing the collection to the cmdlet's -InputObject (alias -S3ObjectCollection) parameter to enable the deletion to occur as a batch with a single call to the service.

(Get-S3Version -BucketName "test-files").Versions | Where {$_..IsDeleteMarker -eq "True"} | Remove-S3Object -Force

Example 7: This script shows how to perform a batch delete of a set of objects (in this case delete markers) by constructing an array of objects to be used with the -KeyAndVersionCollection parameter.

$keyVersions = @()
$markers = (Get-S3Version -BucketName $BucketName).Versions | Where {$_..IsDeleteMarker -eq "True"}
foreach ($marker in $markers) { $keyVersions += @{ Key = $marker.Key; VersionId = $marker.VersionId } }
Remove-S3Object -BucketName $BucketName -KeyAndVersionCollection $keyVersions -Force

- For API details, see DeleteObjects in AWS Tools for PowerShell Cmdlet Reference.

Remove-S3ObjectTagSet

The following code example shows how to use Remove-S3ObjectTagSet.

Tools for PowerShell

Example 1: This command removes all the tags associated with the object with key 'testfile.txt' in the given S3 Bucket.
Remove-S3ObjectTagSet -Key 'testfile.txt' -BucketName 's3testbucket' -Select '^Key'

**Output:**

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-S3ObjectTagSet (DeleteObjectTagging)" on target "testfile.txt".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): Y
testfile.txt

- For API details, see [DeleteObjectTagging](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-S3PublicAccessBlock**

The following code example shows how to use Remove-S3PublicAccessBlock.

**Tools for PowerShell**

**Example 1:** This command turns off the block public access setting for the given bucket.

Remove-S3PublicAccessBlock -BucketName 's3testbucket' -Force -Select '^BucketName'

**Output:**

s3testbucket

- For API details, see [DeletePublicAccessBlock](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Set-S3BucketEncryption**

The following code example shows how to use Set-S3BucketEncryption.

**Tools for PowerShell**

**Example 1:** This command enables default AES256 server side encryption with Amazon S3 Managed Keys(SSE-S3) on the given bucket.
$Encryptionconfig = @{ServerSideEncryptionByDefault = @{ServerSideEncryptionAlgorithm = "AES256"}}
Set-S3BucketEncryption -BucketName 's3testbucket' -ServerSideEncryptionConfiguration_ServerSideEncryptionRule $Encryptionconfig

- For API details, see [PutBucketEncryption](https://aws.amazon.com) in *AWS Tools for PowerShell Cmdlet Reference*.

**Test-S3Bucket**

The following code example shows how to use `Test-S3Bucket`.

**Tools for PowerShell**

**Example 1:** This command returns True if the bucket exists, False otherwise. The command returns True even if the bucket does not belong to the user.

```powershell
Test-S3Bucket -BucketName test-files
```

- For API details, see [Test-S3Bucket](https://aws.amazon.com) in *AWS Tools for PowerShell Cmdlet Reference*.

**Write-S3BucketAccelerateConfiguration**

The following code example shows how to use `Write-S3BucketAccelerateConfiguration`.

**Tools for PowerShell**

**Example 1:** This command enables the transfer acceleration for the given S3 bucket.

```powershell
$statusVal = New-Object Amazon.S3.BucketAccelerateStatus('Enabled')
Write-S3BucketAccelerateConfiguration -BucketName 's3testbucket' -AccelerateConfiguration_Status $statusVal
```

- For API details, see [PutBucketAccelerateConfiguration](https://aws.amazon.com) in *AWS Tools for PowerShell Cmdlet Reference*.

**Write-S3BucketNotification**

The following code example shows how to use `Write-S3BucketNotification`.
Example 1: This example configures the SNS topic configuration for the S3 event ObjectRemovedDelete and enables notification for the given s3 bucket

```
    Id = "delete-event"
    Topic = "arn:aws:sns:eu-west-1:123456789012:topic-1"
    Event = [Amazon.S3.EventType]::ObjectRemovedDelete
}

Write-S3BucketNotification -BucketName kt-tools -TopicConfiguration $topic
```

Example 2: This example enables notifications of ObjectCreatedAll for the given bucket sending it to Lambda function.

```
    Events = "s3:ObjectCreated:*"
    FunctionArn = "arn:aws:lambda:eu-west-1:123456789012:function:rdplock"
    Id = "ObjectCreated-Lambda"
    Filter = @{
        S3KeyFilter = @{
            FilterRules = @(  
                @{Name="Prefix";Value="dada"}  
                @{Name="Suffix";Value=".pem"}
            )
        }
    }
}

Write-S3BucketNotification -BucketName ssm-editor -LambdaFunctionConfiguration $lambdaConfig
```

Example 3: This example creates 2 different Lambda configuration on the basis of different key-suffix and configured both in a single command.

```
#Lambda Config 1

    Events = "s3:ObjectCreated:*"
    FunctionArn = "arn:aws:lambda:eu-west-1:123456789012:function:verifynet"
    Id = "ObjectCreated-dada-ps1"
```
Filter = @{
  S3KeyFilter = @{
    FilterRules = @(
      @{Name="Prefix";Value="dada"}
      @{Name="Suffix";Value=".ps1"}
    )
  }
}

#Lambda Config 2
  Events = [Amazon.S3.EventType]::ObjectCreatedAll
  FunctionArn = "arn:aws:lambda:eu-west-1:123456789012:function:verifyssm"
  Id = "ObjectCreated-dada-json"
  Filter = @{
    S3KeyFilter = @{
      FilterRules = @(
        @{Name="Prefix";Value="dada"}
        @{Name="Suffix";Value=".json"}
      )
    }
  }
}
Write-S3BucketNotification -BucketName ssm-editor -LambdaFunctionConfiguration $firstLambdaConfig,$secondlambdaConfig


**Write-S3BucketReplication**

The following code example shows how to use *Write-S3BucketReplication*.

**Tools for PowerShell**

**Example 1:** This example sets a replication configuration with a single rule enabling replication to the 'exampletargetbucket' bucket any new objects created with the key name prefix "TaxDocs" in the bucket 'examplebucket'.

```powershell
$rule1 = New-Object Amazon.S3.Model.ReplicationRule
```
Example 2: This example sets a replication configuration with multiple rules enabling replication to the 'exampletargetbucket' bucket any new objects created with either the key name prefix "TaxDocs" or "OtherDocs". The key prefixes must not overlap.

```powershell
$rule1 = New-Object Amazon.S3.Model.ReplicationRule
$rule1.ID = "Rule-1"
$rule1.Status = "Enabled"
$rule1.Prefix = "TaxDocs"
$rule1.Destination = @{ BucketArn = "arn:aws:s3:::exampletargetbucket" }
$rule2 = New-Object Amazon.S3.Model.ReplicationRule
$rule2.ID = "Rule-2"
$rule2.Status = "Enabled"
$rule2.Prefix = "OtherDocs"
$rule2.Destination = @{ BucketArn = "arn:aws:s3:::exampletargetbucket" }
$params = @{
    BucketName = "examplebucket"
    Configuration_Role = "arn:aws:iam::35667example:role/CrossRegionReplicationRoleForS3"
    Configuration_Rule = $rule1,$rule2
}
Write-S3BucketReplication @params
```

Example 3: This example updates the replication configuration on the specified bucket to disable the rule controlling replication of objects with the key name prefix "TaxDocs" to the bucket 'exampletargetbucket'.

```powershell
$rule1.Destination.BucketArn = "$null"
$params = @{
    BucketName = "examplebucket"
    Configuration_Role = "arn:aws:iam::35667example:role/CrossRegionReplicationRoleForS3"
    Configuration_Rule = $rule1
}
Write-S3BucketReplication @params
```
$rule1 = New-Object Amazon.S3.Model.ReplicationRule
$rule1.ID = "Rule-1"
$rule1.Status = "Disabled"
$rule1.Prefix = "TaxDocs"
$rule1.Destination = @{ BucketArn = "arn:aws:s3:::exampletargetbucket" }

$params = @{
    BucketName = "examplebucket"
    Configuration_Role = "arn:aws:iam::35667example:role/CrossRegionReplicationRoleForS3"
    Configuration_Rule = $rule1
}

Write-S3BucketReplication @params

- For API details, see [PutBucketReplication](https://docs.aws.amazon.com/powershell/latest/reference/commands.php) in *AWS Tools for PowerShell Cmdlet Reference*.

---

### Write-S3BucketRequestPayment

The following code example shows how to use `Write-S3BucketRequestPayment`.

**Tools for PowerShell**

**Example 1:** Updates the request payment configuration for the bucket named 'mybucket' so that the person requesting downloads from the bucket will be charged for the download. By default the bucket owner pays for downloads. To set the request payment back to the default use 'BucketOwner' for the RequestPaymentConfiguration_Payer parameter.

```
Write-S3BucketRequestPayment -BucketName mybucket -RequestPaymentConfiguration_Payer Requester
```

- For API details, see [PutBucketRequestPayment](https://docs.aws.amazon.com/powershell/latest/reference/commands.php) in *AWS Tools for PowerShell Cmdlet Reference*.

---

### Write-S3BucketTagging

The following code example shows how to use `Write-S3BucketTagging`.

**Tools for PowerShell**

**Example 1:** This command applies two tags to a bucket named `cloudtrail-test-2018`: a tag with a key of Stage and a value of Test, and a tag with a key of Environment and a value...
of Alpha. To verify that the tags were added to the bucket, run `Get-S3BucketTagging -BucketName bucket_name`. The results should show the tags that you applied to the bucket in the first command. Note that `Write-S3BucketTagging` overwrites the entire existing tag set on a bucket. To add or delete individual tags, run the Resource Groups and Tagging API cmdlets, `Add-RGTResourceTag` and `Remove-RGTResourceTag`. Alternatively, use Tag Editor in the AWS Management Console to manage S3 bucket tags.

```powershell
Write-S3BucketTagging -BucketName cloudtrail-test-2018 -TagSet @( @{ Key="Stage"; Value="Test" }, @{ Key="Environment"; Value="Alpha" } )
```

**Example 2:** This command pipes a bucket named `cloudtrail-test-2018` into the `Write-S3BucketTagging` cmdlet. It applies tags `Stage:Production` and `Department:Finance` to the bucket. Note that `Write-S3BucketTagging` overwrites the entire existing tag set on a bucket.

```powershell
Get-S3Bucket -BucketName cloudtrail-test-2018 | Write-S3BucketTagging -TagSet @( @{ Key="Stage"; Value="Production" }, @{ Key="Department"; Value="Finance" } )
```


**Write-S3BucketVersioning**

The following code example shows how to use `Write-S3BucketVersioning`.

**Tools for PowerShell**

**Example 1:** The command enables versioning for the given S3 bucket.

```powershell
Write-S3BucketVersioning -BucketName 's3testbucket' -VersioningConfig_Status Enabled
```


**Write-S3BucketWebsite**

The following code example shows how to use `Write-S3BucketWebsite`.

**Tools for PowerShell**

**Example 1:** The command enables website hosting for the given bucket with the index document as 'index.html' and error document as 'error.html'.
Write-S3BucketWebsite -BucketName 's3testbucket' -
WebsiteConfiguration_IndexDocumentSuffix 'index.html' -
WebsiteConfiguration_ErrorDocument 'error.html'


Write-S3LifecycleConfiguration

The following code example shows how to use Write-S3LifecycleConfiguration.

Tools for PowerShell

Example 1: This example writes / replaces the configuration provided in the $NewRule. This configuration is making sure to limit the scope objects with given prefix and tag values.

```
    Expiration = @{
        Days= 50
    }
    Id = "Test-From-Write-cmdlet-1"
    Filter= @{
        LifecycleFilterPredicate = [Amazon.S3.Model.LifecycleAndOperator] @{
            Operands= @(  
                [Amazon.S3.Model.LifecyclePrefixPredicate] @{
                    "Prefix" = "py"
                },
                [Amazon.S3.Model.LifecycleTagPredicate] @{
                    "Tag"= @{
                        "Key" = "non-use"
                        "Value" = "yes"
                    }
                }
            )
        }
    }
    "Status"= 'Enabled'
    NoncurrentVersionExpiration = @{
        NoncurrentDays = 75
    }
}
```
Example 2: This example sets multiple rules with filtering. $ArchiveRule sets the objects to archive in 30 days to Glacier and 120 to DeepArchive. $ExpireRule expires both current and previous versions in 150 days for objects with 'py' prefix and tag:key 'archived' set to 'yes'.

```powershell
    Expiration = @{
        Days = 150
    }
    Id = "Remove-in-150-days"
    Filter = @{
        LifecycleFilterPredicate = [Amazon.S3.Model.LifecycleAndOperator]@{
            Operands = @(
                [Amazon.S3.Model.LifecyclePrefixPredicate] @{
                    "Prefix" = "py"
                },
                [Amazon.S3.Model.LifecycleTagPredicate] @{
                    "Tag" = @{
                        "Key" = "archived"
                        "Value" = "yes"
                    }
                }
            )
        }
    }
    Status = 'Enabled'
    NoncurrentVersionExpiration = @{
        NoncurrentDays = 150
    }
}

    Expiration = $null
    Id = "Archive-to-Glacier-in-30-days"
    Filter = @{
        LifecycleFilterPredicate = [Amazon.S3.Model.LifecycleAndOperator]@{
            Operands = @(
                [Amazon.S3.Model.LifecyclePrefixPredicate] @{
                    "Prefix" = "py"
                },
                [Amazon.S3.Model.LifecycleTagPredicate] @{
                    "Tag" = @{
                        "Key" = "archived"
                        "Value" = "yes"
                    }
                }
            )
        }
    }
    Status = 'Enabled'
    NoncurrentVersionExpiration = @{
        NoncurrentDays = 150
    }
}
```

Write-S3LifecycleConfiguration -BucketName my-review-scrap -Configuration_Rule $NewRule
"Tag" = @{
    "Key" = "reviewed"
    "Value" = "yes"
}

Status = 'Enabled'
NoncurrentVersionExpiration = @{
    NoncurrentDays = 75
}
Transitions = @( 
    @{
        Days = 30
        "StorageClass" = 'Glacier'
    },
    @{
        Days = 120
        "StorageClass" = [Amazon.S3.S3StorageClass]::DeepArchive
    }
)

Write-S3LifecycleConfiguration -BucketName my-review-scrap -Configuration_Rule $ExpireRule,$ArchiveRule

- For API details, see [PutLifecycleConfiguration](#) in AWS Tools for PowerShell Cmdlet Reference.

**Write-S3Object**

The following code example shows how to use `Write-S3Object`.

**Tools for PowerShell**

**Example 1:** This command uploads the single file "local-sample.txt" to Amazon S3, creating an object with key "sample.txt" in bucket "test-files".

`Write-S3Object -BucketName test-files -Key "sample.txt" -File .\local-sample.txt`
Example 2: This command uploads the single file "sample.txt" to Amazon S3, creating an object with key "sample.txt" in bucket "test-files". If the -Key parameter is not supplied, the filename is used as the S3 object key.

Write-S3Object -BucketName test-files -File .\sample.txt

Example 3: This command uploads the single file "local-sample.txt" to Amazon S3, creating an object with key "prefix/to/sample.txt" in bucket "test-files".

Write-S3Object -BucketName test-files -Key "prefix/to/sample.txt" -File .\local-sample.txt

Example 4: This command uploads all files in the subdirectory "Scripts" to the bucket "test-files" and applies the common key prefix "SampleScripts" to each object. Each uploaded file will have a key of "SampleScripts/filename" where 'filename' varies.

Write-S3Object -BucketName test-files -Folder .\Scripts -KeyPrefix SampleScripts\n
Example 5: This command uploads all *.ps1 files in the local director "Scripts" to bucket "test-files" and applies the common key prefix "SampleScripts" to each object. Each uploaded file will have a key of "SampleScripts/filename.ps1" where 'filename' varies.

Write-S3Object -BucketName test-files -Folder .\Scripts -KeyPrefix SampleScripts\ -SearchPattern *.ps1

Example 6: This command creates a new S3 object containing the specified content string with key 'sample.txt'.

Write-S3Object -BucketName test-files -Key "sample.txt" -Content "object contents"

Example 7: This command uploads the specified file (the filename is used as the key) and applies the specified tags to the new object.

Write-S3Object -BucketName test-files -File "sample.txt" -TagSet @{Key="key1";Value="value1"},@{Key="key2";Value="value2"}

Example 8: This command recursively uploads the specified folder and applies the specified tags to all the new objects.
Write-S3Object -BucketName test-files -Folder . -KeyPrefix "TaggedFiles" -Recurse -TagSet @{Key="key1";Value="value1"},@{Key="key2";Value="value2"}

- For API details, see PutObject in AWS Tools for PowerShell Cmdlet Reference.

Write-S3ObjectRetention

The following code example shows how to use Write-S3ObjectRetention.

Tools for PowerShell

Example 1: The command enables governance retention mode until the date '31st Dec 2019 00:00:00' for 'testfile.txt' object in the given S3 bucket.

Write-S3ObjectRetention -BucketName 's3buckettesting' -Key 'testfile.txt' -Retention_Mode GOVERNANCE -Retention_RetainUntilDate "2019-12-31T00:00:00"

- For API details, see PutObjectRetention in AWS Tools for PowerShell Cmdlet Reference.

S3 Glacier examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with S3 Glacier.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

- Actions
Actions

Get-GLCJob

The following code example shows how to use Get-GLCJob.

Tools for PowerShell

Example 1: Returns details of the specified job. When the job completes successfully the Read-GCJobOutput cmdlet can be used to retrieve the contents of the job (an archive or inventory list) to the local file system.

```
Get-GLCJob -VaultName myvault -JobId "op1x...JSbthM"
```

Output:

```
Action                       : ArchiveRetrieval
ArchiveId                    : o9O9j...X-TpIhQJw
ArchiveSHA256TreeHash        : 79f3ea754c02f58...dc57bf4395b
ArchiveSizeInBytes           : 38034480
Completed                    : False
CompletionDate               : 1/1/0001 12:00:00 AM
CreationDate                 : 12/13/2018 11:00:14 AM
InventoryRetrievalParameters :
InventorySizeInBytes         : 0
JobDescription               :
JobId                        : op1x...JSbthM
JobOutputPath                :
OutputLocation               :
RetrievalByteRange           : 0-38034479
SelectParameters             :
SHA256TreeHash               : 79f3ea754c02f58...dc57bf4395b
SNSTopic                     :
StatusCode                   : InProgress
StatusMessage                :
Tier                         : Standard
```

- For API details, see [DescribeJob](#) in *AWS Tools for PowerShell Cmdlet Reference*. 
New-GLCVault

The following code example shows how to use New-GLCVault.

Tools for PowerShell

Example 1: Creates a new vault for the user’s account. As no value was supplied to the -AccountId parameter the cmdlets uses a default of "-" indicating the current account.

```
New-GLCVault -VaultName myvault
```

Output:

```
/01234567812/vaults/myvault
```

- For API details, see CreateVault in AWS Tools for PowerShell Cmdlet Reference.

Read-GLCJobOutput

The following code example shows how to use Read-GLCJobOutput.

Tools for PowerShell

Example 1: Downloads the archive content that was scheduled for retrieval in the specified job and stores the contents into a file on disk. The download validates the checksum for you, if one is available. If required the checksum can be obtained from the service response history like so (assuming this cmdlet was the last run):

```
$AWSHistory.LastServiceResponse
```

If the cmdlet was not the most recently run, inspect the $AWSHistory.Commands collection to obtain the relevant service response.

```
Read-GLCJobOutput -VaultName myvault -JobId "HSWjArc...Zq2XLiW" -FilePath "c:\temp\blue.bin"
```

- For API details, see GetJobOutput in AWS Tools for PowerShell Cmdlet Reference.

Start-GLCJob

The following code example shows how to use Start-GLCJob.
Tools for PowerShell

Example 1: Starts a job to retrieve an archive from the specified vault owned by the user. The status of the job can be checked using the Get-GLCJob cmdlet. When the job completes successfully the Read-GCJobOutput cmdlet can be used to retrieve the contents of the archive to the local file system.

```
Start-GLCJob -VaultName myvault -JobType "archive-retrieval" -JobDescription "archive retrieval" -ArchiveId "o909j...TX-TpIhQJw"
```

Output:

```
<table>
<thead>
<tr>
<th>JobId</th>
<th>JobOutputPath</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>op1x...JSbthM</td>
<td>/012345678912/vaults/test/jobs/op1xe...I4HqCHKSJSbthM</td>
<td></td>
</tr>
</tbody>
</table>
```

- For API details, see [InitiateJob](aws/tools/powershell/cmdlet-reference#initiatejob) in *AWS Tools for PowerShell Cmdlet Reference*.

Write-GLCArchive

The following code example shows how to use `Write-GLCArchive`.

Tools for PowerShell

Example 1: Uploads a single file to the specified vault, returning the archive ID and computed checksum.

```
Write-GLCArchive -VaultName myvault -FilePath c:\temp\blue.bin
```

Output:

```
<table>
<thead>
<tr>
<th>FilePath</th>
<th>ArchiveId</th>
<th>Checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td>C:\temp\blue.bin</td>
<td>o909jUUs...TTX-TpIhQJw</td>
<td>79f3e...f4395b</td>
</tr>
</tbody>
</table>
```

Example 2: Uploads the contents of a folder hierarchy to the specified vault in the user's account. For each file uploaded the cmdlet emits the filename, corresponding archive ID and the computed checksum of the archive.

```
Write-GLCArchive -VaultName myvault -FolderPath . -Recurse
```
Output:

<table>
<thead>
<tr>
<th>FilePath</th>
<th>ArchiveId</th>
<th>Checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td>C:\temp\blue.bin</td>
<td>o9O9jUUs...TTX-TpIhQJw</td>
<td>79f3e...f4395b</td>
</tr>
<tr>
<td>C:\temp\green.bin</td>
<td>qXAf0dS6...cz0729UHXrw</td>
<td>d50a1...9184b9</td>
</tr>
<tr>
<td>C:\temp\lum.bin</td>
<td>39aNifP3...q9nb8nZkF1g</td>
<td>28886...5c3e27</td>
</tr>
<tr>
<td>C:\temp\red.bin</td>
<td>vp7E6rU_...Ejk_HhjAxAKA</td>
<td>e05f7...4e34f5</td>
</tr>
<tr>
<td>C:\temp\Folder1\file1.txt</td>
<td>_eRINlip...5Sxy7d2BaA</td>
<td>d0d2a...c8a3ba</td>
</tr>
<tr>
<td>C:\temp\Folder2\file2.iso</td>
<td>-Ix3jlm...iXiDh-Xf0PA</td>
<td>7469e...3e86f1</td>
</tr>
</tbody>
</table>

- For API details, see [UploadArchive](#) in *AWS Tools for PowerShell Cmdlet Reference*.

## Amazon SES examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with Amazon SES.

*Actions* are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

*Scenarios* are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

### Topics

- **Actions**

### Actions

#### Get-SESIdentity

The following code example shows how to use Get-SESIdentity.

#### Tools for PowerShell

**Example 1:** This command returns a list containing all of the identities (email addresses and domains) for a specific AWS Account, regardless of verification status.
Get-SESIdentity

- For API details, see ListIdentities in AWS Tools for PowerShell Cmdlet Reference.

Get-SESSendQuota

The following code example shows how to use Get-SESSendQuota.

**Tools for PowerShell**

**Example 1:** This command returns the user's current sending limits.

```
Get-SESSendQuota
```

- For API details, see GetSendQuota in AWS Tools for PowerShell Cmdlet Reference.

Get-SESSendStatistic

The following code example shows how to use Get-SESSendStatistic.

**Tools for PowerShell**

**Example 1:** This command returns the user's sending statistics. The result is a list of data points, representing the last two weeks of sending activity. Each data point in the list contains statistics for a 15-minute interval.

```
Get-SESSendStatistic
```

- For API details, see GetSendStatistics in AWS Tools for PowerShell Cmdlet Reference.

**Amazon SNS examples using Tools for PowerShell**

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with Amazon SNS.

*Actions* are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.
Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

• Actions

Actions

Publish-SNSMessage

The following code example shows how to use Publish-SNSMessage.

Tools for PowerShell

Example 1: This example shows publishing a message with a single MessageAttribute declared inline.

```
```

Example 2: This example shows publishing a message with multiple MessageAttributes declared in advance.

```
$cityAttributeValue.DataType = "String"
$cityAttributeValue.StringValue = "AnyCity"

$populationAttributeValue = New-Object Amazon.SimpleNotificationService.Model.MessageAttributeValue
$populationAttributeValue.DataType = "Number"
$populationAttributeValue.StringValue = "1250800"

$messageAttributes = New-Object System.Collections.Hashtable
$messageAttributes.Add("City", $cityAttributeValue)
$messageAttributes.Add("Population", $populationAttributeValue)
```

- For API details, see Publish in AWS Tools for PowerShell Cmdlet Reference.

Amazon SQS examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with Amazon SQS.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics
- Actions

Actions

Add-SQSPermission

The following code example shows how to use Add-SQSPermission.

Tools for PowerShell

Example 1: This example allows the specified AWS account to send messages from the specified queue.

Add-SQSPermission -Action SendMessage -AWSAccountID 80398EXAMPLE -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue

- For API details, see AddPermission in AWS Tools for PowerShell Cmdlet Reference.
Clear-SQSQueue

The following code example shows how to use Clear-SQSQueue.

Tools for PowerShell

Example 1: This example deletes all messages from the specified queue.

``` powershell
Clear-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- For API details, see PurgeQueue in AWS Tools for PowerShell Cmdlet Reference.

Edit-SQSMessageVisibility

The following code example shows how to use Edit-SQSMessageVisibility.

Tools for PowerShell

Example 1: This example changes the visibility timeout for the message with the specified receipt handle in the specified queue to 10 hours (10 hours * 60 minutes * 60 seconds = 36000 seconds).

``` powershell
Edit-SQSMessageVisibility -QueueUrl https://sqs.us-east-1.amazonaws.com/8039EXAMPLE/MyQueue -ReceiptHandle AQEBgGDh...J/Iqww== -VisibilityTimeout 36000
```

- For API details, see ChangeMessageVisibility in AWS Tools for PowerShell Cmdlet Reference.

Edit-SQSMessageVisibilityBatch

The following code example shows how to use Edit-SQSMessageVisibilityBatch.

Tools for PowerShell

Example 1: This example changes the visibility timeout for 2 messages with the specified receipt handles in the specified queue. The first message's visibility timeout is changed to 10 hours (10 hours * 60 minutes * 60 seconds = 36000 seconds). The second message's visibility timeout is changed to 5 hours (5 hours * 60 minutes * 60 seconds = 18000 seconds).

``` powershell
$changeVisibilityRequest1 = New-Object Amazon.SQS.Model.ChangeMessageVisibilityBatchRequestEntry
```

Amazon SQS
$changeVisibilityRequest1.Id = "Request1"
$changeVisibilityRequest1.ReceiptHandle = "AQEBd329...v6g18Q=="
$changeVisibilityRequest1.VisibilityTimeout = 36000

$changeVisibilityRequest2 = New-Object
    Amazon.SQS.Model.ChangeMessageVisibilityBatchRequestEntry
$changeVisibilityRequest2.Id = "Request2"
$changeVisibilityRequest2.ReceiptHandle = "AQEBgGDh...J/Iqw=="
$changeVisibilityRequest2.VisibilityTimeout = 18000

Edit-SQSMessageVisibilityBatch -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue -Entry $changeVisibilityRequest1,
$changeVisibilityRequest2

Output:

Failed    Successful
------    ----------
{}        {Request2, Request1}

• For API details, see ChangeMessageVisibilityBatch in AWS Tools for PowerShell Cmdlet Reference.

Get-SQSDeadLetterSourceQueue

The following code example shows how to use Get-SQSDeadLetterSourceQueue.

Tools for PowerShell

Example 1: This example lists the URLs of any queues that rely on the specified queue as their dead letter queue.

Get-SQSDeadLetterSourceQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue

Output:

https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
Get-SQSQueue

The following code example shows how to use Get-SQSQueue.

Tools for PowerShell

Example 1: This example lists all queues.

```powershell
Get-SQSQueue
```

Output:

https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/AnotherQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/DeadLetterQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue

Example 2: This example lists any queues that start with the specified name.

```powershell
Get-SQSQueue -QueueNamePrefix My
```

Output:

https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue

• For API details, see ListQueues in AWS Tools for PowerShell Cmdlet Reference.

Get-SQSQueueAttribute

The following code example shows how to use Get-SQSQueueAttribute.

Tools for PowerShell

Example 1: This example lists all attributes for the specified queue.

```powershell
```

Output:

```powershell
```

• For API details, see ListDeadLetterSourceQueues in AWS Tools for PowerShell Cmdlet Reference.
AWS Tools for PowerShell

User Guide

Get-SQSQueueAttribute -AttributeName All -QueueUrl https://sqs.useast-1.amazonaws.com/80398EXAMPLE/MyQueue

Output:
VisibilityTimeout
DelaySeconds
MaximumMessageSize
MessageRetentionPeriod
ApproximateNumberOfMessages
ApproximateNumberOfMessagesNotVisible
ApproximateNumberOfMessagesDelayed
CreatedTimestamp

:
:
:
:
:
:
:
:

30
0
262144
345600
0
0
0
2/11/2015 5:53:35 PM

LastModifiedTimestamp
QueueARN
Policy

: 12/29/2015 2:23:17 PM
: arn:aws:sqs:us-east-1:80398EXAMPLE:MyQueue
:

{"Version":"2008-10-17","Id":"arn:aws:sqs:us-east-1:80398EXAMPLE:MyQueue/
SQSDefaultPolicy","Statement":[{"Sid":"Sid14
495134224EX","Effect":"Allow","Principal":
{"AWS":"*"},"Action":"SQS:SendMessage","Resource":"arn:aws:sqs:us-east-1:80
398EXAMPLE:MyQueue","Condition":
{"Sid":
"SendMessagesFromMyQueue","Effect":"Allow","Principal":
{"AWS":"80398EXAMPLE"},"Action":"SQS:SendMessage","Resource":"
arn:aws:sqs:useast-1:80398EXAMPLE:MyQueue"}]}
Attributes
: {[QueueArn, arn:aws:sqs:useast-1:80398EXAMPLE:MyQueue], [ApproximateNumberOfMessages, 0],
[ApproximateNumberOfMessagesNotVisible, 0],
[ApproximateNumberOfMessagesDelayed, 0]...}

Example 2: This example lists separately only the speciﬁed attributes for the speciﬁed
queue.
Get-SQSQueueAttribute -AttributeName MaximumMessageSize, VisibilityTimeout -QueueUrl
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue

Output:
VisibilityTimeout
Amazon SQS

: 30
717


DelaySeconds : 0
MaximumMessageSize : 262144
MessageRetentionPeriod : 345600
ApproximateNumberOfMessages : 0
ApproximateNumberOfMessagesNotVisible : 0
ApproximateNumberOfMessagesDelayed : 0
CreatedTimestamp : 2/11/2015 5:53:35 PM
LastModifiedTimestamp : 12/29/2015 2:23:17 PM
QueueARN : arn:aws:sqs:us-east-1:80398EXAMPLE:MyQueue
Policy :
    {"Sid":"Sid14495134224EX","Effect":"Allow","Principal":
      {"AWS":"*"},"Action":"SQS:SendMessage","Resource":"arn:aws:sqs:us-east-1:80398EXAMPLE:MyQueue","Condition":
    {"Sid":"SendMessagesFromMyQueue","Effect":"Allow","Principal":
      {"AWS":"80398EXAMPLE"},"Action":"SQS:SendMessage","Resource":"arn:aws:sqs:us-
      east-1:80398EXAMPLE:MyQueue"}],
  Attributes : {[MaximumMessageSize, 262144],
  [VisibilityTimeout, 30]}

- For API details, see [GetQueueAttributes](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-SQSQueueUrl**

The following code example shows how to use Get-SQSQueueUrl.

**Tools for PowerShell**

**Example 1: This example lists the URL of the queue with the specified name.**

```
Get-SQSQueueUrl -QueueName MyQueue
```

**Output:**

https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue

- For API details, see [GetQueueUrl](#) in *AWS Tools for PowerShell Cmdlet Reference*.
New-SQSQueue

The following code example shows how to use New-SQSQueue.

Tools for PowerShell

Example 1: This example creates a queue with the specified name.

```
New-SQSQueue -QueueName MyQueue
```

Output:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- For API details, see [CreateQueue](#) in AWS Tools for PowerShell Cmdlet Reference.

Receive-SQSMessage

The following code example shows how to use Receive-SQSMessage.

Tools for PowerShell

Example 1: This example lists information for up to the next 10 messages to be received for the specified queue. The information will contain values for the specified message attributes, if they exist.

```
Receive-SQSMessage -AttributeName SenderId, SentTimestamp -MessageAttributeName StudentName, StudentGrade -MessageCount 10 -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

Output:

```
Attributes             : {
                           [SenderId, AIDAIAZKMSNQ7EXAMPLE],
                           [SentTimestamp, 1451495923744]
                        }
Body                   : Information about John Doe's grade.
MD5OfBody              : ea572796e3c231f974fe75d89EXAMPLE
MD5OfMessageAttributes : 48c1ee811f0fe7c4e88f8f5EXAMPLE
MessageAttributes      : {
                           [StudentGrade, Amazon.SQS.Model.MessageAttributeValue],
                           [StudentName, Amazon.SQS.Model.MessageAttributeValue]
                        }
MessageId              : 53828c4b-631b-469b-8833-c093cEXAMPLE
ReceiptHandle          : AQEBpfGp...20Q5cg==
```
For API details, see ReceiveMessage in AWS Tools for PowerShell Cmdlet Reference.

Remove-SQSMessage

The following code example shows how to use Remove-SQSMessage.

Tools for PowerShell

Example 1: This example deletes the message with the specified receipt handle from the specified queue.

```powershell
Remove-SQSMessage -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue -ReceiptHandle AQEBd329...v6gI8Q==
```

For API details, see DeleteMessage in AWS Tools for PowerShell Cmdlet Reference.

Remove-SQSMessageBatch

The following code example shows how to use Remove-SQSMessageBatch.

Tools for PowerShell

Example 1: This example deletes 2 messages with the specified receipt handles from the specified queue.

```powershell
$deleteMessageRequest1 = New-Object Amazon.SQS.Model.DeleteMessageBatchRequestEntry
$deleteMessageRequest1.Id = "Request1"
$deleteMessageRequest1.ReceiptHandle = "AQEBX2g4...wtJ5Qg=="

$deleteMessageRequest2 = New-Object Amazon.SQS.Model.DeleteMessageBatchRequestEntry
$deleteMessageRequest2.Id = "Request2"
$deleteMessageRequest2.ReceiptHandle = "AQEBqOVY...KTsLYg=="

Remove-SQSMessageBatch -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue -Entry $deleteMessageRequest1, $deleteMessageRequest2
```

Output:

<table>
<thead>
<tr>
<th>Failed</th>
<th>Successful</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• For API details, see `DeleteMessageBatch` in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-SQSPermission**

The following code example shows how to use `Remove-SQSPermission`.

**Tools for PowerShell**

**Example 1:** This example removes the permission settings with the specified label from the specified queue.

```
Remove-SQSPermission -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

• For API details, see `RemovePermission` in *AWS Tools for PowerShell Cmdlet Reference*.

**Remove-SQSQueue**

The following code example shows how to use `Remove-SQSQueue`.

**Tools for PowerShell**

**Example 1:** This example deletes the specified queue.

```
Remove-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

• For API details, see `DeleteQueue` in *AWS Tools for PowerShell Cmdlet Reference*.

**Send-SQSMessage**

The following code example shows how to use `Send-SQSMessage`.

**Tools for PowerShell**

**Example 1:** This example sends a message with the specified attributes and message body to the specified queue with message delivery delayed for 10 seconds.
$cityAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$cityAttributeValue.DataType = "String"
$cityAttributeValue.StringValue = "AnyCity"

$populationAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$populationAttributeValue.DataType = "Number"
$populationAttributeValue.StringValue = "1250800"

$messageAttributes = New-Object System.Collections.Hashtable
$messageAttributes.Add("City", $cityAttributeValue)
$messageAttributes.Add("Population", $populationAttributeValue)

Send-SQSMessage -DelayInSeconds 10 -MessageAttributes $messageAttributes -MessageBody "Information about the largest city in Any Region." -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue

Output:

<table>
<thead>
<tr>
<th>MD5OfMessageAttributes</th>
<th>MD5OfMessageBody</th>
<th>MessageId</th>
</tr>
</thead>
<tbody>
<tr>
<td>1d3e51347bc042efbd6dda31EXAMPLE</td>
<td>51b0a3256d59467f973009b73EXAMPLE</td>
<td>c35fed8f-c739-4d0c-818b-1820eEXAMPLE</td>
</tr>
</tbody>
</table>

- For API details, see [SendMessage](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

**Send-SQSMessageBatch**

The following code example shows how to use Send-SQSMessageBatch.

**Tools for PowerShell**

**Example 1:** This example sends 2 messages with the specified attributes and message bodies to the specified queue. Delivery is delayed for 15 seconds for the first message and 10 seconds for the second message.

@student1NameAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
@student1NameAttributeValue.DataType = "String"
@student1NameAttributeValue.StringValue = "John Doe"
$student1GradeAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$student1GradeAttributeValue.DataType = "Number"
$student1GradeAttributeValue.StringValue = "89"

@student2NameAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
@student2NameAttributeValue.DataType = "String"
@student2NameAttributeValue.StringValue = "Jane Doe"

@student2GradeAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
@student2GradeAttributeValue.DataType = "Number"
@student2GradeAttributeValue.StringValue = "93"

$message1 = New-Object Amazon.SQS.Model.SendMessageBatchRequestEntry
$message1.DelaySeconds = 15
$message1.Id = "FirstMessage"
$message1.MessageAttributes.Add("StudentName", $student1NameAttributeValue)
$message1.MessageAttributes.Add("StudentGrade", $student1GradeAttributeValue)
$message1.MessageBody = "Information about John Doe's grade."

$message2 = New-Object Amazon.SQS.Model.SendMessageBatchRequestEntry
$message2.DelaySeconds = 10
$message2.Id = "SecondMessage"
$message2.MessageAttributes.Add("StudentName", $student2NameAttributeValue)
$message2.MessageAttributes.Add("StudentGrade", $student2GradeAttributeValue)
$message2.MessageBody = "Information about Jane Doe's grade."

Send-SQSMessageBatch -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue -Entry $message1, $message2

Output:

<table>
<thead>
<tr>
<th>Failed</th>
<th>Successful</th>
</tr>
</thead>
<tbody>
<tr>
<td>------</td>
<td>----------</td>
</tr>
<tr>
<td>{}</td>
<td>{FirstMessage, SecondMessage}</td>
</tr>
</tbody>
</table>

- For API details, see [SendMessageBatch](https://docs.aws.amazon.com/sdk-f KobobGuidelines/PowerShell/latest/reference/complex-type-SendMessageBatch.html) in *AWS Tools for PowerShell Cmdlet Reference*.

**Set-SQSQueueAttribute**

The following code example shows how to use Set-SQSQueueAttribute.
Example 1: This example shows how to set a policy subscribing a queue to an SNS topic. When a message is published to the topic, a message is sent to the subscribed queue.

```powershell
# create the queue and topic to be associated
$qurl = New-SQSQueue -QueueName "myQueue"
$topicarn = New-SNSTopic -Name "myTopic"

# get the queue ARN to inject into the policy; it will be returned
# in the output's QueueARN member but we need to put it into a variable
# so text expansion in the policy string takes effect
$qarn = (Get-SQSQueueAttribute -QueueUrl $qurl -AttributeName "QueueArn").QueueARN

# construct the policy and inject arns
$policy = @
{
    "Version": "2008-10-17",
    "Id": "$qarn/SQSPOLICY",
    "Statement": [
        {
            "Sid": "1",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "SQS:SendMessage",
            "Resource": "$qarn",
            "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": "$topicarn"
                }
            }
        }
    ]
}

# set the policy
Set-SQSQueueAttribute -QueueUrl $qurl -Attribute @{ Policy=$policy }
```

Example 2: This example sets the specified attributes for the specified queue.

```powershell
Set-SQSQueueAttribute -Attribute @{"DelaySeconds" = "10"; "MaximumMessageSize" = "131072"} -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```
For API details, see SetQueueAttributes in AWS Tools for PowerShell Cmdlet Reference.

**AWS STS examples using Tools for PowerShell**

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with AWS STS.

*Actions* are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

*Scenarios* are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

**Topics**

- **Actions**

**Actions**

**Convert-STSAuthorizationMessage**

The following code example shows how to use Convert-STSAuthorizationMessage.

**Tools for PowerShell**

Example 1: Decodes the additional information contained in the supplied encoded message content that was returned in response to a request. The additional information is encoded because details of the authorization status can constitute privileged information that the user who requested the action should not see.

Convert-STSAuthorizationMessage -EncodedMessage "...encoded message..."

For API details, see DecodeAuthorizationMessage in AWS Tools for PowerShell Cmdlet Reference.
Get-STSFederationToken

The following code example shows how to use Get-STSFederationToken.

Tools for PowerShell

Example 1: Requests a federated token valid for one hour using "Bob" as the name of the federated user. This name can be used to reference the federated user name in a resource-based policy (such as an Amazon S3 bucket policy). The supplied IAM policy, in JSON format, is used to scope down the permissions that are available to the IAM user. The supplied policy cannot grant more permissions than those granted to the requesting user, with the final permissions for the federated user being the most restrictive set based on the intersection of the passed policy and the IAM user policy.

```
Get-STSFederationToken -Name "Bob" -Policy "...JSON policy..." -DurationInSeconds 3600
```

- For API details, see GetFederationToken in AWS Tools for PowerShell Cmdlet Reference.

Get-STSSessionToken

The following code example shows how to use Get-STSSessionToken.

Tools for PowerShell

Example 1: Returns an Amazon.RuntimeAWSCredentials instance containing temporary credentials valid for a set period of time. The credentials used to request temporary credentials are inferred from the current shell defaults. To specify other credentials, use the -ProfileName or -AccessKey/-SecretKey parameters.

```
Get-STSSessionToken
```

Output:

<table>
<thead>
<tr>
<th>AccessKeyId</th>
<th>Expiration</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXAMPLEACCESSKEYID</td>
<td>2/16/2015 9:12:28 PM</td>
</tr>
<tr>
<td>examplesecretaccesskey...</td>
<td>SampleToken...</td>
</tr>
</tbody>
</table>
Example 2: Returns an Amazon.RuntimeAWSCredentials instance containing temporary credentials valid for one hour. The credentials used to make the request are obtained from the specified profile.

Get-STSSessionToken -DurationInSeconds 3600 -ProfileName myprofile

Output:

<table>
<thead>
<tr>
<th>AccessKeyId</th>
<th>Expiration</th>
</tr>
</thead>
<tbody>
<tr>
<td>SecretAccessKey</td>
<td>SessionToken</td>
</tr>
<tr>
<td>EXAMPLEACCESSKEYID</td>
<td>2/16/2015 9:12:28 PM</td>
</tr>
<tr>
<td>examplesecretaccesskey...</td>
<td>SamPleTokeN.....</td>
</tr>
</tbody>
</table>

Example 3: Returns an Amazon.RuntimeAWSCredentials instance containing temporary credentials valid for one hour using the identification number of the MFA device associated with the account whose credentials are specified in the profile 'myprofilename' and the value provided by the device.

Get-STSSessionToken -DurationInSeconds 3600 -ProfileName myprofile -SerialNumber YourMFADeviceSerialNumber -TokenCode 123456

Output:

<table>
<thead>
<tr>
<th>AccessKeyId</th>
<th>Expiration</th>
</tr>
</thead>
<tbody>
<tr>
<td>SecretAccessKey</td>
<td>SessionToken</td>
</tr>
<tr>
<td>EXAMPLEACCESSKEYID</td>
<td>2/16/2015 9:12:28 PM</td>
</tr>
<tr>
<td>examplesecretaccesskey...</td>
<td>SamPleTokeN.....</td>
</tr>
</tbody>
</table>

- For API details, see [GetSessionToken](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

**Use-STSRole**

The following code example shows how to use Use-STSRole.
Tools for PowerShell

Example 1: Returns a set of temporary credentials (access key, secret key and session token) that can be used for one hour to access AWS resources that the requesting user might not normally have access to. The returned credentials have the permissions that are allowed by the access policy of the role being assumed and the policy that was supplied (you cannot use the supplied policy to grant permissions in excess of those defined by the access policy of the role being assumed).

Use-STSRole -RoleSessionName "Bob" -RoleArn "arn:aws:iam::123456789012:role/demo" -Policy "...JSON policy..." -DurationInSeconds 3600

Example 2: Returns a set of temporary credentials, valid for one hour, that have the same permissions that are defined in the access policy of the role being assumed.

Use-STSRole -RoleSessionName "Bob" -RoleArn "arn:aws:iam::123456789012:role/demo" -DurationInSeconds 3600

Example 3: Returns a set of temporary credentials supplying the serial number and generated token from an MFA associated with the user credentials used to execute the cmdlet.

Use-STSRole -RoleSessionName "Bob" -RoleArn "arn:aws:iam::123456789012:role/demo" -DurationInSeconds 3600 -SerialNumber "GAHT12345678" -TokenCode "123456"

Example 4: Returns a set of temporary credentials that have assumed a role defined in a customer account. For each role that the third party can assume, the customer account must create a role using an identifier that must be passed in the -ExternalId parameter each time the role is assumed.

Use-STSRole -RoleSessionName "Bob" -RoleArn "arn:aws:iam::123456789012:role/demo" -DurationInSeconds 3600 -ExternalId "ABC123"

- For API details, see AssumeRole in AWS Tools for PowerShell Cmdlet Reference.

Use-STSWebIdentityRole

The following code example shows how to use Use-STSWebIdentityRole.
Tools for PowerShell

Example 1: Returns a temporary set of credentials, valid for one hour, for a user who has been authenticated with the Login with Amazon identity provider. The credentials assume the access policy associated with the role identified by the role ARN. Optionally, you can pass a JSON policy to the -Policy parameter that further refine the access permissions (you cannot grant more permissions than are available in the permissions associated with the role). The value supplied to the -WebIdentityToken is the unique user identifier that was returned by the identity provider.

```
Use-STSWebIdentityRole -DurationInSeconds 3600 -ProviderId "www.amazon.com" -RoleSessionName "app1" -RoleArn "arn:aws:iam::123456789012:role/FederatedWebIdentityRole" -WebIdentityToken "Atza...DVI0r1"
```

- For API details, see AssumeRoleWithWebIdentity in AWS Tools for PowerShell Cmdlet Reference.

AWS Support examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with AWS Support.

*Actions* are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

*Scenarios* are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

**Topics**

- Actions
Actions

Add-ASACommunicationToCase

The following code example shows how to use Add-ASACommunicationToCase.

Tools for PowerShell

Example 1: Adds the body of an email communication to the specified case.

Add-ASACommunicationToCase -CaseId "case-12345678910-2013-c4c1d2bf33c5cf47" -CommunicationBody "Some text about the case"

Example 2: Adds the body of an email communication to the specified case plus one or more email addresses contained in the CC line of the email.

Add-ASACommunicationToCase -CaseId "case-12345678910-2013-c4c1d2bf33c5cf47" -CcEmailAddress @("email1@address.com", "email2@address.com") -CommunicationBody "Some text about the case"

- For API details, see AddCommunicationToCase in AWS Tools for PowerShell Cmdlet Reference.

Get-ASACase

The following code example shows how to use Get-ASACase.

Tools for PowerShell

Example 1: Returns the details of all support cases.

Get-ASACase

Example 2: Returns the details of all support cases since the specified date and time.

Get-ASACase -AfterTime "2013-09-10T03:06Z"

Example 3: Returns the details of the first 10 support cases, including those that have been resolved.

Get-ASACase -MaxResult 10 -IncludeResolvedCases $true
Example 4: Returns the details of the single specified support case.

Get-ASACase -CaseIdList "case-12345678910-2013-c4c1d2bf33c5cf47"

Example 5: Returns the details of specified support cases.

Get-ASACase -CaseIdList @("case-12345678910-2013-c4c1d2bf33c5cf47", "case-18929034710-2011-c4fdeabf33c5cf47")

Example 6: Returns all support cases using manual paging. The cases are retrieved in batches of 20.

$nextToken = $null
do {
    Get-ASACase -NextToken $nextToken -MaxResult 20
    $nextToken = $AWSHistory.LastServiceResponse.NextToken
} while ($nextToken -ne $null)

- For API details, see [DescribeCases](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-ASACommunication

The following code example shows how to use Get-ASACommunication.

Tools for PowerShell

Example 1: Returns all communications for the specified case.

Get-ASACommunication -CaseId "case-12345678910-2013-c4c1d2bf33c5cf47"

Example 2: Returns all communications since midnight UTC on January 1st 2012 for the specified case.

Get-ASACommunication -CaseId "case-12345678910-2013-c4c1d2bf33c5cf47" -AfterTime "2012-01-10T00:00Z"

Example 3: Returns all communications since midnight UTC on January 1st 2012 for the specified case, using manual paging. The communications are retrieved in batches of 20.

$nextToken = $null
do {
    Get-ASACommunication -CaseId "case-12345678910-2013-c4c1d2bf33c5cf47" -NextToken $nextToken -MaxResult 20
    $nextToken = $AWSHistory.LastServiceResponse.NextToken
} while ($nextToken -ne $null)

- For API details, see `DescribeCommunications` in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-ASAService**

The following code example shows how to use Get-ASAService.

**Tools for PowerShell**

**Example 1:** Returns all available service codes, names and categories.

```powershell
Get-ASAService
```

**Example 2:** Returns the name and categories for the service with the specified code.

```powershell
Get-ASAService -ServiceCodeList "amazon-cloudfront"
```

**Example 3:** Returns the name and categories for the specified service codes.

```powershell
Get-ASAService -ServiceCodeList @("amazon-cloudfront", "amazon-cloudwatch")
```

**Example 4:** Returns the name and categories (in Japanese) for the specified service codes. Currently English ("en") and Japanese ("ja") language codes are supported.

```powershell
Get-ASAService -ServiceCodeList @("amazon-cloudfront", "amazon-cloudwatch") -Language "ja"
```

- For API details, see `DescribeServices` in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-ASASeverityLevel**

The following code example shows how to use Get-ASASeverityLevel.

**Tools for PowerShell**

**Example 1:** Returns the list of severity levels that can be assigned to an AWS Support case.
Get-ASASeverityLevel

Example 2: Returns the list of severity levels that can be assigned to an AWS Support case. The names of the levels are returned in Japanese.

Get-ASASeverityLevel -Language "ja"

For API details, see DescribeSeverityLevels in AWS Tools for PowerShell Cmdlet Reference.

Get-ASATrustedAdvisorCheck

The following code example shows how to use Get-ASATrustedAdvisorCheck.

Tools for PowerShell

Example 1: Returns the collection of Trusted Advisor checks. You must specify the Language parameter which can accept either "en" for English output or "ja" for Japanese output.

Get-ASATrustedAdvisorCheck -Language "en"

For API details, see DescribeTrustedAdvisorChecks in AWS Tools for PowerShell Cmdlet Reference.

Get-ASATrustedAdvisorCheckRefreshStatus

The following code example shows how to use Get-ASATrustedAdvisorCheckRefreshStatus.

Tools for PowerShell

Example 1: Returns the current status of refresh requests for the specified checks. RequestASATrustedAdvisorCheckRefresh can be used to request that the status information of the checks be refreshed.

Get-ASATrustedAdvisorCheckRefreshStatus -CheckId @("checkid1", "checkid2")

For API details, see DescribeTrustedAdvisorCheckRefreshStatuses in AWS Tools for PowerShell Cmdlet Reference.
Get-ASATrustedAdvisorCheckResult

The following code example shows how to use Get-ASATrustedAdvisorCheckResult.

Tools for PowerShell

Example 1: Returns the results of a Trusted Advisor check. The list of available Trusted Advisor checks can be obtained using Get-ASATrustedAdvisorChecks. The output is the overall status of the check, the timestamp at which the check was last run and the unique checkid for the specific check. To have the results output in Japanese, add the -Language "ja" parameter.

Get-ASATrustedAdvisorCheckResult -CheckId "checkid1"

- For API details, see DescribeTrustedAdvisorCheckResult in AWS Tools for PowerShell Cmdlet Reference.

Get-ASATrustedAdvisorCheckSummary

The following code example shows how to use Get-ASATrustedAdvisorCheckSummary.

Tools for PowerShell

Example 1: Returns the latest summary for the specified Trusted Advisor check.

Get-ASATrustedAdvisorCheckSummary -CheckId "checkid1"

Example 2: Returns the latest summaries for the specified Trusted Advisor checks.

Get-ASATrustedAdvisorCheckSummary -CheckId @("checkid1", "checkid2")

- For API details, see DescribeTrustedAdvisorCheckSummaries in AWS Tools for PowerShell Cmdlet Reference.

New-ASACase

The following code example shows how to use New-ASACase.
Tools for PowerShell

Example 1: Creates a new case in the AWS Support Center. Values for the -ServiceCode and -CategoryCode parameters can be obtained using the Get-ASAService cmdlet. The value for the -SeverityCode parameter can be obtained using the Get-ASASeverityLevel cmdlet. The -IssueType parameter value can be either "customer-service" or "technical". If successful the AWS Support case number is output. By default the case will be handled in English, to use Japanese add the -Language "ja" parameter. The -ServiceCode, -CategoryCode, -Subject and -CommunicationBody parameters are mandatory.

```
New-ASACase -ServiceCode "amazon-cloudfront" -CategoryCode "APIs" -SeverityCode "low" -Subject "subject text" -CommunicationBody "description of the case" -CcEmailAddress @("email1@domain.com", "email2@domain.com") -IssueType "technical"
```

- For API details, see [CreateCase](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Request-ASATrustedAdvisorCheckRefresh

The following code example shows how to use Request-ASATrustedAdvisorCheckRefresh.

Tools for PowerShell

Example 1: Requests a refresh for the specified Trusted Advisor check.

```
Request-ASATrustedAdvisorCheckRefresh -CheckId "checkid1"
```

- For API details, see [RefreshTrustedAdvisorCheck](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Resolve-ASACase

The following code example shows how to use Resolve-ASACase.

Tools for PowerShell

Example 1: Returns the initial state of the specified case and the current state after the call to resolve it is completed.

```
Resolve-ASACase -CaseId "case-12345678910-2013-c4c1d2bf33c5cf47"
```
For API details, see ResolveCase in AWS Tools for PowerShell Cmdlet Reference.

Systems Manager examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with Systems Manager.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

- Actions

Actions

Add-SSMResourceTag

The following code example shows how to use Add-SSMResourceTag.

Tools for PowerShell

Example 1: This example updates a maintenance window with new tags. There is no output if the command succeeds. The syntax used by this example requires PowerShell version 3 or later.

```powershell
$options1 = @{Key="Stack";Value=(@("Production"))}
Add-SSMResourceTag -ResourceId "mw-03eb9db42890fb82d" -ResourceType "MaintenanceWindow" -Tag $options1
```

Example 2: With PowerShell version 2, you must use New-Object to create each tag. There is no output if the command succeeds.

```powershell
$tag1 = New-Object Amazon.SimpleSystemsManagement.Model.Tag
```

736
```
$tag1.Key = "Stack"
$tag1.Value = "Production"

Add-SSMResourceTag -ResourceId "mw-03eb9db42890fb82d" -ResourceType "MaintenanceWindow" -Tag $tag1
```

- For API details, see AddTagsToResource in AWS Tools for PowerShell Cmdlet Reference.

**Edit-SSMDocumentPermission**

The following code example shows how to use Edit-SSMDocumentPermission.

**Tools for PowerShell**

**Example 1:** This example adds "share" permissions to all accounts for a document. There is no output if the command succeeds.

```
Edit-SSMDocumentPermission -Name "RunShellScript" -PermissionType "Share" -AccountIdsToAdd all
```

**Example 2:** This example adds "share" permissions to a specific account for a document. There is no output if the command succeeds.

```
Edit-SSMDocumentPermission -Name "RunShellScriptNew" -PermissionType "Share" -AccountIdsToAdd "123456789012"
```

- For API details, see ModifyDocumentPermission in AWS Tools for PowerShell Cmdlet Reference.

**Get-SSMAactivation**

The following code example shows how to use Get-SSMAactivation.

**Tools for PowerShell**

**Example 1:** This example provides details about the activations on your account.

```
Get-SSMAactivation
```

**Output:**


Get-SSMAssociation

The following code example shows how to use Get-SSMAssociation.

Tools for PowerShell

Example 1: This example describes the association between an instance and a document.

Get-SSMAssociation -InstanceId "i-0000293ffd8c57862" -Name "AWS-UpdateSSMAgent"

Output:

Name                  : AWS-UpdateSSMAgent
InstanceId            : i-0000293ffd8c57862
Date                  : 2/23/2017 6:55:22 PM
Status.Name           : Pending
Status.Date           : 2/20/2015 8:31:11 AM
Status.Message        : temp_status_change
Status.AdditionalInfo : Additional-Config-Needed

• For API details, see DescribeAssociation in AWS Tools for PowerShell Cmdlet Reference.

Get-SSMAssociationExecution

The following code example shows how to use Get-SSMAssociationExecution.

Tools for PowerShell

Example 1: This example returns the executions for the association ID provided
Get-SSMAssociationExecution -AssociationId 123a45a0-c678-9012-3456-78901234db5e

Output:

AssociationId : 123a45a0-c678-9012-3456-78901234db5e
AssociationVersion : 2
CreateTime : 3/2/2019 8:53:29 AM
DetailedStatus :
ExecutionId : 123a45a0-c678-9012-3456-78901234db5e
LastExecutionDate : 1/1/0001 12:00:00 AM
ResourceCountByStatus : {Success=4}
Status : Success

- For API details, see [DescribeAssociationExecutions](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

Get-SSMAssociationExecutionTarget

The following code example shows how to use Get-SSMAssociationExecutionTarget.

**Tools for PowerShell**

**Example 1**: This example displays the resource ID and its execution status that are part of the association execution targets

Get-SSMAssociationExecutionTarget -AssociationId 123a45a0-c678-9012-3456-78901234db5e -ExecutionId 123a45a0-c678-9012-3456-78901234db5e | Select-Object ResourceId, Status

Output:

<table>
<thead>
<tr>
<th>ResourceId</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>i-0b1b2a3456f7a890b</td>
<td>Success</td>
</tr>
<tr>
<td>i-01c12a45d6fc7a89f</td>
<td>Success</td>
</tr>
<tr>
<td>i-0a1caf234f56d7dc8</td>
<td>Success</td>
</tr>
<tr>
<td>i-012a3fd45af6dbcfe</td>
<td>Failed</td>
</tr>
<tr>
<td>i-0ddc1df23c4a5fb67</td>
<td>Success</td>
</tr>
</tbody>
</table>
Example 2: This command checks the particular execution of a particular automation since yesterday, where a command document is associated. It further checks if the association execution failed, and if so, it will display the command invocation details for the execution along with the instance id.

```powershell
$AssociationExecution = Get-SSMAssociationExecutionTarget -AssociationId 1c234567-890f-1aca-a234-5a678d901cb0 -ExecutionId 12345ca12-3456-2345-2b45-23456789012 | Where-Object {$_ LastExecutionDate -gt (Get-Date -Hour 00 -Minute 00).AddDays(-1)}

foreach ($execution in $AssociationExecution) {
    if($execution.Status -ne 'Success'){
        Write-Output "There was an issue executing the association $($execution.AssociationId) on $($execution.ResourceId)"
        Get-SSMCommandInvocation -CommandId $execution.OutputSource.OutputSourceId -Detail:$true | Select-Object -ExpandProperty CommandPlugins
    }
}
```

Output:

```
There was an issue executing the association 1c234567-890f-1aca-a234-5a678d901cb0 on i-0a1caf234f56d7dc8

Name               : aws:runPowerShellScript
Output             :
                   ----------ERROR----------
                   failed to run commands: exit status 1
OutputS3BucketName :
OutputS3KeyPrefix  :
OutputS3Region     : eu-west-1
ResponseCode       : 1
ResponseFinishDateTime : 5/29/2019 11:04:49 AM
ResponseStartDateTime : 5/29/2019 11:04:49 AM
StandardErrorUrl   :
StandardOutputUrl  :
Status             : Failed
StatusDetails      : Failed
```
For API details, see `DescribeAssociationExecutionTargets` in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-SSMAssociationList

The following code example shows how to use `Get-SSMAssociationList`.

#### Tools for PowerShell

**Example 1**: This example lists all the associations for an instance. The syntax used by this example requires PowerShell version 3 or later.

```powershell
$filter1 = @{Key="InstanceId";Value=@("i-0000293ffd8c57862")}
Get-SSMAssociationList -AssociationFilterList $filter1
```

**Output**:

```
AssociationId : d8617c07-2079-4c18-9847-1655fc2698b0
DocumentVersion :  
InstanceId : i-0000293ffd8c57862
LastExecutionDate : 2/20/2015 8:31:11 AM
Name : AWS-UpdateSSMAgent
Overview : Amazon.SimpleSystemsManagement.Model.AssociationOverview
ScheduleExpression : 
Targets : {InstanceIds}
```

**Example 2**: This example lists all associations for a configuration document. The syntax used by this example requires PowerShell version 3 or later.

```powershell
$filter2 = @{Key="Name";Value=@("AWS-UpdateSSMAgent")}
Get-SSMAssociationList -AssociationFilterList $filter2
```

**Output**:

```
AssociationId : d8617c07-2079-4c18-9847-1655fc2698b0
DocumentVersion :  
InstanceId : i-0000293ffd8c57862
LastExecutionDate : 2/20/2015 8:31:11 AM
Name : AWS-UpdateSSMAgent
Overview : Amazon.SimpleSystemsManagement.Model.AssociationOverview
ScheduleExpression : 
```
Example 3: With PowerShell version 2, you must use New-Object to create each filter.

```powershell
$filter1 = New-Object Amazon.SimpleSystemsManagement.Model.AssociationFilter
$filter1.Key = "InstanceId"
$filter1.Value = "i-0000293ffd8c57862"

Get-SSMAssociationList -AssociationFilterList $filter1
```

Output:

```
AssociationId      : d8617c07-2079-4c18-9847-1655fc2698b0
DocumentVersion    :
InstanceId         : i-0000293ffd8c57862
LastExecutionDate  : 2/20/2015 8:31:11 AM
Name               : AWS-UpdateSSMAgent
Overview           : Amazon.SimpleSystemsManagement.Model.AssociationOverview
ScheduleExpression :
Targets            : {InstanceIds}
```

- For API details, see [ListAssociations](https://aws.amazon.com) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-SSMAssociationVersionList**

The following code example shows how to use Get-SSMAssociationVersionList.

**Tools for PowerShell**

**Example 1: This example retrieves all versions of the association provided.**

```powershell
Get-SSMAssociationVersionList -AssociationId 123a45a0-c678-9012-3456-78901234db5e
```

**Output:**

```
AssociationId      : 123a45a0-c678-9012-3456-78901234db5e
AssociationName    :
AssociationVersion : 2
ComplianceSeverity :
CreatedDate        : 3/12/2019 9:21:01 AM
DocumentVersion    :
```
MaxConcurrency : 
MaxErrors : 
Name : AWS-GatherSoftwareInventory 
OutputLocation : 
Parameters : {} 
ScheduleExpression : 
Targets : {InstanceIds}

AssociationId : 123a45a0-c678-9012-3456-78901234db5e 
AssociationName : test-case-1234567890 
AssociationVersion : 1 
ComplianceSeverity : 
CreatedDate : 3/2/2019 8:53:29 AM 
DocumentVersion : 
MaxConcurrency : 
MaxErrors : 
Name : AWS-GatherSoftwareInventory 
OutputLocation : 
Parameters : {} 
ScheduleExpression : rate(30minutes) 
Targets : {InstanceIds}

• For API details, see ListAssociationVersions in AWS Tools for PowerShell Cmdlet Reference.

Get-SSMAutomationExecution

The following code example shows how to use Get-SSMAutomationExecution.

Tools for PowerShell

Example 1: This example displays the details of an Automation Execution.

Get-SSMAutomationExecution -AutomationExecutionId "4105a4fc-f944-11e6-9d32-8fb2db27a909"

Output:

AutomationExecutionId : 4105a4fc-f944-11e6-9d32-8fb2db27a909
AutomationExecutionStatus : Failed
DocumentName : AWS-UpdateLinuxAmi
DocumentVersion : 1
ExecutionEndTime : 2/22/2017 9:17:08 PM
ExecutionStartTime : 2/22/2017 9:17:02 PM
FailureMessage : Step launchInstance failed maximum allowed times. You are not authorized to perform this operation. Encoded authorization failure message: 
B_V2QyyN7NhSZQYpmVzpEc4oSnj2GLTNyXUHuTbq3kNMoDgubmbtthLmZyaiUYekORIrA42-fvIx-04q5Fjff6glh
Yb6TISb0GQeeNrpwNvpDzm0-
PSR1sw1Abg9fdM9BcNjyiznspUkwPwKu9EC10u6v30XU1KC9nZ7mPlWMFZNkSioQqpwWEvMw-GZktsQzm67qOhUkBONLWYhbS
pkfizqY-5nw3S0obx30fhd3EJa50_-
GjV_a0nFXQJa70ik40bF0rEh3MtCSbrQT6--DvFy_FQ8TKvkIXadyVskeJI84XOF5WmA60f1p5I08i-nRfZS6oDeU
gELBjjoFKD8s3L2ai0B6umWVxq0jqhQRxwJ53s54sZJ2PW3v_mtg9-q0CK0ezS3xfh_y0ilaUGOA2G-xjQFuvU_JZedWpla3xi-MZsmblAifBI
(Service: AmazonEC2; Status Code: 403; Error Code: UnauthorizedOperation; Request ID: 6a002f94-ba37-43fd-99e6-395177515fce5)
Outputs : {[createImage.ImageId,
Amazon.Runtime.Internal.Util.AlwaysSendList`1[System.String]]}
Parameters : {[AutomationAssumeRole,
Amazon.Runtime.Internal.Util.AlwaysSendList`1[System.String]], [InstanceIamRole,
Amazon.Runtime.Internal.Util.AlwaysSendList`1[System.String]], [SourceAmiId,
Amazon.Runtime.Internal.Util.AlwaysSendList`1[System.String]]}
StepExecutions : {launchInstance, updateOSSoftware, stopInstance,
createImage...}

Example 2: This example lists step details for the given automation execution id

Get-SSMAutomationExecution -AutomationExecutionId e1d2bad3-4567-8901-ae23-456c7c8901be | Select-Object -ExpandProperty StepExecutions | Select-Object StepName, Action, StepStatus, ValidNextSteps

Output:

<table>
<thead>
<tr>
<th>StepName</th>
<th>Action</th>
<th>StepStatus</th>
<th>ValidNextSteps</th>
</tr>
</thead>
<tbody>
<tr>
<td>LaunchInstance</td>
<td>aws:runInstances</td>
<td>Success</td>
<td>{OSCompatibilityCheck}</td>
</tr>
<tr>
<td>OSCompatibilityCheck</td>
<td>aws:runCommand</td>
<td>Success</td>
<td>{RunPreUpdateScript}</td>
</tr>
<tr>
<td>RunPreUpdateScript</td>
<td>aws:runCommand</td>
<td>Success</td>
<td>{UpdateEC2Config}</td>
</tr>
<tr>
<td>UpdateEC2Config</td>
<td>aws:runCommand</td>
<td>Cancelled</td>
<td>{}</td>
</tr>
<tr>
<td>UpdateSSMAgent</td>
<td>aws:runCommand</td>
<td>Pending</td>
<td>{}</td>
</tr>
</tbody>
</table>
AWS Tools for PowerShell

UpdateAWSPVDriver         aws:runCommand          Pending    {}
UpdateAWSEnaNetworkDriver aws:runCommand          Pending    {}
UpdateAWSNVMe             aws:runCommand          Pending    {}
InstallWindowsUpdates     aws:runCommand          Pending    {}
RunPostUpdateScript       aws:runCommand          Pending    {}
RunSysprepGeneralize      aws:runCommand          Pending    {}
StopInstance              aws:changeInstanceState Pending    {}
CreateImage               aws:createImage         Pending    {}
TerminateInstance         aws:changeInstanceState Pending    {}

For API details, see [GetAutomationExecution](#) in AWS Tools for PowerShell Cmdlet Reference.

Get-SSMAutomationExecutionList

The following code example shows how to use Get-SSMAutomationExecutionList.

Tools for PowerShell

**Example 1:** This example describes all active and terminated Automation Executions associated with your account.

```powershell
Get-SSMAutomationExecutionList
```

**Output:**

```powershell
AutomationExecutionId     : 4105a4fc-f944-11e6-9d32-8fb2db27a909
AutomationExecutionStatus : Failed
DocumentName              : AWS-UpdateLinuxAmi
DocumentVersion           : 1
ExecutedBy                : admin
ExecutionEndTime          : 2/22/2017 9:17:08 PM
ExecutionStartTime        : 2/22/2017 9:17:02 PM
LogFile                   :
Outputs                   : {
                           
```
```powershell

**Example 2:** This example displays ExecutionID, document, execution start/end timestamp for executions with AutomationExecutionStatus other than 'Success'

```powershell
Get-SSMAutomationExecutionList | Where-Object AutomationExecutionStatus -ne "Success" | Select-Object AutomationExecutionId, DocumentName,
```
Output:

<table>
<thead>
<tr>
<th>AutomationExecutionId</th>
<th>DocumentName</th>
<th>AutomationExecutionStatus</th>
<th>ExecutionStartTime</th>
<th>ExecutionEndTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1d2bad3-4567-8901-ae23-456c7c8901be</td>
<td>AWS-UpdateWindowsAmi</td>
<td>Cancelled</td>
<td>4/16/2019 5:37:04 AM</td>
<td>4/16/2019 5:47:29 AM</td>
</tr>
<tr>
<td>61234567-a7f8-90e1-2b34-567b8bf9012c</td>
<td>Fixed-UpdateAmi</td>
<td>Cancelled</td>
<td>4/16/2019 5:33:04 AM</td>
<td>4/16/2019 5:40:15 AM</td>
</tr>
<tr>
<td>91234d56-7e89-0ac1-2aee-34ea5d6a7c89</td>
<td>AWS-UpdateWindowsAmi</td>
<td>Failed</td>
<td>4/16/2019 5:22:46 AM</td>
<td>4/16/2019 5:27:29 AM</td>
</tr>
</tbody>
</table>

For API details, see [DescribeAutomationExecutions](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

**Get-SSMAutomationStepExecution**

The following code example shows how to use `Get-SSMAutomationStepExecution`.

**Tools for PowerShell**

**Example 1:** This example displays information about all active and terminated step executions in an Automation workflow.

```powershell
Get-SSMAutomationStepExecution -AutomationExecutionId e1d2bad3-4567-8901-ae23-456c7c8901be | Select-Object StepName, Action, StepStatus
```

**Output:**

<table>
<thead>
<tr>
<th>StepName</th>
<th>Action</th>
<th>StepStatus</th>
</tr>
</thead>
<tbody>
<tr>
<td>LaunchInstance</td>
<td>aws:runInstances</td>
<td>Success</td>
</tr>
<tr>
<td>OSCompatibilityCheck</td>
<td>aws:runCommand</td>
<td>Success</td>
</tr>
<tr>
<td>RunPreUpdateScript</td>
<td>aws:runCommand</td>
<td>Success</td>
</tr>
<tr>
<td>UpdateEC2Config</td>
<td>aws:runCommand</td>
<td>Cancelled</td>
</tr>
<tr>
<td>UpdateSSMAgent</td>
<td>aws:runCommand</td>
<td>Pending</td>
</tr>
<tr>
<td>UpdateAWSPVDriver</td>
<td>aws:runCommand</td>
<td>Pending</td>
</tr>
</tbody>
</table>
For API details, see DescribeAutomationStepExecutions in AWS Tools for PowerShell Cmdlet Reference.

Get-SSMAvailablePatch

The following code example shows how to use Get-SSMAvailablePatch.

Tools for PowerShell

Example 1: This example gets all available patches for Windows Server 2012 that have a MSRC severity of Critical. The syntax used by this example requires PowerShell version 3 or later.

```powershell
$filter1 = @{Key="PRODUCT";Values= @("WindowsServer2012")}
$filter2 = @{Key="MSRC_SEVERITY";Values= @("Critical")}
Get-SSMAvailablePatch -Filter $filter1,$filter2
```

Output:

<table>
<thead>
<tr>
<th>Classification</th>
<th>SecurityUpdates</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContentUrl</td>
<td><a href="https://support.microsoft.com/en-us/kb/2727528">https://support.microsoft.com/en-us/kb/2727528</a></td>
</tr>
<tr>
<td>Description</td>
<td>A security issue has been identified that could allow an unauthenticated remote attacker to compromise your system and gain control over it. You can help protect your system by installing this update from Microsoft. After you install this update, you may have to restart your system.</td>
</tr>
<tr>
<td>Id</td>
<td>1eb507be-2040-4eeb-803d-abc55700b715</td>
</tr>
<tr>
<td>KbNumber</td>
<td>KB2727528</td>
</tr>
<tr>
<td>Language</td>
<td>All</td>
</tr>
<tr>
<td>MsrscNumber</td>
<td>MS12-072</td>
</tr>
<tr>
<td>MsrscSeverity</td>
<td>Critical</td>
</tr>
<tr>
<td>Product</td>
<td>WindowsServer2012</td>
</tr>
</tbody>
</table>
Example 2: With PowerShell version 2, you must use New-Object to create each filter.

```powershell
$filter1 = New-Object Amazon.SimpleSystemsManagement.Model.PatchOrchestratorFilter
$filter1.Key = "PRODUCT"
$filter1.Values = "WindowsServer2012"
$filter2.Key = "MSRC_SEVERITY"
$filter2.Values = "Critical"

Get-SSMAvailablePatch -Filter $filter1,$filter2
```

Example 3: This example fetches all the updates which are released in last 20 days and applicable to products matching WindowsServer2019

```powershell
```

Output:

<table>
<thead>
<tr>
<th>ReleaseDate</th>
<th>Product</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>4/2/2019 5:00:06 PM</td>
<td>WindowsServer2019</td>
<td>2019-03 Servicing Stack Update for Windows Server 2019 for x64-based Systems (KB4493510)</td>
</tr>
</tbody>
</table>

- For API details, see DescribeAvailablePatches in AWS Tools for PowerShell Cmdlet Reference.

Get-SSMCommand

The following code example shows how to use Get-SSMCommand.
Tools for PowerShell

Example 1: This example lists all commands requested.

Get-SSMCommand

Output:

<table>
<thead>
<tr>
<th>CommandId</th>
<th>4b75a163-d39a-4d97-87c9-98ae52c6be35</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment</td>
<td>Apply association with id at update time: 4cc73e42-d5ae-4879-84f8-57e09c0efcd0</td>
</tr>
<tr>
<td>CompletedCount</td>
<td>1</td>
</tr>
<tr>
<td>DocumentName</td>
<td>AWS-RefreshAssociation</td>
</tr>
<tr>
<td>ErrorCount</td>
<td>0</td>
</tr>
<tr>
<td>ExpiresAfter</td>
<td>2/24/2017 3:19:08 AM</td>
</tr>
<tr>
<td>InstanceIds</td>
<td>{i-0cb2b964d3e14fd9f}</td>
</tr>
<tr>
<td>MaxConcurrency</td>
<td>50</td>
</tr>
<tr>
<td>MaxErrors</td>
<td>0</td>
</tr>
<tr>
<td>OutputS3BucketName</td>
<td></td>
</tr>
<tr>
<td>OutputS3KeyPrefix</td>
<td></td>
</tr>
<tr>
<td>OutputS3Region</td>
<td></td>
</tr>
<tr>
<td>RequestedDateTime</td>
<td>2/24/2017 3:18:08 AM</td>
</tr>
<tr>
<td>ServiceRole</td>
<td></td>
</tr>
<tr>
<td>Status</td>
<td>Success</td>
</tr>
<tr>
<td>StatusDetails</td>
<td>Success</td>
</tr>
<tr>
<td>TargetCount</td>
<td>1</td>
</tr>
<tr>
<td>Targets</td>
<td>{}</td>
</tr>
</tbody>
</table>

Example 2: This example gets the status of a specific command.

Get-SSMCommand -CommandId "4b75a163-d39a-4d97-87c9-98ae52c6be35"

Example 3: This example retrieves all SSM commands invoked after 2019-04-01T00:00:00Z

Get-SSMCommand -Filter @{Key="InvokedAfter";Value="2019-04-01T00:00:00Z"} | Select-Object CommandId, DocumentName, Status, RequestedDateTime | Sort-Object -Property RequestedDateTime -Descending
### Output:

<table>
<thead>
<tr>
<th>CommandId</th>
<th>DocumentName</th>
<th>Status</th>
<th>RequestedDateTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>edb1b23e-456a-7adb-aef8-90e-012ac34f</td>
<td>AWS-RunPowerShellScript</td>
<td>Cancelled</td>
<td>4/16/2019 5:45:23 AM</td>
</tr>
<tr>
<td>1a2dc3fb-4567-890d-a1ad-234b5d6bc7d9</td>
<td>AWS-ConfigureAWSPackage</td>
<td>Success</td>
<td>4/6/2019 9:19:42 AM</td>
</tr>
<tr>
<td>12c3456c-7e90-4f12-1232-1234f5b67893</td>
<td>KT-Retrieve-Cloud-Type-Win</td>
<td>Failed</td>
<td>4/2/2019 4:13:07 AM</td>
</tr>
<tr>
<td>fe123b45-240c-4123-a2b3-234bdd567ecf</td>
<td>AWS-RunInspecChecks</td>
<td>Failed</td>
<td>4/1/2019 2:27:31 PM</td>
</tr>
<tr>
<td>1eb23aa4-567d-4123-12a3-4c1c2ab34561</td>
<td>AWS-RunPowerShellScript</td>
<td>Success</td>
<td>4/1/2019 1:05:55 PM</td>
</tr>
<tr>
<td>1c2f3bb4-ee12-4bc1-1a23-12345eea123e</td>
<td>AWS-RunInspecChecks</td>
<td>Failed</td>
<td>4/1/2019 11:13:09 AM</td>
</tr>
</tbody>
</table>

- For API details, see [ListCommands](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-SSMCommandInvocation

The following code example shows how to use Get-SSMCommandInvocation.

#### Tools for PowerShell

**Example 1: This example lists all the invocations of a command.**

```powershell
Get-SSMCommandInvocation -CommandId "b8eac879-0541-439d-94ec-47a80d554f44" -Detail $true
```

**Output:**

<table>
<thead>
<tr>
<th>CommandId</th>
<th>CommandPlugins</th>
<th>Comment</th>
<th>DocumentName</th>
<th>InstanceId</th>
<th>InstanceName</th>
<th>NotificationConfig</th>
</tr>
</thead>
<tbody>
<tr>
<td>b8eac879-0541-439d-94ec-47a80d554f44</td>
<td>{aws:runShellScript}</td>
<td>IP config</td>
<td>AWS-RunShellScript</td>
<td>i-0cb2b964d3e14fd9f</td>
<td></td>
<td>Amazon.SimpleSystemsManagement.Model.NotificationConfig</td>
</tr>
</tbody>
</table>
Example 2: This example lists CommandPlugins for invocation of the command id e1eb2e3c-ed4c-5123-45c1-234f5612345f

Get-SSMCommandInvocation -CommandId e1eb2e3c-ed4c-5123-45c1-234f5612345f -Detail: $true | Select-Object -ExpandProperty CommandPlugins

Output:

Name                   : aws:runPowerShellScript
Output                 : Completed 17.7 KiB/17.7 KiB (40.1 KiB/s) with 1 file(s) remainingdownload: s3://dd-aess-r-ctmer/KUMO.png to ..\..\programdata\KUMO.png
                        kumo available
OutputS3BucketName     :
OutputS3KeyPrefix      :
OutputS3Region         : eu-west-1
ResponseCode           : 0
StandardErrorUrl       :
StandardOutputUrl      :
Status                 : Success
StatusDetails          : Success

• For API details, see [ListCommandInvocations](#) in AWS Tools for PowerShell Cmdlet Reference.

Get-SSMCommandInvocationDetail

The following code example shows how to use Get-SSMCommandInvocationDetail.

Tools for PowerShell

Example 1: This example displays the details of a command executed on an instance.
Get-SSMCommandInvocationDetail -InstanceId "i-0cb2b964d3e14fd9f" -CommandId "b8eac879-0541-439d-94ec-47a80d554f44"

Output:

```
CommandId : b8eac879-0541-439d-94ec-47a80d554f44
Comment   : IP config
DocumentName : AWS-RunShellScript
ExecutionElapsedTime : PT0.004S
ExecutionEndDateTime : 2017-02-22T20:13:16.651Z
ExecutionStartDateTime : 2017-02-22T20:13:16.651Z
InstanceId : i-0cb2b964d3e14fd9f
PluginName : aws:runShellScript
ResponseCode : 0
StandardErrorContent :
StandardErrorUrl :
StandardOutputContent :
StandardOutputUrl :
Status : Success
StatusDetails : Success
```

- For API details, see [GetCommandInvocation](https://docs.aws.amazon.com/powershell/latest/ug/get-command-invocation.html) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-SSMComplianceItemList**

The following code example shows how to use Get-SSMComplianceItemList.

**Tools for PowerShell**

**Example 1: This example lists compliance items list for the given resource id and type, filtering compliance-type being 'Association'**

```
Get-SSMComplianceItemList -ResourceId i-1a2caf345f67d0dc2 -ResourceType ManagedInstance -Filter @{Key="ComplianceType";Values="Association"}
```

Output:

```
ComplianceType : Association
Details : [[DocumentName, AWS-GatherSoftwareInventory], [DocumentVersion, 1]]
```
ExecutionSummary : Amazon.SimpleSystemsManagement.Model.ComplianceExecutionSummary
Id               : 123a45a1-c234-1234-1245-67891236db4e
ResourceId       : i-1a2caf345f67d0dc2
ResourceType     : ManagedInstance
Severity         : UNSPECIFIED
Status           : COMPLIANT
Title            : 

- For API details, see ListComplianceItems in AWS Tools for PowerShell Cmdlet Reference.

Get-SSMComplianceSummaryList

The following code example shows how to use Get-SSMComplianceSummaryList.

Tools for PowerShell

Example 1: This example returns a summary count of compliant and non-compliant resources for all compliance types.

Get-SSMComplianceSummaryList

Output:

<table>
<thead>
<tr>
<th>ComplianceType</th>
<th>CompliantSummary</th>
<th>NonCompliantSummary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- For API details, see ListComplianceSummaries in AWS Tools for PowerShell Cmdlet Reference.

Get-SSMConnectionStatus

The following code example shows how to use Get-SSMConnectionStatus.
Tools for PowerShell

Example 1: This example retrieves the Session Manager connection status for an instance to determine whether it is connected and ready to receive Session Manager connections.

```powershell
Get-SSMConnectionStatus -Target i-0a1caf234f12d3dc4
```

Output:

<table>
<thead>
<tr>
<th>Status</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connected</td>
<td>i-0a1caf234f12d3dc4</td>
</tr>
</tbody>
</table>

- For API details, see [GetConnectionStatus](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-SSMDefaultPatchBaseline**

The following code example shows how to use Get-SSMDefaultPatchBaseline.

Tools for PowerShell

Example 1: This example displays the default patch baseline.

```powershell
Get-SSMDefaultPatchBaseline
```

Output:

```
```

- For API details, see [GetDefaultPatchBaseline](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-SSMDeployablePatchSnapshotForInstance**

The following code example shows how to use Get-SSMDeployablePatchSnapshotForInstance.

Tools for PowerShell

Example 1: This example displays the current snapshot for the patch baseline used by an Instance. This command must be run from the instance using the instance Systems Manager.
credentials. To ensure it uses the instance credentials, the example passes an
Amazon.Runtime.InstanceProfileAWSCredentials object to the Credentials parameter.

```powershell
$credentials = [Amazon.Runtime.InstanceProfileAWSCredentials]::new()
Get-SSMDeployablePatchSnapshotForInstance -SnapshotId "4681775b-098f-4435-a956-0ef33373ac11" -InstanceId "i-0cb2b964d3e14fd9f" -Credentials $credentials
```

**Output:**

<table>
<thead>
<tr>
<th>InstanceId</th>
<th>SnapshotDownloadUrl</th>
</tr>
</thead>
<tbody>
<tr>
<td>i-0cb2b964d3e14fd9f</td>
<td><a href="https://patch-baseline-snapshot-us-west-2.s3-us-west-2.amazonaws.com/853d0d3db0f0cafe...1692/4681775b-098f-4435">https://patch-baseline-snapshot-us-west-2.s3-us-west-2.amazonaws.com/853d0d3db0f0cafe...1692/4681775b-098f-4435</a>...</td>
</tr>
</tbody>
</table>

**Example 2:** This example shows how to get the full SnapshotDownloadUrl. This command must be run from the instance using the instance credentials. To ensure it uses the instance credentials, the example configures the PowerShell session to use an Amazon.Runtime.InstanceProfileAWSCredentials object.

```powershell
Set-AWSCredential -Credential ([Amazon.Runtime.InstanceProfileAWSCredentials]::new())
(Get-SSMDeployablePatchSnapshotForInstance -SnapshotId "4681775b-098f-4435-a956-0ef33373ac11" -InstanceId "i-0cb2b964d3e14fd9f").SnapshotDownloadUrl
```

**Output:**

```powershell
https://patch-baseline-snapshot-us-west-2.s3-us-west-2.amazonaws.com/853d0d3db0f0cafe...
```

- For API details, see [GetDeployablePatchSnapshotForInstance](https://aws.amazon.com/manual) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-SSMDocument**

The following code example shows how to use Get-SSMDocument.

**Tools for PowerShell**

**Example 1:** This example returns the content of a document.
Get-SSMDocument -Name "RunShellScript"

**Output:**

```
Content
-------
...
```

**Example 2: This example displays the complete contents of a document.**

```powershell
(Get-SSMDocument -Name "RunShellScript").Content
{
    "schemaVersion":"2.0",
    "description":"Run an updated script",
    "parameters":{
        "commands":{
            "type":"StringList",
            "description": "(Required) Specify a shell script or a command to run.",
            "minItems":1,
            "displayType":"textarea"
        }
    },
    "mainSteps":[
        {
            "action":"aws:runShellScript",
            "name":"runShellScript",
            "inputs":{
                "commands":"{[ commands ]}"            }
        },
        {
            "action":"aws:runPowerShellScript",
            "name":"runPowerShellScript",
            "inputs":{
                "commands":"{[ commands ]}"            }
        }
    ]
}
```

- For API details, see [GetDocument](#) in *AWS Tools for PowerShell Cmdlet Reference*.
Get-SSMDocumentDescription

The following code example shows how to use Get-SSMDocumentDescription.

Tools for PowerShell

Example 1: This example returns information about a document.

```
Get-SSMDocumentDescription -Name "RunShellScript"
```

Output:

```
CreatedDate     : 2/24/2017 5:25:13 AM
DefaultVersion  : 1
Description     : Run an updated script
DocumentType    : Command
DocumentVersion : 1
Hash            : f775e5df4904c6fa46686c4722f9e9de1950dace25cd9608ff8d622046b68d9b
HashType        : Sha256
LatestVersion   : 1
Name            : RunShellScript
Owner           : 123456789012
Parameters      : {commands}
PlatformTypes   : {Linux}
SchemaVersion   : 2.0
Sha1            :
Status          : Active
```

- For API details, see [DescribeDocument](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-SSMDocumentList

The following code example shows how to use Get-SSMDocumentList.

Tools for PowerShell

Example 1: Lists all the configuration documents in your account.

```
Get-SSMDocumentList
```

Output:
Example 2: This example retrieves all automation documents with name matching 'Platform'

```powershell
Get-SSMDocumentList -DocumentFilterList @{Key="DocumentType";Value="Automation"} | Where-Object Name -Match "Platform"
```

**Output:**

```
DocumentFormat : JSON
DocumentType    : Automation
DocumentVersion : 7
Name            : KT-Get-Platform
Owner           : 987654123456
PlatformTypes   : {Windows, Linux}
SchemaVersion   : 0.3
Tags            : {}
TargetType      : 
VersionName     :
```

• For API details, see [ListDocuments](#) in *AWS Tools for PowerShell Cmdlet Reference.*
Get-SSMDocumentPermission

The following code example shows how to use Get-SSMDocumentPermission.

**Tools for PowerShell**

**Example 1: This example lists all the versions for a document.**

```powershell
Get-SSMDocumentVersionList -Name "RunShellScript"
```

**Output:**

<table>
<thead>
<tr>
<th>CreatedDate</th>
<th>DocumentVersion</th>
<th>IsDefaultVersion</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>2/24/2017 5:25:13 AM</td>
<td>1</td>
<td>True</td>
<td>RunShellScript</td>
</tr>
</tbody>
</table>

- For API details, see [DescribeDocumentPermission](https://aws.amazon.com/tools/powershell/cmdlet-reference/) in *AWS Tools for PowerShell Cmdlet Reference*.

Get-SSMDocumentVersionList

The following code example shows how to use Get-SSMDocumentVersionList.

**Tools for PowerShell**

**Example 1: This example returns the permission list for a document.**

```powershell
Get-SSMDocumentPermission -Name "RunShellScript" -PermissionType "Share"
```

**Output:**

```
all
```


Get-SSMEffectiveInstanceAssociationList

The following code example shows how to use Get-SSMEffectiveInstanceAssociationList.
Tools for PowerShell

Example 1: This example describes the effective associations for an instance.

Get-SSMEffectiveInstanceAssociationList -InstanceId "i-0000293fffd8c57862" -MaxResult 5

Output:

<table>
<thead>
<tr>
<th>AssociationId</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>d8617c07-2079-4c18-9847-1655fc2698b0</td>
<td>{...</td>
</tr>
</tbody>
</table>

Example 2: This example displays the contents of the effective associations for an instance.

(Get-SSMEffectiveInstanceAssociationList -InstanceId "i-0000293fffd8c57862" -MaxResult 5).Content

Output:

```json
{
  "schemaVersion": "1.2",
  "description": "Update the Amazon SSM Agent to the latest version or specified version.",
  "parameters": {
    "version": {
      "default": "",
      "description": "(Optional) A specific version of the Amazon SSM Agent to install. If not specified, the agent will be updated to the latest version.",
      "type": "String"
    },
    "allowDowngrade": {
      "default": "false",
      "description": "(Optional) Allow the Amazon SSM Agent service to be downgraded to an earlier version. If set to false, the service can be upgraded to newer versions only (default). If set to true, specify the earlier version.",
      "type": "String",
      "allowedValues": [
        "true",
        "false"
      ]
    }
  }
}
```
• For API details, see [DescribeEffectiveInstanceAssociations](#) in AWS Tools for PowerShell Cmdlet Reference.

**Get-SSMEffectivePatchesForPatchBaseline**

The following code example shows how to use Get-SSMEffectivePatchesForPatchBaseline.

**Tools for PowerShell**

**Example 1:** This example lists all patch baselines, with a maximum result list of 1.

```powershell
Get-SSMEffectivePatchesForPatchBaseline -BaselineId "pb-0a2f1059b670ebd31" -MaxResult 1
```

**Output:**

<table>
<thead>
<tr>
<th>Patch</th>
<th>PatchStatus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon.SimpleSystemsManagement.Model.Patch</td>
<td></td>
</tr>
</tbody>
</table>

**Example 2:** This example displays the patch status for all patch baselines, with a maximum result list of 1.
(Get-SSMEffectivePatchesForPatchBaseline -BaselineId "pb-0a2f1059b670ebd31" -MaxResult 1).PatchStatus

Output:

<table>
<thead>
<tr>
<th>ApprovalDate</th>
<th>DeploymentStatus</th>
</tr>
</thead>
<tbody>
<tr>
<td>12/21/2010 6:00:00 PM</td>
<td>APPROVED</td>
</tr>
</tbody>
</table>

- For API details, see DescribeEffectivePatchesForPatchBaseline in AWS Tools for PowerShell Cmdlet Reference.

Get-SSMInstanceAssociationsStatus

The following code example shows how to use Get-SSMInstanceAssociationsStatus.

Tools for PowerShell

**Example 1:** This example shows details of the associations for an instance.

```
Get-SSMInstanceAssociationsStatus -InstanceId "i-0000293ffd8c57862"
```

Output:

```
AssociationId    : d8617c07-2079-4c18-9847-1655fc2698b0
DetailedStatus   : Pending
DocumentVersion  : 1
ErrorCode        : 
ExecutionDate    : 2/20/2015 8:31:11 AM
ExecutionSummary : temp_status_change
InstanceId       : i-0000293ffd8c57862
Name             : AWS-UpdateSSMAgent
OutputUrl        : 
Status           : Pending
```

**Example 2:** This example checks the instance association status for the given instance id and further, displays the execution status of those associations

```
Get-SSMInstanceAssociationsStatus -InstanceId i-012e3cb4df567e8aa | ForEach-Object {Get-SSMAssociationExecution -AssociationId .AssociationId}
```
Output:

<table>
<thead>
<tr>
<th>AssociationId</th>
<th>512a34a5-c678-1234-1234-12345678db9e</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssociationVersion</td>
<td>2</td>
</tr>
<tr>
<td>CreatedTime</td>
<td>3/2/2019 8:53:29 AM</td>
</tr>
<tr>
<td>DetailedStatus</td>
<td></td>
</tr>
<tr>
<td>ExecutionId</td>
<td>512a34a5-c678-1234-1234-12345678db9e</td>
</tr>
<tr>
<td>LastExecutionDate</td>
<td>1/1/0001 12:00:00 AM</td>
</tr>
<tr>
<td>ResourceCountByStatus</td>
<td>{Success=9}</td>
</tr>
<tr>
<td>Status</td>
<td>Success</td>
</tr>
</tbody>
</table>

For API details, see [DescribeInstanceAssociationsStatus](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

**Get-SSMInstanceInformation**

The following code example shows how to use Get-SSMInstanceInformation.

**Tools for PowerShell**

**Example 1: This example shows details of each of your instances.**

Get-SSMInstanceInformation

**Output:**

<table>
<thead>
<tr>
<th>ActivationId</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentVersion</td>
<td>2.0.672.0</td>
</tr>
<tr>
<td>AssociationOverview</td>
<td></td>
</tr>
<tr>
<td>Amazon.SimpleSystemsManagement.Model.InstanceAggregatedAssociationOverview</td>
<td></td>
</tr>
<tr>
<td>AssociationStatus</td>
<td>Success</td>
</tr>
<tr>
<td>ComputerName</td>
<td>ip-172-31-44-222.us-west-2.compute.internal</td>
</tr>
<tr>
<td>IamRole</td>
<td></td>
</tr>
<tr>
<td>InstanceId</td>
<td>i-0cb2b964d3e14fd9f</td>
</tr>
<tr>
<td>IPAddress</td>
<td>172.31.44.222</td>
</tr>
<tr>
<td>IsLatestVersion</td>
<td>True</td>
</tr>
<tr>
<td>LastAssociationExecutionDate</td>
<td>2/24/2017 3:18:09 AM</td>
</tr>
<tr>
<td>LastPingDateTime</td>
<td>2/24/2017 3:35:03 AM</td>
</tr>
<tr>
<td>LastSuccessfulAssociationExecutionDate</td>
<td>2/24/2017 3:18:09 AM</td>
</tr>
<tr>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>PingStatus</td>
<td>ConnectionLost</td>
</tr>
</tbody>
</table>
Example 2: This example shows how to use the -Filter parameter to filter results to only those AWS Systems Manager instances in region us-east-1 with an AgentVersion of 2.2.800.0. You can find a list of valid -Filter key values in the InstanceInformation API reference topic (https://docs.aws.amazon.com/systems-manager/latest/APIReference/API_InstanceInformation.html#systemsmanager-Type-InstanceInformation-ActivationId).

```powershell
Example 2: This example shows how to use the -Filter parameter to filter results to only those AWS Systems Manager instances in region us-east-1 with an AgentVersion of 2.2.800.0. You can find a list of valid -Filter key values in the InstanceInformation API reference topic (https://docs.aws.amazon.com/systems-manager/latest/APIReference/API_InstanceInformation.html#systemsmanager-Type-InstanceInformation-ActivationId).

$Filters = @{
    Key="AgentVersion"
    Values="2.2.800.0"
}
Get-SSMInstanceInformation -Region us-east-1 -Filter $Filters
```

Output:

<table>
<thead>
<tr>
<th>ActivationId</th>
<th>:</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentVersion</td>
<td>2.2.800.0</td>
</tr>
<tr>
<td>AssociationOverview</td>
<td>Amazon.SimpleSystemsManagement.Model.InstanceAggregatedAssociationOverview</td>
</tr>
<tr>
<td>AssociationStatus</td>
<td>Success</td>
</tr>
<tr>
<td>ComputerName</td>
<td>EXAMPLE-EXAMPLE.WORKGROUP</td>
</tr>
<tr>
<td>IamRole</td>
<td>:</td>
</tr>
<tr>
<td>InstanceId</td>
<td>i-EXAMPLEb0792d98ce</td>
</tr>
<tr>
<td>IPAddress</td>
<td>10.0.0.01</td>
</tr>
<tr>
<td>IsLatestVersion</td>
<td>False</td>
</tr>
<tr>
<td>LastAssociationExecutionDate</td>
<td>8/16/2018 12:02:50 AM</td>
</tr>
<tr>
<td>LastPingDateTime</td>
<td>8/16/2018 7:40:27 PM</td>
</tr>
<tr>
<td>LastSuccessfulAssociationExecutionDate</td>
<td>8/16/2018 12:02:50 AM</td>
</tr>
<tr>
<td>Name</td>
<td>:</td>
</tr>
<tr>
<td>PingStatus</td>
<td>Online</td>
</tr>
<tr>
<td>PlatformName</td>
<td>Microsoft Windows Server 2016 Datacenter</td>
</tr>
<tr>
<td>PlatformType</td>
<td>Windows</td>
</tr>
<tr>
<td>PlatformVersion</td>
<td>10.0.14393</td>
</tr>
<tr>
<td>RegistrationDate</td>
<td>1/1/0001 12:00:00 AM</td>
</tr>
<tr>
<td>ResourceType</td>
<td>EC2Instance</td>
</tr>
<tr>
<td>ActivationId</td>
<td>:</td>
</tr>
</tbody>
</table>
Example 3: This example shows how to use the `-InstanceInformationFilterList` parameter to filter results to only those AWS Systems Manager instances in region `us-east-1` with `PlatformTypes` of `Windows` or `Linux`. You can find a list of valid `-InstanceInformationFilterList` key values in the InstanceInformationFilter API reference topic (https://docs.aws.amazon.com/systems-manager/latest/APIReference/API_InstanceInformationFilter.html).

```powershell
$Filters = @{
    Key="PlatformTypes"
    ValueSet=("Windows","Linux")
}
Get-SSMInstanceInformation -Region us-east-1 -InstanceInformationFilterList $Filters
```

Output:

```
ActivationId                           :
AgentVersion                           : 2.2.800.0
AssociationOverview                    :
    Amazon.SimpleSystemsManagement.Model.InstanceAggregatedAssociationOverview
AssociationStatus                      : Success
ComputerName                           : EXAMPLE-EXAMPLE.WORKGROUP
IamRole                                :
InstanceId                             : i-EXAMPLEac7501d023
IPAddress                              : 10.0.0.02
IsLatestVersion                        : False
LastAssociationExecutionDate           : 8/16/2018 12:00:20 AM
LastPingDateTime                       : 8/16/2018 7:40:35 PM
LastSuccessfulAssociationExecutionDate : 8/16/2018 12:00:20 AM
Name                                   :
PingStatus                             : Online
PlatformName                           : Microsoft Windows Server 2016 Datacenter
PlatformType                           : Windows
PlatformVersion                        : 10.0.14393
RegistrationDate                       : 1/1/0001 12:00:00 AM
ResourceType                           : EC2Instance
```
**Example 4:** This example lists ssm managed instances and exports Instanceld, PingStatus, LastPingDateTime and PlatformName to a csv file.

```powershell
Get-SSMInstanceInformation | Select-Object InstanceId, PingStatus, LastPingDateTime, PlatformName | Export-Csv Instance-details.csv -NoTypeInformation
```

- For API details, see [DescribeInstanceInformation](#) in AWS Tools for PowerShell Cmdlet Reference.
Get-SSMInstancePatch

The following code example shows how to use Get-SSMInstancePatch.

Tools for PowerShell

Example 1: This example gets the patch compliance details for an instance.

```bash
Get-SSMInstancePatch -InstanceId "i-08ee91c0b17045407"
```

- For API details, see [DescribeInstancePatches](#) in AWS Tools for PowerShell Cmdlet Reference.

Get-SSMInstancePatchState

The following code example shows how to use Get-SSMInstancePatchState.

Tools for PowerShell

Example 1: This example gets the patch summary states for an instance.

```bash
Get-SSMInstancePatchState -InstanceId "i-08ee91c0b17045407"
```

Example 2: This example gets the patch summary states for two instances.

```bash
Get-SSMInstancePatchState -InstanceId "i-08ee91c0b17045407","i-09a618aec652973a9"
```

- For API details, see [DescribeInstancePatchStates](#) in AWS Tools for PowerShell Cmdlet Reference.

Get-SSMInstancePatchStatesForPatchGroup

The following code example shows how to use Get-SSMInstancePatchStatesForPatchGroup.

Tools for PowerShell

Example 1: This example gets the patch summary states per-instance for a patch group.

```bash
Get-SSMInstancePatchStatesForPatchGroup -PatchGroup "Production"
```

- For API details, see [DescribeInstancePatchStatesForPatchGroup](#) in AWS Tools for PowerShell Cmdlet Reference.
Get-SSMInventory

The following code example shows how to use Get-SSMInventory.

Tools for PowerShell

Example 1: This example gets the custom metadata for your inventory.

Get-SSMInventory

Output:

<table>
<thead>
<tr>
<th>Data</th>
<th>Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>----</td>
<td>-----------</td>
</tr>
<tr>
<td>----</td>
<td>--</td>
</tr>
</tbody>
</table>

{[[AWS:InstanceInformation, Amazon.SimpleSystemsManagement.Model.InventoryResultItem]} i-0cb2b964d3e14fd9f

- For API details, see [GetInventory](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

Get-SSMInventoryEntriesList

The following code example shows how to use Get-SSMInventoryEntriesList.

Tools for PowerShell

Example 1: This example lists all the custom inventory entries for an instance.

Get-SSMInventoryEntriesList -InstanceId "i-0cb2b964d3e14fd9f" -TypeName "Custom:RackInfo"

Output:

```
CaptureTime  : 2016-08-22T10:01:01Z
Entries      :
InstanceId   : i-0cb2b964d3e14fd9f
NextToken    :
SchemaVersion : 1.0
```
Example 2: This example lists the details.

```
(Get-SSMInventoryEntriesList -InstanceId "i-0cb2b964d3e14fd9f" -TypeName "Custom:RackInfo").Entries
```

Output:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RackLocation</td>
<td>Bay B/Row C/Rack D/Shelf E</td>
</tr>
</tbody>
</table>

- For API details, see [ListInventoryEntries](https://docs.aws.amazon.com/powershell/latest/ug/list-inventory-entries.html) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-SSMInventoryEntryList**

The following code example shows how to use `Get-SSMInventoryEntryList`.

**Tools for PowerShell**

Example 1: This example retrieves AWS:Network type inventory entries for the instance.

```
Get-SSMInventoryEntryList -InstanceId mi-088dcb0ecea37b076 -TypeName AWS:Network | Select-Object -ExpandProperty Entries
```

Output:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DHCPServer</td>
<td>172.31.11.2</td>
</tr>
<tr>
<td>DNSServer</td>
<td>172.31.0.1</td>
</tr>
<tr>
<td>Gateway</td>
<td>172.31.11.2</td>
</tr>
<tr>
<td>IPV4</td>
<td>172.31.11.222</td>
</tr>
<tr>
<td>IPV6</td>
<td>fe12::3456:7da8:901a:12a3</td>
</tr>
<tr>
<td>MacAddress</td>
<td>1A:23:4E:5B:FB:67</td>
</tr>
<tr>
<td>Name</td>
<td>Amazon Elastic Network Adapter</td>
</tr>
<tr>
<td>SubnetMask</td>
<td>255.255.240.0</td>
</tr>
</tbody>
</table>

- For API details, see [Get-SSMInventoryEntryList](https://docs.aws.amazon.com/powershell/latest/ug/get-inventory-entries-list.html) in *AWS Tools for PowerShell Cmdlet Reference*. 
Get-SSMInventorySchema

The following code example shows how to use Get-SSMInventorySchema.

Tools for PowerShell

Example 1: This example returns a list of inventory type names for the account.

```
Get-SSMInventorySchema
```

- For API details, see GetInventorySchema in AWS Tools for PowerShell Cmdlet Reference.

Get-SSMLatestEC2Image

The following code example shows how to use Get-SSMLatestEC2Image.

Tools for PowerShell

Example 1: This example lists all the latest Windows AMIs.

```
PS Get-SSMLatestEC2Image -Path ami-windows-latest
```

Output:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows_Server-2008-R2_SP1-English-64Bit-SQL_2012_SP4_Express</td>
<td>ami-0e5ddd288daf4fab</td>
</tr>
<tr>
<td>Windows_Server-2012-R2_RTM-Chinese_Simplified-64Bit-Base</td>
<td>ami-0c5ea6466dc1cb50</td>
</tr>
<tr>
<td>Windows_Server-2012-R2_RTM-Chinese_Traditional-64Bit-Base</td>
<td>ami-09775eff0bf8c113d</td>
</tr>
<tr>
<td>Windows_Server-2012-R2_RTM-Dutch-64Bit-Base</td>
<td>ami-025064b67e28cf5df</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Example 2: This example retrieves the AMI id of a specific Amazon Linux image for the us-west-2 region.

```
PS Get-SSMLatestEC2Image -Path ami-amazon-linux-latest -ImageName amzn-ami-hvm-x86_64-ebs -Region us-west-2
```
Output:

ami-09b92cd132204c704

Example 3: This example lists all the latest Windows AMIs matching the specified wildcard expression.

Get-SSMLatestEC2Image -Path ami-windows-latest -ImageName *Windows*2019*English*

Output:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows_Server-2019-English-Full-SQL_2017/Web</td>
<td>ami-085e9d27da5b73a42</td>
</tr>
<tr>
<td>Windows_Server-2019-English-STIG-Core</td>
<td>ami-0bfd85c29148c7f80</td>
</tr>
<tr>
<td>Windows_Server-2019-English-Full-SQL_2016_SP2_Standard</td>
<td>ami-0d7ae2d81c07bd598</td>
</tr>
</tbody>
</table>

- For API details, see [Get-SSMLatestEC2Image](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-SSMMaintenanceWindow**

The following code example shows how to use Get-SSMMaintenanceWindow.

**Tools for PowerShell**

**Example 1: This example gets details about a maintenance window.**

Get-SSMMaintenanceWindow -WindowId "mw-03eb9db42890fb82d"

**Output:**

<table>
<thead>
<tr>
<th>AllowUnassociatedTargets</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreatedDate</td>
<td>2/20/2017 6:14:05 PM</td>
</tr>
<tr>
<td>Cutoff</td>
<td>1</td>
</tr>
<tr>
<td>Duration</td>
<td>2</td>
</tr>
<tr>
<td>Enabled</td>
<td>True</td>
</tr>
<tr>
<td>ModifiedDate</td>
<td>2/20/2017 6:14:05 PM</td>
</tr>
<tr>
<td>Name</td>
<td>TestMaintWin</td>
</tr>
</tbody>
</table>
Schedule : cron(0 */30 * * * ? *)
WindowId : mw-03eb9db42890fb82d

- For API details, see [GetMaintenanceWindow](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-SSMMaintenanceWindowExecution**

The following code example shows how to use `Get-SSMMaintenanceWindowExecution`.

**Tools for PowerShell**

**Example 1:** This example lists information about a task executed as part of a maintenance window execution.

```powershell
Get-SSMMaintenanceWindowExecution -WindowExecutionId "518d5565-5969-4cca-8f0e-da3b2a638355"
```

**Output:**

- EndTime : 2/21/2017 4:00:35 PM
- StartTime : 2/21/2017 4:00:34 PM
- Status : FAILED
- StatusDetails : One or more tasks in the orchestration failed.
- TaskIds : {ac0c6ae1-daa3-4a89-832e-d384503b6586}
- WindowExecutionId : 518d5565-5969-4cca-8f0e-da3b2a638355

- For API details, see [GetMaintenanceWindowExecution](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-SSMMaintenanceWindowExecutionList**

The following code example shows how to use `Get-SSMMaintenanceWindowExecutionList`.

**Tools for PowerShell**

**Example 1:** This example lists all of the executions for a maintenance window.

```powershell
Get-SSMMaintenanceWindowExecutionList -WindowId "mw-03eb9db42890fb82d"
```

**Output:**
Example 2: This example lists all of the executions for a maintenance window before a specified date.

```powershell
$options1 = @{'Key'='ExecutedBefore'; 'Values'= @('2016-11-04T05:00:00Z')}
Get-SSMMaintenanceWindowExecutionList -WindowId "mw-03eb9db42890fb82d" -Filter $options1
```

Example 3: This example lists all of the executions for a maintenance window after a specified date.

```powershell
$options1 = @{'Key'='ExecutedAfter'; 'Values'= @('2016-11-04T05:00:00Z')}
Get-SSMMaintenanceWindowExecutionList -WindowId "mw-03eb9db42890fb82d" -Filter $options1
```

- For API details, see DescribeMaintenanceWindowExecutions in AWS Tools for PowerShell Cmdlet Reference.

Get-SSMMaintenanceWindowExecutionTask

The following code example shows how to use Get-SSMMaintenanceWindowExecutionTask.

Tools for PowerShell

Example 1: This example lists information about a task that was part of a maintenance window execution.

```powershell
Get-SSMMaintenanceWindowExecutionTask -TaskId "ac0c6ae1-daa3-4a89-832e-d384503b6586" -WindowExecutionId "518d5565-5969-4cca-8f0e-da3b2a638355"
```

Output:

<table>
<thead>
<tr>
<th>EndTime</th>
<th>2/21/2017 4:00:35 PM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxConcurrency</td>
<td>1</td>
</tr>
</tbody>
</table>
For API details, see `GetMaintenanceWindowExecutionTask` in *AWS Tools for PowerShell Cmdlet Reference.*

**Get-SSMMaintenanceWindowExecutionTaskInvocationList**

The following code example shows how to use `Get-SSMMaintenanceWindowExecutionTaskInvocationList`.

**Tools for PowerShell**

**Example 1:** This example lists the invocations for a task executed as part of a maintenance window execution.

```powershell
Get-SSMMaintenanceWindowExecutionTaskInvocationList -TaskId "ac0c6ae1-daa3-4a89-832e-d384503b6586" -WindowExecutionId "518d5565-5969-4cca-8f0e-da3b2a638355"
```

**Output:**

```powershell
EndTime           : 2/21/2017 4:00:34 PM
ExecutionId       :
InvocationId      : e274b6e1-fe56-4e32-bd2a-8073c6381d8b
OwnerInformation  :
Parameters        : "documentName":"AWS-RunShellScript","instanceIds":
                   ["i-0000293fffd8c57862"],"parameters":{"commands":"df"},"maxConcurrency":"1",
                   "maxErrors":"1"
StartTime         : 2/21/2017 4:00:34 PM
Status            : FAILED
```
StatusDetails : The instance IDs list contains an invalid entry.
TaskExecutionId : ac0c6ae1-daa3-4a89-832e-d384503b6586
WindowExecutionId : 518d5565-5969-4cca-8f0e-da3b2a638355
WindowTargetId :

- For API details, see `DescribeMaintenanceWindowExecutionTaskInvocations` in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-SSMMaintenanceWindowExecutionTaskList**

The following code example shows how to use `Get-SSMMaintenanceWindowExecutionTaskList`.

**Tools for PowerShell**

**Example 1:** This example lists the tasks associated with a maintenance window execution.

```
Get-SSMMaintenanceWindowExecutionTaskList -WindowExecutionId
"518d5565-5969-4cca-8f0e-da3b2a638355"
```

**Output:**

```
EndTime           : 2/21/2017 4:00:35 PM
StartTime         : 2/21/2017 4:00:34 PM
Status            : SUCCESS
TaskArn           : AWS-RunShellScript
TaskExecutionId   : ac0c6ae1-daa3-4a89-832e-d384503b6586
TaskType          : RUN_COMMAND
WindowExecutionId : 518d5565-5969-4cca-8f0e-da3b2a638355
```

- For API details, see `DescribeMaintenanceWindowExecutionTasks` in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-SSMMaintenanceWindowList**

The following code example shows how to use `Get-SSMMaintenanceWindowList`.

**Tools for PowerShell**

**Example 1:** This example lists all maintenance windows on your account.
Get-SSMMaintenanceWindowList

Output:

Cutoff : 1
Duration : 4
Enabled : True
Name : My-First-Maintenance-Window
WindowId : mw-06d59c1a07c022145

- For API details, see [DescribeMaintenanceWindows](#) in [AWS Tools for PowerShell Cmdlet Reference](#).

Get-SSMMaintenanceWindowTarget

The following code example shows how to use Get-SSMMaintenanceWindowTarget.

Tools for PowerShell

**Example 1:** This example lists all of the targets for a maintenance window.

Get-SSMMaintenanceWindowTarget -WindowId "mw-06cf17cbebc4b4f4f"

Output:

OwnerInformation : Single instance
ResourceType : INSTANCE
Targets : {InstanceIds}
WindowId : mw-06cf17cbebc4b4f4f
WindowTargetId : 350d44e6-28cc-44e2-951f-4b2c985838f6

OwnerInformation : Two instances in a list
ResourceType : INSTANCE
Targets : {InstanceIds}
WindowId : mw-06cf17cbebc4b4f4f
WindowTargetId : e078a987-2866-47be-bedd-d9cf49177d3a

- For API details, see [DescribeMaintenanceWindowTargets](#) in [AWS Tools for PowerShell Cmdlet Reference](#).
Get-SSMMaintenanceWindowTaskList

The following code example shows how to use Get-SSMMaintenanceWindowTaskList.

Tools for PowerShell

Example 1: This example lists all of the tasks for a maintenance window.

Get-SSMMaintenanceWindowTaskList -WindowId "mw-06cf17cbefcb4bf4f"

Output:

<table>
<thead>
<tr>
<th>LoggingInfo</th>
<th>:</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxConcurrency</td>
<td>: 1</td>
</tr>
<tr>
<td>MaxErrors</td>
<td>: 1</td>
</tr>
<tr>
<td>Priority</td>
<td>: 10</td>
</tr>
<tr>
<td>ServiceRoleArn</td>
<td>: arn:aws:iam::123456789012:role/MaintenanceWindowsRole</td>
</tr>
<tr>
<td>Targets</td>
<td>: {InstanceIds}</td>
</tr>
<tr>
<td>TaskArn</td>
<td>: AWS-RunShellScript</td>
</tr>
<tr>
<td>Type</td>
<td>: RUN_COMMAND</td>
</tr>
<tr>
<td>WindowId</td>
<td>: mw-06cf17cbefcb4bf4f</td>
</tr>
<tr>
<td>WindowTaskId</td>
<td>: a23e338d-ff30-4398-8aa3-09cd052ebf17</td>
</tr>
</tbody>
</table>

- For API details, see [DescribeMaintenanceWindowTasks](#) in AWS Tools for PowerShell Cmdlet Reference.

Get-SSMParameterHistory

The following code example shows how to use Get-SSMParameterHistory.

Tools for PowerShell

Example 1: This example lists the value history for a parameter.

Get-SSMParameterHistory -Name "Welcome"

Output:

<table>
<thead>
<tr>
<th>Description</th>
<th>:</th>
</tr>
</thead>
</table>
KeyId            :  
LastModifiedDate : 3/3/2017 6:55:25 PM  
LastModifiedUser : arn:aws:iam::123456789012:user/admin  
Name             : Welcome  
Type             : String  
Value            : helloWorld

• For API details, see GetParameterHistory in AWS Tools for PowerShell Cmdlet Reference.

Get-SSMParameterList

The following code example shows how to use Get-SSMParameterList.

Tools for PowerShell

Example 1: This example lists all parameters.

Get-SSMParameterList

Output:

Description      :  
KeyId            :  
LastModifiedDate : 3/3/2017 6:58:23 PM  
LastModifiedUser : arn:aws:iam::123456789012:user/admin  
Name             : Welcome  
Type             : String

• For API details, see DescribeParameters in AWS Tools for PowerShell Cmdlet Reference.

Get-SSMParameterValue

The following code example shows how to use Get-SSMParameterValue.

Tools for PowerShell

Example 1: This example lists the values for a parameter.

Get-SSMParameterValue -Name "Welcome"
Example 2: This example lists the details of the value.

```
(Get-SSMParameterValue -Name "Welcome").Parameters
```

Output:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Welcome</td>
<td>String</td>
<td>Good day, Sunshine!</td>
</tr>
</tbody>
</table>

- For API details, see [GetParameters](#) in AWS Tools for PowerShell Cmdlet Reference.

Get-SSMPatchBaseline

The following code example shows how to use Get-SSMPatchBaseline.

Tools for PowerShell

Example 1: This example lists all patch baselines.

```
Get-SSMPatchBaseline
```

Output:

<table>
<thead>
<tr>
<th>BaselineDescription</th>
<th>BaselineId</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline containing all updates approved for production systems</td>
<td>pb-045f10b4f382baeda Production-B...</td>
</tr>
<tr>
<td>Baseline containing all updates approved for production systems</td>
<td>pb-0a2f1059b670ebd31 Production-B...</td>
</tr>
</tbody>
</table>
Example 2: This example lists all patch baselines provided by AWS. The syntax used by this example requires PowerShell version 3 or later.

```
$filter1 = @{Key="OWNER";Values=@("AWS")}
```

Output:

```
Get-SSMPatchBaseline -Filter $filter1
```

Example 3: This example lists all patch baselines with you as the owner. The syntax used by this example requires PowerShell version 3 or later.

```
$filter1 = @{Key="OWNER";Values=@("Self")}
```

Output:

```
Get-SSMPatchBaseline -Filter $filter1
```

Example 4: With PowerShell version 2, you must use New-Object to create each tag.

```
$filter1 = New-Object Amazon.SimpleSystemsManagement.Model.PatchOrchestratorFilter
$filter1.Key = "OWNER"
$filter1.Values = "AWS"
```

Output:

```
Get-SSMPatchBaseline -Filter $filter1
```

<table>
<thead>
<tr>
<th>BaselineDescription</th>
<th>BaselineId</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Patch Baseline Provided by AWS.</td>
<td>arn:aws:ssm:us-west-2:123456789012:patchbaseline/pb-04fb4ae6142167966 AWS-DefaultPatchBaseline True</td>
</tr>
</tbody>
</table>

- For API details, see [DescribePatchBaselines](#) in [AWS Tools for PowerShell Cmdlet Reference](#).
Get-SSMPatchBaselineDetail

The following code example shows how to use Get-SSMPatchBaselineDetail.

Tools for PowerShell

Example 1: This example displays the details for a patch baseline.

Example code:

```
Get-SSMPatchBaselineDetail -BaselineId "pb-03da896ca3b68b639"
```

Output:

```
ApprovedPatches : {}
BaselineId      : pb-03da896ca3b68b639
CreatedDate     : 3/3/2017 5:02:19 PM
Description     : Baseline containing all updates approved for production systems
ModifiedDate    : 3/3/2017 5:02:19 PM
Name            : Production-Baseline
PatchGroups     : {}
RejectedPatches : {}
```

- For API details, see GetPatchBaseline in AWS Tools for PowerShell Cmdlet Reference.

Get-SSMPatchBaselineForPatchGroup

The following code example shows how to use Get-SSMPatchBaselineForPatchGroup.

Tools for PowerShell

Example 1: This example displays the patch baseline for a patch group.

Example code:

```
Get-SSMPatchBaselineForPatchGroup -PatchGroup "Production"
```

Output:

```
BaselineId                  PatchGroup
----------                  ----------
pb-045f10b4f382baeda        Production
```
• For API details, see [GetPatchBaselineForPatchGroup](#) in AWS Tools for PowerShell Cmdlet Reference.

### Get-SSMPatchGroup

The following code example shows how to use Get-SSMPatchGroup.

**Tools for PowerShell**

#### Example 1: This example lists the patch group registrations.

```powershell
Get-SSMPatchGroup
```

**Output:**

<table>
<thead>
<tr>
<th>BaselineIdentity</th>
<th>PatchGroup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon.SimpleSystemsManagement.Model.PatchBaselineIdentity</td>
<td>Production</td>
</tr>
</tbody>
</table>

• For API details, see [DescribePatchGroups](#) in AWS Tools for PowerShell Cmdlet Reference.

### Get-SSMPatchGroupState

The following code example shows how to use Get-SSMPatchGroupState.

**Tools for PowerShell**

#### Example 1: This example gets the high-level patch compliance summary for a patch group.

```powershell
Get-SSMPatchGroupState -PatchGroup "Production"
```

**Output:**

- Instances                          : 4
- InstancesWithFailedPatches         : 1
- InstancesWithInstalledOtherPatches : 4
- InstancesWithInstalledPatches      : 3
- InstancesWithMissingPatches        : 0
- InstancesWithNotApplicablePatches  : 0

Systems Manager
For API details, see DescribePatchGroupState in AWS Tools for PowerShell Cmdlet Reference.

Get-SSMResourceComplianceSummaryList

The following code example shows how to use Get-SSMResourceComplianceSummaryList.

**Tools for PowerShell**

**Example 1:** This example gets a resource-level summary count. The summary includes information about compliant and non-compliant statuses and detailed compliance-item severity counts for products that match "Windows10". Because the MaxResult default is 100 if the parameter is not specified, and this value is not valid, MaxResult parameter is added, and the value is set to 50.

```
$FilterValues = @{
    "Key"="Product"
    "Type"="EQUAL"
    "Values"="Windows10"
}
Get-SSMResourceComplianceSummaryList -Filter $FilterValues -MaxResult 50
```

For API details, see ListResourceComplianceSummaries in AWS Tools for PowerShell Cmdlet Reference.

Get-SSMResourceTag

The following code example shows how to use Get-SSMResourceTag.

**Tools for PowerShell**

**Example 1:** This example lists the tags for a maintenance window.

```
Get-SSMResourceTag -ResourceId "mw-03eb9db42890fb82d" -ResourceType "MaintenanceWindow"
```

**Output:**

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack</td>
<td>Production</td>
</tr>
</tbody>
</table>
• For API details, see ListTagsForResource in AWS Tools for PowerShell Cmdlet Reference.

New-SSMAntivation

The following code example shows how to use New-SSMAntivation.

Tools for PowerShell

Example 1: This example creates a managed instance.

New-SSMAntivation -DefaultInstanceName "MyWebServers" -IamRole "SSMAutomationRole" -RegistrationLimit 10

Output:

<table>
<thead>
<tr>
<th>ActivationCode</th>
<th>ActivationId</th>
</tr>
</thead>
<tbody>
<tr>
<td>KWChhOxBTiwDcKE9BlKC</td>
<td>08e51e79-1e36-446c-8e63-9458569c1363</td>
</tr>
</tbody>
</table>

• For API details, see CreateActivation in AWS Tools for PowerShell Cmdlet Reference.

New-SSMAssociation

The following code example shows how to use New-SSMAssociation.

Tools for PowerShell

Example 1: This example associates a configuration document with an instance, using instance IDs.

New-SSMAssociation -InstanceId "i-0cb2b964d3e14fd9f" -Name "AWS-UpdateSSMAgent"

Output:

<table>
<thead>
<tr>
<th>Name</th>
<th>AWS-UpdateSSMAgent</th>
</tr>
</thead>
<tbody>
<tr>
<td>InstanceId</td>
<td>i-0000293ff8c57862</td>
</tr>
<tr>
<td>Date</td>
<td>2/23/2017 6:55:22 PM</td>
</tr>
<tr>
<td>Status.Name</td>
<td>Associated</td>
</tr>
<tr>
<td>Status.Date</td>
<td>2/20/2015 8:31:11 AM</td>
</tr>
<tr>
<td>Status.Message</td>
<td>Associated with AWS-UpdateSSMAgent</td>
</tr>
</tbody>
</table>
Example 2: This example associates a configuration document with an instance, using targets.

```powershell
$target = @{Key="instanceids";Values=@("i-0cb2b964d3e14fd9f")}
New-SSMAssociation -Name "AWS-UpdateSSMAgent" -Target $target
```

Output:

```
Name                  : AWS-UpdateSSMAgent
InstanceId            :
Date                  : 3/1/2017 6:22:21 PM
Status.Name           :
Status.Date           :
Status.Message        :
Status.AdditionalInfo :
```

Example 3: This example associates a configuration document with an instance, using targets and parameters.

```powershell
$target = @{Key="instanceids";Values@="i-0cb2b964d3e14fd9f")
$params = @{
    "action"="configure"
    "mode"="ec2"
    "optionalConfigurationSource"="ssm"
    "optionalConfigurationLocation"=""
    "optionalRestart"="yes"
}
New-SSMAssociation -Name "Configure-CloudWatch" -AssociationName "CWConfiguration" -Target $target -Parameter $params
```

Output:

```
Name                  : Configure-CloudWatch
InstanceId            :
Date                  : 5/17/2018 3:17:44 PM
Status.Name           :
Status.Date           :
Status.Message        :
Status.AdditionalInfo :
```
Example 4: This example creates an association with all instances in the region, with AWS-GatherSoftwareInventory. It also provides custom files and registry locations in the parameters to collect

```powershell
$params = [Collections.Generic.Dictionary[String,Collections.Generic.List[String]]]::new() $params["windowsRegistry"] = '[["Path":"HKEY_LOCAL_MACHINE\SOFTWARE\Amazon\MachineImage","Recursive":false,"ValueNames":["AMIName"]]
$params["files"] = '[["Path":"C:\Program Files","Pattern":
["*.exe"],"Recursive":true}, {{"Path":"C:\ProgramData","Pattern":
["*.log"],"Recursive":true}]
New-SSMAssociation -AssociationName new-in-mum -Name AWS-GatherSoftwareInventory -Target @{Key="instanceids";Values="*"} -Parameter $params -region ap-south-1 -ScheduleExpression "rate(720 minutes)"
```

Output:

<table>
<thead>
<tr>
<th>Name</th>
<th>AWS-GatherSoftwareInventory</th>
</tr>
</thead>
<tbody>
<tr>
<td>InstanceId</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>6/9/2019 8:57:56 AM</td>
</tr>
<tr>
<td>Status.Name</td>
<td></td>
</tr>
<tr>
<td>Status.Date</td>
<td></td>
</tr>
<tr>
<td>Status.Message</td>
<td></td>
</tr>
<tr>
<td>Status.AdditionalInfo</td>
<td></td>
</tr>
</tbody>
</table>

- For API details, see [CreateAssociation](#) in AWS Tools for PowerShell Cmdlet Reference.

**New-SSMAssociationFromBatch**

The following code example shows how to use New-SSMAssociationFromBatch.

**Tools for PowerShell**

Example 1: This example associates a configuration document with multiple instances. The output returns a list of successful and failed operations, if applicable.

```powershell
$option1 = @{InstanceId="i-0cb2b964d3e14fd9f";Name="AWS-UpdateSSMAgent"} $option2 = @{InstanceId="i-0000293fffd8c57862";Name="AWS-UpdateSSMAgent"} New-SSMAssociationFromBatch -Entry $option1,$option2
```

Output:
Example 2: This example will show the full details of a successful operation.

```powershell
$option1 = @{InstanceId="i-0cb2b964d3e14fd9f";Name=@("AWS-UpdateSSMAgent")}
$option2 = @{InstanceId="i-0000293f8c57862";Name=@("AWS-UpdateSSMAgent")}
(New-SSMAssociationFromBatch -Entry $option1,$option2).Successful
```

- For API details, see [CreateAssociationBatch](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**New-SSMDocument**

The following code example shows how to use New-SSMDocument.

**Tools for PowerShell**

**Example 1: This example creates a document in your account. The document must be in JSON format. For more information about writing a configuration document, see Configuration Document in the SSM API Reference.**

```powershell
New-SSMDocument -Content (Get-Content -Raw "c:\temp\RunShellScript.json") -Name "RunShellScript" -DocumentType "Command"
```

**Output:**

```
CreatedDate     : 3/1/2017 1:21:33 AM
DefaultVersion  : 1
Description     : Run an updated script
DocumentType    : Command
DocumentVersion : 1
Hash            : 1d5ce820e999ff051eb4841ed887593daf77120fd76cae0d18a53cc42e4e22c1
HashType        : Sha256
LatestVersion   : 1
Name            : RunShellScript
Owner           : 809632081692
Parameters      : {commands}
PlatformTypes   : {Linux}
```
New-SSMMaintenanceWindow

The following code example shows how to use New-SSMMaintenanceWindow.

Tools for PowerShell

Example 1: This example creates a new maintenance window with the specified name that runs at 4 PM on every Tuesday for 4 hours, with a 1 hour cutoff, and that allows unassociated targets.

```
New-SSMMaintenanceWindow -Name "MyMaintenanceWindow" -Duration 4 -Cutoff 1 -AllowUnassociatedTarget $true -Schedule "cron(0 16 ? * TUE *)"
```

Output:

```
mw-03eb53e1ea7383998
```

For API details, see CreateMaintenanceWindow in AWS Tools for PowerShell Cmdlet Reference.

New-SSMPatchBaseline

The following code example shows how to use New-SSMPatchBaseline.

Tools for PowerShell

Example 1: This example creates a patch baseline that approves patches, seven days after they are released by Microsoft, for managed instances running Windows Server 2019 in a production environment.

```
$rule = New-Object Amazon.SimpleSystemsManagement.Model.PatchRule
$rule.ApproveAfterDays = 7
$ruleFilters = New-Object Amazon.SimpleSystemsManagement.Model.PatchFilterGroup
```

For API details, see CreateDocument in AWS Tools for PowerShell Cmdlet Reference.
$patchFilter.Key="PRODUCT"
$patchFilter.Values="WindowsServer2019"

$severityFilter.Key="MSRC_SEVERITY"
$severityFilter.Values.Add("Critical")
$severityFilter.Values.Add("Important")
$severityFilter.Values.Add("Moderate")

$classificationFilter.Key = "CLASSIFICATION"
$classificationFilter.Values.Add( "SecurityUpdates" )
$classificationFilter.Values.Add( "Updates" )
$classificationFilter.Values.Add( "UpdateRollups" )
$classificationFilter.Values.Add( "CriticalUpdates" )

$ruleFilters.PatchFilters.Add($severityFilter)
$ruleFilters.PatchFilters.Add($classificationFilter)
$ruleFilters.PatchFilters.Add($patchFilter)
$rule.PatchFilterGroup = $ruleFilters

New-SSMPatchBaseline -Name "Production-Baseline-Windows2019" -Description "Baseline containing all updates approved for production systems" -ApprovalRules_PatchRule $rule

Output:

pb-0z4z6221c4296b23z

- For API details, see [CreatePatchBaseline](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Register-SSMDefaultPatchBaseline**

The following code example shows how to use `Register-SSMDefaultPatchBaseline`.

**Tools for PowerShell**

**Example 1: This example registers a patch baseline as the default patch baseline.**

Register-SSMDefaultPatchBaseline -BaselineId "pb-03da896ca3b68b639"
Output:

pb-03da896ca3b68b639

- For API details, see `RegisterDefaultPatchBaseline` in *AWS Tools for PowerShell Cmdlet Reference*.

**Register-SSMPatchBaselineForPatchGroup**

The following code example shows how to use `Register-SSMPatchBaselineForPatchGroup`.

**Tools for PowerShell**

**Example 1:** This example registers a patch baseline for a patch group.

```
Register-SSMPatchBaselineForPatchGroup -BaselineId "pb-03da896ca3b68b639" -PatchGroup "Production"
```

**Output:**

<table>
<thead>
<tr>
<th>BaselineId</th>
<th>PatchGroup</th>
</tr>
</thead>
<tbody>
<tr>
<td>pb-03da896ca3b68b639</td>
<td>Production</td>
</tr>
</tbody>
</table>

- For API details, see `RegisterPatchBaselineForPatchGroup` in *AWS Tools for PowerShell Cmdlet Reference*.

**Register-SSMTTargetWithMaintenanceWindow**

The following code example shows how to use `Register-SSMTTargetWithMaintenanceWindow`.

**Tools for PowerShell**

**Example 1:** This example registers an instance with a maintenance window.

```
$options1 = @
Register-SSMTTargetWithMaintenanceWindow -WindowId "mw-06cf17cbefcb4bf4f" -Target $options1 -OwnerInformation "Single instance" -ResourceType "INSTANCE"
```

**Output:**
Example 2: This example registers multiple instances with a maintenance window.

```powershell
$option1 = @{Key="InstanceIds";Values=["i-0000293ffd8c57862","i-0cb2b964d3e14fd9f"]}
Register-SSMTargetWithMaintenanceWindow -WindowId "mw-06cf17cbefcb4bf4f" -Target $option1 -OwnerInformation "Single instance" -ResourceType "INSTANCE"
```

Output:

```
6ab5c208-9fc4-4697-84b7-b02a6cc25f7d
```

Example 3: This example registers an instance with a maintenance window using EC2 tags.

```powershell
$option1 = @{Key="tag:Environment";Values=["Production"]}
Register-SSMTargetWithMaintenanceWindow -WindowId "mw-06cf17cbefcb4bf4f" -Target $option1 -OwnerInformation "Production Web Servers" -ResourceType "INSTANCE"
```

Output:

```
2994977e-aefb-4a71-beac-df620352f184
```

- For API details, see [RegisterTargetWithMaintenanceWindow](#) in AWS Tools for PowerShell Cmdlet Reference.

Register-SSMTaskWithMaintenanceWindow

The following code example shows how to use Register-SSMTaskWithMaintenanceWindow.

Tools for PowerShell

Example 1: This example registers a task with a maintenance window using an instance ID. The output is the Task ID.

```powershell
$parameters = @{}
$parameterValues.Values = @("Install")
$parameters.Add("Operation", $parameterValues)
```
Register-SSMTaskWithMaintenanceWindow -WindowId "mw-03a342e62c96d31b0" -ServiceRoleArn "arn:aws:iam::123456789012:role/MaintenanceWindowsRole" -MaxConcurrency 1 -MaxError 1 -TaskArn "AWS-RunShellScript" -Target @{ Key="InstanceIds";Values="i-0000293ff8c57862" } -TaskType "RUN_COMMAND" -Priority 10 -TaskParameter $parameters

Output:

f34a2c47-ddfd-4c85-a88d-72366b69af1b

Example 2: This example registers a task with a maintenance window using a target ID. The output is the Task ID.

$parameters = @{}
$parameterValues.Values = @("Install")
$parameters.Add("Operation", $parameterValues)

register-ssmtaskwithmaintenancewindow -WindowId "mw-03a342e62c96d31b0" -ServiceRoleArn "arn:aws:iam::123456789012:role/MaintenanceWindowsRole" -MaxConcurrency 1 -MaxError 1 -TaskArn "AWS-RunShellScript" -Target @{ Key="WindowTargetIds";Values="350d44e6-28cc-44e2-951f-4b2c985838f6" } -TaskType "RUN_COMMAND" -Priority 10 -TaskParameter $parameters

Output:

f34a2c47-ddfd-4c85-a88d-72366b69af1b

Example 3: This example creates a parameter object for the run command document AWS-RunPowerShellScript and creates a task with given maintenance window using target ID. The return output is the task ID.

$parameters.Add("commands",@("ipconfig","dir env:\computername"))
$parameters.Add("executionTimeout",@(3600))

$props = @{
  WindowId = "mw-0123e4cce56ff78ae"
}
ServiceRoleArn = "arn:aws:iam::123456789012:role/MaintenanceWindowsRole"
MaxConcurrency = 1
MaxError = 1
TaskType = "RUN_COMMAND"
TaskArn = "AWS-RunPowerShellScript"
Target = @{Key="WindowTargetIds";Values="fe1234ea-56d7-890b-12f3-456b789bee0f"}
Priority = 1
RunCommand_Parameter = $parameters
Name = "set-via-cmdlet"
}

Register-SSMTaskWithMaintenanceWindow @props

Output:
f1e2ef34-5678-12e3-456a-12334c5c6cbe

Example 4: This example registers an AWS Systems Manager Automation task by using a
document named Create-Snapshots.

$automationParameters = @{}
$automationParameters.Add( "instanceId", @{"{ TARGET_ID }"} )
$automationParameters.Add( "AutomationAssumeRole", @{"arn:aws:iam::111111111111:role/AutomationRole"}" )
$automationParameters.Add( "SnapshotTimeout", @{"PT20M"} )
Register-SSMTaskWithMaintenanceWindow -WindowId mw-123EXAMPLE456` -ServiceRoleArn "arn:aws:iam::123456789012:role/MW-Role"` -MaxConcurrency 1 -MaxError 1 -TaskArn "CreateVolumeSnapshots"` -Target @{$ Key="WindowTargetIds";Values="4b5acdf4-946c-4355-bd68-4329a43a5fd1"} ` -TaskType "AUTOMATION"` -Priority 4` -Automation_DocumentVersion '$DEFAULT' -Automation_Parameter $automationParameters -Name "Create-Snapshots"

• For API details, see RegisterTaskWithMaintenanceWindow in AWS Tools for PowerShell Cmdlet Reference.

Remove-SSMAActivation

The following code example shows how to use Remove-SSMAActivation.
Tools for PowerShell

Example 1: This example deletes an activation. There is no output if the command succeeds.

```
Remove-SSMAssociation -ActivationId "08e51e79-1e36-446c-8e63-9458569c1363"
```

- For API details, see [DeleteActivation](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Remove-SSMAssociation

The following code example shows how to use Remove-SSMAssociation.

Tools for PowerShell

Example 1: This example deletes the association between an instance and a document. There is no output if the command succeeds.

```
Remove-SSMAssociation -InstanceId "i-0cb2b964d3e14fd9f" -Name "AWS-UpdateSSMAgent"
```

- For API details, see [DeleteAssociation](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Remove-SSMDocument

The following code example shows how to use Remove-SSMDocument.

Tools for PowerShell

Example 1: This example deletes a document. There is no output if the command succeeds.

```
Remove-SSMDocument -Name "RunShellScript"
```

- For API details, see [DeleteDocument](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Remove-SSMMaintenanceWindow

The following code example shows how to use Remove-SSMMaintenanceWindow.

Tools for PowerShell

Example 1: This example removes a maintenance window.
Remove-SSMMaintenanceWindow -WindowId "mw-06d59c1a07c022145"

Output:

mw-06d59c1a07c022145

- For API details, see DeleteMaintenanceWindow in AWS Tools for PowerShell Cmdlet Reference.

Remove-SSMParameter

The following code example shows how to use Remove-SSMParameter.

Tools for PowerShell

Example 1: This example deletes a parameter. There is no output if the command succeeds.

Remove-SSMParameter -Name "helloWorld"

- For API details, see DeleteParameter in AWS Tools for PowerShell Cmdlet Reference.

Remove-SSMPatchBaseline

The following code example shows how to use Remove-SSMPatchBaseline.

Tools for PowerShell

Example 1: This example deletes a patch baseline.

Remove-SSMPatchBaseline -BaselineId "pb-045f10b4f382baeda"

Output:

pb-045f10b4f382baeda

- For API details, see DeletePatchBaseline in AWS Tools for PowerShell Cmdlet Reference.

Remove-SSMResourceTag

The following code example shows how to use Remove-SSMResourceTag.
Tools for PowerShell

Example 1: This example removes a tag from a maintenance window. There is no output if the command succeeds.

```powershell
Remove-SSMResourceTag -ResourceId "mw-03eb9db42890fb82d" -ResourceType "MaintenanceWindow" -TagKey "Production"
```

- For API details, see [RemoveTagsFromResource](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Send-SSMCommand

The following code example shows how to use Send-SSMCommand.

Tools for PowerShell

Example 1: This example runs an echo command on a target instance.

```powershell
Send-SSMCommand -DocumentName "AWS-RunPowerShellScript" -Parameter @{commands = "echo helloWorld"} -Target @{Key="instanceids";Values=@("i-0cb2b964d3e14fd9f")}
```

Output:

```
CommandId          : d8d190fc-32c1-4d65-a0df-ff5ff3965524
Comment            :
CompletedCount     : 0
DocumentName       : AWS-RunPowerShellScript
ErrorCount         : 0
ExpiresAfter       : 3/7/2017 10:48:37 PM
InstanceIds        : {}
MaxConcurrency     : 50
MaxErrors          : 0
OutputS3BucketName :
OutputS3KeyPrefix  :
OutputS3Region     :
Parameters         : {[commands,
Amazon.Runtime.Internal.Util.AlwaysSendList`1[System.String]]}
RequestedDateTime  : 3/7/2017 9:48:37 PM
ServiceRole        :
Status             : Pending
```
Example 2: This example shows how to run a command that accepts nested parameters.

```
Send-SSMCommand -DocumentName "AWS-RunRemoteScript" -Parameter
@{ sourceType="GitHub"; sourceInfo='{"owner": "me", "repository": "amazon-ssm", "path": "Examples/Install-Win32OpenSSH"}; "commandLine"=".\Install-Win32OpenSSH.ps1"} -InstanceId i-0cb2b964d3e14fd9f
```

• For API details, see SendCommand in AWS Tools for PowerShell Cmdlet Reference.

Start-SSMAutomationExecution

The following code example shows how to use Start-SSMAutomationExecution.

Tools for PowerShell

Example 1: This example runs a document specifying an Automation role, an AMI source ID, and an Amazon EC2 instance role.

```
Start-SSMAutomationExecution -DocumentName AWS-UpdateLinuxAmi -
Parameter @'AutomationAssumeRole'='arn:aws:iam::123456789012:role/SSMAutomationRole'; 'SourceAmiId'='ami-f173cc91'; 'InstanceIamRole'='EC2InstanceRole'}
```

Output:

```
3a532a4f-0382-11e7-9df7-6f11185f6dd1
```

• For API details, see StartAutomationExecution in AWS Tools for PowerShell Cmdlet Reference.

Start-SSMSession

The following code example shows how to use Start-SSMSession.

Tools for PowerShell

Example 1: This example initiates a connection to a target for a Session Manager session, enabling port forwarding.
Start-SSMSession -Target 'i-064578e5e7454488f' -DocumentName 'AWS-StartPortForwardingSession' -Parameter @{ localPortNumber = '8080'; portNumber = '80' }

Output:

<table>
<thead>
<tr>
<th>SessionId</th>
<th>StreamUrl</th>
</tr>
</thead>
<tbody>
<tr>
<td>random-id0</td>
<td>wss://ssmmessages.amazonaws.com/v1/data-channel/random-id</td>
</tr>
</tbody>
</table>

- For API details, see [StartSession](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Stop-SSMAutomationExecution**

The following code example shows how to use `Stop-SSMAutomationExecution`.

**Tools for PowerShell**

**Example 1:** This example stops an Automation Execution. There is no output if the command succeeds.

```powershell
Stop-SSMAutomationExecution -AutomationExecutionId "4105a4fc-f944-11e6-9d32-8fb2db27a909"
```

- For API details, see [StopAutomationExecution](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Stop-SSMCommand**

The following code example shows how to use `Stop-SSMCommand`.

**Tools for PowerShell**

**Example 1:** This example attempts to cancel a command. There is no output if the operation succeeds.

```powershell
Stop-SSMCommand -CommandId "9ded293e-e792-4440-8e3e-7b8ec5feaa38"
```

- For API details, see [CancelCommand](#) in *AWS Tools for PowerShell Cmdlet Reference*. 

---

*Systems Manager*
Unregister-SSMManagedInstance

The following code example shows how to use Unregister-SSMManagedInstance.

**Tools for PowerShell**

**Example 1:** This example deregisters a managed instance. There is no output if the command succeeds.

```
Unregister-SSMManagedInstance -InstanceId "mi-08ab247cdf1046573"
```

- For API details, see [DeregisterManagedInstance](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Unregister-SSMPatchBaselineForPatchGroup

The following code example shows how to use Unregister-SSMPatchBaselineForPatchGroup.

**Tools for PowerShell**

**Example 1:** This example deregisters a patch group from a patch baseline.

```
Unregister-SSMPatchBaselineForPatchGroup -BaselineId "pb-045f10b4f382baeda" -PatchGroup "Production"
```

**Output:**

<table>
<thead>
<tr>
<th>BaselineId</th>
<th>PatchGroup</th>
</tr>
</thead>
<tbody>
<tr>
<td>pb-045f10b4f382baeda</td>
<td>Production</td>
</tr>
</tbody>
</table>

- For API details, see [DeregisterPatchBaselineForPatchGroup](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Unregister-SSMTargetFromMaintenanceWindow

The following code example shows how to use Unregister-SSMTargetFromMaintenanceWindow.
**Tools for PowerShell**

**Example 1:** This example removes a target from a maintenance window.

```powershell
Unregister-SSMTargetFromMaintenanceWindow -WindowTargetId "6ab5c208-9fc4-4697-84b7-b02a6cc25f7d" -WindowId "mw-06cf17cbefc4b4f4f"
```

**Output:**

<table>
<thead>
<tr>
<th>WindowId</th>
<th>WindowTargetId</th>
</tr>
</thead>
<tbody>
<tr>
<td>mw-06cf17cbefc4b4f4f</td>
<td>6ab5c208-9fc4-4697-84b7-b02a6cc25f7d</td>
</tr>
</tbody>
</table>

- For API details, see [DeregisterTargetFromMaintenanceWindow](https://docs.aws.amazon.com/powershell/latest/ug/api-reference.html) in *AWS Tools for PowerShell Cmdlet Reference*.

**Unregister-SSMTaskFromMaintenanceWindow**

The following code example shows how to use `Unregister-SSMTaskFromMaintenanceWindow`.

**Tools for PowerShell**

**Example 1:** This example removes a task from a maintenance window.

```powershell
Unregister-SSMTaskFromMaintenanceWindow -WindowTaskId "f34a2c47-ddfd-4c85-a88d-72366b69af1b" -WindowId "mw-03a342e62c96d31b0"
```

**Output:**

<table>
<thead>
<tr>
<th>WindowId</th>
<th>WindowTaskId</th>
</tr>
</thead>
<tbody>
<tr>
<td>mw-03a342e62c96d31b0</td>
<td>f34a2c47-ddfd-4c85-a88d-72366b69af1b</td>
</tr>
</tbody>
</table>

- For API details, see [DeregisterTaskFromMaintenanceWindow](https://docs.aws.amazon.com/powershell/latest/ug/api-reference.html) in *AWS Tools for PowerShell Cmdlet Reference*.

**Update-SSMAssociation**

The following code example shows how to use `Update-SSMAssociation`.

Systems Manager
**Tools for PowerShell**

**Example 1: This example updates an association with a new document version.**

```
Update-SSMAssociation -AssociationId "93285663-92df-44cb-9f26-2292d4ecc439" -DocumentVersion "1"
```

**Output:**

<table>
<thead>
<tr>
<th>Name</th>
<th>AWS-UpdateSSMAgent</th>
</tr>
</thead>
<tbody>
<tr>
<td>InstanceId</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>3/1/2017 6:22:21 PM</td>
</tr>
<tr>
<td>Status.Name</td>
<td></td>
</tr>
<tr>
<td>Status.Date</td>
<td></td>
</tr>
<tr>
<td>Status.Message</td>
<td></td>
</tr>
<tr>
<td>Status.AdditionalInfo</td>
<td></td>
</tr>
</tbody>
</table>

- For API details, see [UpdateAssociation](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Update-SSMAssociationStatus**

The following code example shows how to use Update-SSMAssociationStatus.

**Tools for PowerShell**

**Example 1: This example updates the association status of the association between an instance and a configuration document.**

```
Update-SSMAssociationStatus -Name "AWS-UpdateSSMAgent" -InstanceId "i-0000293fffd8c57862" -AssociationStatus_Date "2015-02-20T08:31:11Z" -AssociationStatus_Name "Pending" -AssociationStatus_Message "temporary_status_change" -AssociationStatus_AdditionalInfo "Additional-Config-Needed"
```

**Output:**

<table>
<thead>
<tr>
<th>Name</th>
<th>AWS-UpdateSSMAgent</th>
</tr>
</thead>
<tbody>
<tr>
<td>InstanceId</td>
<td>i-0000293fffd8c57862</td>
</tr>
<tr>
<td>Date</td>
<td>2/23/2017 6:55:22 PM</td>
</tr>
<tr>
<td>Status.Name</td>
<td>Pending</td>
</tr>
<tr>
<td>Status.Date</td>
<td>2/20/2015 8:31:11 AM</td>
</tr>
</tbody>
</table>
• For API details, see UpdateAssociationStatus in AWS Tools for PowerShell Cmdlet Reference.

Update-SSMDocument

The following code example shows how to use Update-SSMDocument.

**Tools for PowerShell**

**Example 1:** This creates a new version of a document with the updated contents of the json file you specify. The document must be in JSON format. You can obtain the document version with the "Get-SSMDocumentVersionList" cmdlet.

```
Update-SSMDocument -Name RunShellScript -DocumentVersion "1" -Content (Get-Content -Raw "c:\temp\RunShellScript.json")
```

**Output:**

```
CreatedDate : 3/1/2017 2:59:17 AM
DefaultVersion : 1
Description : Run an updated script
DocumentType : Command
DocumentVersion : 2
Hash : 1d5ce820e999ff051eb4841ed887593daf77120fd76cae0d18a53cc42e4e22c1
HashType : Sha256
LatestVersion : 2
Name : RunShellScript
Owner : 809632081692
Parameters : {commands}
PlatformTypes : {Linux}
SchemaVersion : 2.0
Sha1 :
Status : Updating
```

• For API details, see UpdateDocument in AWS Tools for PowerShell Cmdlet Reference.

Update-SSMDocumentDefaultVersion

The following code example shows how to use Update-SSMDocumentDefaultVersion.
**Tools for PowerShell**

**Example 1:** This updates the default version of a document. You can obtain the available document versions with the "Get-SSMDocumentVersionList" cmdlet.

```
Update-SSMDocumentDefaultVersion -Name "RunShellScript" -DocumentVersion "2"
```

**Output:**

```
<table>
<thead>
<tr>
<th>DefaultVersion</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>RunShellScript</td>
</tr>
</tbody>
</table>
```

- For API details, see [UpdateDocumentDefaultVersion](#) in **AWS Tools for PowerShell Cmdlet Reference**.

**Update-SSMMaintenanceWindow**

The following code example shows how to use **Update-SSMMaintenanceWindow**.

**Tools for PowerShell**

**Example 1:** This example updates the name of a maintenance window.

```
Update-SSMMaintenanceWindow -WindowId "mw-03eb9db42890fb82d" -Name "My-Renamed-MW"
```

**Output:**

```
AllowUnassociatedTargets : False
Cutoff                   : 1
Duration                 : 2
Enabled                  : True
Name                     : My-Renamed-MW
Schedule                 : cron(0 */30 * * * ? *)
WindowId                 : mw-03eb9db42890fb82d
```

**Example 2:** This example enables a maintenance window.

```
Update-SSMMaintenanceWindow -WindowId "mw-03eb9db42890fb82d" -Enabled $true
```
Output:

<table>
<thead>
<tr>
<th>AllowUnassociatedTargets</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cutoff</td>
<td>1</td>
</tr>
<tr>
<td>Duration</td>
<td>2</td>
</tr>
<tr>
<td>Enabled</td>
<td>True</td>
</tr>
<tr>
<td>Name</td>
<td>My-Renamed-MW</td>
</tr>
<tr>
<td>Schedule</td>
<td>cron(0 */30 * * ? *)</td>
</tr>
<tr>
<td>WindowId</td>
<td>mw-03eb9db42890fb82d</td>
</tr>
</tbody>
</table>

Example 3: This example disables a maintenance window.

Update-SSMMaintenanceWindow -WindowId "mw-03eb9db42890fb82d" -Enabled $false

Output:

<table>
<thead>
<tr>
<th>AllowUnassociatedTargets</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cutoff</td>
<td>1</td>
</tr>
<tr>
<td>Duration</td>
<td>2</td>
</tr>
<tr>
<td>Enabled</td>
<td>False</td>
</tr>
<tr>
<td>Name</td>
<td>My-Renamed-MW</td>
</tr>
<tr>
<td>Schedule</td>
<td>cron(0 */30 * * ? *)</td>
</tr>
<tr>
<td>WindowId</td>
<td>mw-03eb9db42890fb82d</td>
</tr>
</tbody>
</table>

- For API details, see [UpdateMaintenanceWindow](#) in AWS Tools for PowerShell Cmdlet Reference.

Update-SSMManagedInstanceRole

The following code example shows how to use Update-SSMManagedInstanceRole.

Tools for PowerShell

Example 1: This example updates the role of a managed instance. There is no output if the command succeeds.

Update-SSMManagedInstanceRole -InstanceId "mi-08ab247cdf1046573" -IamRole "AutomationRole"

- For API details, see [UpdateManagedInstanceRole](#) in AWS Tools for PowerShell Cmdlet Reference.
Update-SSMPatchBaseline

The following code example shows how to use Update-SSMPatchBaseline.

Tools for PowerShell

Example 1: This example adds two patches as rejected and one patch as approved to an existing patch baseline.

```
Update-SSMPatchBaseline -BaselineId "pb-03da896ca3b68b639" -RejectedPatch "KB2032276","MS10-048" -ApprovedPatch "KB2124261"
```

Output:

- ApprovedPatches : {KB2124261}
- BaselineId : pb-03da896ca3b68b639
- CreatedDate : 3/3/2017 5:02:19 PM
- Description : Baseline containing all updates approved for production systems
- ModifiedDate : 3/3/2017 5:22:10 PM
- Name : Production-Baseline
- RejectedPatches : {KB2032276, MS10-048}

• For API details, see UpdatePatchBaseline in AWS Tools for PowerShell Cmdlet Reference.

Write-SSMComplianceItem

The following code example shows how to use Write-SSMComplianceItem.

Tools for PowerShell

Example 1: This example writes a custom compliance item for the given managed instance

```
$item = [Amazon.SimpleSystemsManagement.Model.ComplianceItemEntry]::new()
$item.Id = "07Jun2019-3"
$item.Severity="LOW"
$item.Status="COMPLIANT"
$item.Title="Fin-test-1 - custom"
Write-SSMComplianceItem -ResourceId mi-012dcb3ecea45b678 -ComplianceType Custom:VSSCompliant2 -ResourceType ManagedInstance -Item $item -ExecutionSummary_ExecutionTime "07-Jun-2019"
```
• For API details, see PutComplianceItems in AWS Tools for PowerShell Cmdlet Reference.

Write-SSMInventory

The following code example shows how to use Write-SSMInventory.

**Tools for PowerShell**

**Example 1:** This example assigns rack location information to an instance. There is no output if the command succeeds.

```powershell
$data.Add("RackLocation", "Bay B/Row C/Rack D/Shelf F")

$items.Add($data)

$customInventoryItem = New-Object Amazon.SimpleSystemsManagement.Model.InventoryItem
$customInventoryItem.CaptureTime = "2016-08-22T10:01:01Z"
$customInventoryItem.Content = $items
$customInventoryItem.TypeName = "Custom:TestRackInfo2"
$customInventoryItem.SchemaVersion = "1.0"

$inventoryItems = @$($customInventoryItem)

Write-SSMInventory -InstanceId "i-0cb2b964d3e14fd9f" -Item $inventoryItems
```

• For API details, see PutInventory in AWS Tools for PowerShell Cmdlet Reference.

Write-SSMPParameter

The following code example shows how to use Write-SSMPParameter.

**Tools for PowerShell**

**Example 1:** This example creates a parameter. There is no output if the command succeeds.

```powershell
Write-SSMPParameter -Name "Welcome" -Type "String" -Value "helloWorld"
```
Example 2: This example changes a parameter. There is no output if the command succeeds.

```
Write-SSMParameter -Name "Welcome" -Type "String" -Value "Good day, Sunshine!" -Overwrite $true
```

- For API details, see [PutParameter](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Amazon Translate examples using Tools for PowerShell**

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with Amazon Translate.

*Actions* are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

*Scenarios* are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

**Topics**

- **Actions**

**Actions**

*ConvertTo-TRNTargetLanguage*

The following code example shows how to use *ConvertTo-TRNTargetLanguage*.

**Tools for PowerShell**

**Example 1: Converts the specified English text to French. The text to convert can also be passed as the -Text parameter.**

```
"Hello World" | ConvertTo-TRNTargetLanguage -SourceLanguageCode en -TargetLanguageCode fr
```
AWS Tools for PowerShell

• For API details, see TranslateText in AWS Tools for PowerShell Cmdlet Reference.

AWS WAFV2 examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with AWS WAFV2.

**Actions** are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

**Scenarios** are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

**Topics**

• **Actions**

**Actions**

**New-WAF2WebACL**

The following code example shows how to use New-WAF2WebACL.

**Tools for PowerShell**

**Example 1:** This command creates a new web ACL named "waf-test". Kindly note that as per service API documentation, 'DefaultAction' is a required property. Hence, value for either '-DefaultAction_Allow' and/or '-DefaultAction_Block' should be specified. Since '-DefaultAction_Allow' and '-DefaultAction_Block' are not the required properties, value '@{}' could be used as placeholder as shown in above example.

```powershell
New-WAF2WebACL -Name "waf-test" -Scope REGIONAL -Region eu-west-1 -VisibilityConfig_CloudWatchMetricsEnabled $true -VisibilityConfig_SampledRequestsEnabled $true -VisibilityConfig_MetricName "waf-test" -Description "Test" -DefaultAction_Allow @{}
```

**Output:**
For API details, see CreateWebAcl in AWS Tools for PowerShell Cmdlet Reference.

WorkSpaces examples using Tools for PowerShell

The following code examples show you how to perform actions and implement common scenarios by using the AWS Tools for PowerShell with WorkSpaces.

*Actions* are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

*Scenarios* are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

- **Actions**

Actions

**Approve-WKSIpRule**

The following code example shows how to use Approve-WKSIpRule.

**Tools for PowerShell**

**Example 1: This sample adds rules to an existing IP Group**

```powershell
$Rule = @(
    @{IPRule = "10.1.0.0/0"; RuleDesc = "First Rule Added"},
    @{IPRule = "10.2.0.0/0"; RuleDesc = "Second Rule Added"}
)```
Approve-WKSIPRule -GroupId wsipg-abcnx2fcw -UserRule $Rule

- For API details, see AuthorizeIpRules in AWS Tools for PowerShell Cmdlet Reference.

Copy-WKSWorkspaceImage

The following code example shows how to use Copy-WKSWorkspaceImage.

**Tools for PowerShell**

**Example 1:** This sample copies workspace Image with specified ID from us-west-2 to the current region with the name "CopiedImageTest"

```powershell
Copy-WKSWorkspaceImage -Name CopiedImageTest -SourceRegion us-west-2 -SourceImageId wsi-djfoedhw6
```

**Output:**

wsi-456abaqfe

- For API details, see CopyWorkspaceImage in AWS Tools for PowerShell Cmdlet Reference.

Edit-WKSClientProperty

The following code example shows how to use Edit-WKSClientProperty.

**Tools for PowerShell**

**Example 1:** This sample enables Reconnection for the Workspaces Client

```powershell
Edit-WKSClientProperty -Region us-west-2 -ClientProperties_ReconnectEnabled "ENABLED" -ResourceId d-123414a369
```

- For API details, see ModifyClientProperties in AWS Tools for PowerShell Cmdlet Reference.

Edit-WKSSelfServicePermission

The following code example shows how to use Edit-WKSSelfServicePermission.
Tools for PowerShell

Example 1: This sample enables self service permissions to Change compute type and Increase Volume Size for the specified Directory

```powershell
Edit-WKSSelfservicePermission -Region us-west-2 -ResourceId d-123454a369 -SelfservicePermissions_ChangeComputeType ENABLED -SelfservicePermissions_IncreaseVolumeSize ENABLED
```

- For API details, see [ModifySelfservicePermissions](#) in AWS Tools for PowerShell Cmdlet Reference.

Edit-WKSWorkspaceAccessProperty

The following code example shows how to use Edit-WKSWorkspaceAccessProperty.

Tools for PowerShell

Example 1: This sample enables Workspace access on Android and Chrome OS for the specified Directory

```powershell
```

- For API details, see [ModifyWorkspaceAccessProperties](#) in AWS Tools for PowerShell Cmdlet Reference.

Edit-WKSWorkspaceCreationProperty

The following code example shows how to use Edit-WKSWorkspaceCreationProperty.

Tools for PowerShell

Example 1: This sample enables Internet Access and Maintenance Mode to true as default values while creating a Workspace

```powershell
Edit-WKSWorkspaceCreationProperty -Region us-west-2 -ResourceId d-123454a369 -WorkspaceCreationProperties_EnableInternetAccess $true -WorkspaceCreationProperties_EnableMaintenanceMode $true
```
For API details, see ModifyWorkspaceCreationProperties in AWS Tools for PowerShell Cmdlet Reference.

### Edit-WKSWorkspaceProperty

The following code example shows how to use Edit-WKSWorkspaceProperty.

**Tools for PowerShell**

**Example 1: This Sample changes the Workspace Running Mode Property to Auto Stop for the specified Workspace**

```
Edit-WKSWorkspaceProperty -WorkspaceId ws-w361s100v -Region us-west-2 -WorkspaceProperties_RunningMode AUTO_STOP
```

For API details, see ModifyWorkspaceProperties in AWS Tools for PowerShell Cmdlet Reference.

### Edit-WKSWorkspaceState

The following code example shows how to use Edit-WKSWorkspaceState.

**Tools for PowerShell**

**Example 1: This sample changes the state of the specified Workspace to Available**

```
Edit-WKSWorkspaceState -WorkspaceId ws-w361s100v -Region us-west-2 -WorkspaceState AVAILABLE
```

For API details, see ModifyWorkspaceState in AWS Tools for PowerShell Cmdlet Reference.

### Get-WKSClientProperty

The following code example shows how to use Get-WKSClientProperty.

**Tools for PowerShell**

**Example 1: This sample gets the Client Properties of the Workspace Client for the specified Directory**

```
Get-WKSClientProperty -ResourceId d-223562a123
```
For API details, see `DescribeClientProperties` in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-WKSIpGroup

The following code example shows how to use `Get-WKSIpGroup`.

**Tools for PowerShell**

**Example 1: This sample gets the details of the specified IP Group in the specified region**

```powershell
Get-WKSIpGroup -Region us-east-1 -GroupId wsipg-8m1234v45
```

**Output:**

```
GroupDesc GroupId         GroupName UserRules
--------- -------         --------- ---------
          wsipg-8m1234v45 TestGroup {Amazon.WorkSpaces.Model.IpRuleItem,
          Amazon.WorkSpaces.Model.IpRuleItem}
```

For API details, see `DescribeIpGroups` in *AWS Tools for PowerShell Cmdlet Reference*.

### Get-WKSTag

The following code example shows how to use `Get-WKSTag`.

**Tools for PowerShell**

**Example 1: This Sample fetches tag for the given Workspace**

```powershell
Get-WKSTag -WorkspaceId ws-w361s234r -Region us-west-2
```

**Output:**

```
Key         Value
---         -----  
auto-delete no
purpose     Workbench
```

For API details, see `DescribeTags` in *AWS Tools for PowerShell Cmdlet Reference*. 
The following code example shows how to use `Get-WKSWorkspace`.

**Tools for PowerShell**

**Example 1:** Retrieves details of all your WorkSpaces to the pipeline.

```powershell
Get-WKSWorkspace
```

**Output:**

```
BundleId                    : wsb-1a2b3c4d
ComputerName                :
DirectoryId                 : d-1a2b3c4d
ErrorCode                   :
ErrorMessage                :
IpAddress                   :
RootVolumeEncryptionEnabled : False
State                       : PENDING
SubnetId                    :
UserName                    : myuser
UserVolumeEncryptionEnabled : False
VolumeEncryptionKey         :
WorkspaceId                 : ws-1a2b3c4d
WorkspaceProperties         : Amazon.WorkSpaces.Model.WorkspaceProperties
```

**Example 2:** This command shows the values of child properties of `WorkSpaceProperties` for a workspace in the us-west-2 region. For more information about the child properties of `WorkSpaceProperties`, see https://docs.aws.amazon.com/workspaces/latest/api/API_WorkspaceProperties.html.

```powershell
(Get-WKSWorkspace -Region us-west-2 -WorkSpaceId ws-xdaf7hc9s).WorkspaceProperties
```

**Output:**

```
ComputeTypeName                     : STANDARD
RootVolumeSizeGib                   : 80
RunningMode                         : AUTO_STOP
RunningModeAutoStopTimeoutInMinutes : 60
UserVolumeSizeGib                   : 50
```
Example 3: This command shows the value of the child property `RootVolumeSizeGib` of `WorkSpaceProperties` for a workspace in the US-West-2 region. The root volume size, in GiB, is 80.

```powershell
```

Output:

```
80
```

- For API details, see [DescribeWorkspaces](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-WKSWorkspaceBundle**

The following code example shows how to use `Get-WKSWorkspaceBundle`.

**Tools for PowerShell**

**Example 1:** This sample fetches details of all the Workspace bundles in the current region

```powershell
Get-WKSWorkspaceBundle
```

Output:

```
BundleId        : wsb-sfhdgv342
ComputeType     : Amazon.WorkSpaces.Model.ComputeType
Description     : This bundle is custom
ImageId         : wsi-235aeqges
LastUpdatedTime : 12/26/2019 06:44:07
Name            : CustomBundleTest
Owner           : 233816212345
RootStorage     : Amazon.WorkSpaces.Model.RootStorage
UserStorage     : Amazon.WorkSpaces.Model.UserStorage
```

- For API details, see [DescribeWorkspaceBundles](#) in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-WKSWorkspaceDirectory**

The following code example shows how to use `Get-WKSWorkspaceDirectory`. 

```powershell
WorkSpaces
```
Tools for PowerShell

Example 1: This sample lists the directory details for registered directories

Get-WKSWorkspaceDirectory

Output:

<table>
<thead>
<tr>
<th>Alias</th>
<th>TestWorkspace</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomerUserName</td>
<td>Administrator</td>
</tr>
<tr>
<td>DirectoryId</td>
<td>d-123414a369</td>
</tr>
<tr>
<td>DirectoryName</td>
<td>TestDirectory.com</td>
</tr>
<tr>
<td>DirectoryType</td>
<td>MicrosoftAD</td>
</tr>
<tr>
<td>DnsIpAddresses</td>
<td>{172.31.43.45, 172.31.2.97}</td>
</tr>
<tr>
<td>IamRoleId</td>
<td>arn:aws:iam::761234567801:role/workspaces_RoleDefault</td>
</tr>
<tr>
<td>IpGroupIds</td>
<td>{}</td>
</tr>
<tr>
<td>RegistrationCode</td>
<td>WSpdx+4RRT43</td>
</tr>
<tr>
<td>SelfservicePermissions</td>
<td>Amazon.WorkSpaces.Model.SelfservicePermissions</td>
</tr>
<tr>
<td>State</td>
<td>REGISTERED</td>
</tr>
<tr>
<td>SubnetIds</td>
<td>{subnet-1m3m7b43, subnet-ard11aba}</td>
</tr>
<tr>
<td>Tenancy</td>
<td>SHARED</td>
</tr>
<tr>
<td>WorkspaceCreationProperties</td>
<td>Amazon.WorkSpaces.Model.DefaultWorkspaceCreationProperties</td>
</tr>
<tr>
<td>WorkspaceSecurityGroupId</td>
<td>sg-0ed244123a123c43</td>
</tr>
</tbody>
</table>

For API details, see DescribeWorkspaceDirectories in AWS Tools for PowerShell Cmdlet Reference.

Get-WKSWorkspaceImage

The following code example shows how to use Get-WKSWorkspaceImage.

Tools for PowerShell

Example 1: This sample fetches all the details of all images in the region

Get-WKSWorkspaceImage

Output:
For API details, see `DescribeWorkspaceImages` in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-WKSWorkspaceSnapshot**

The following code example shows how to use `Get-WKSWorkspaceSnapshot`.

**Tools for PowerShell**

*Example 1:* This sample shows the timestamp of the most recent snapshot created for the specified Workspace

```
Get-WKSWorkspaceSnapshot -WorkspaceId ws-w361s100v
```

**Output:**

```
RebuildSnapshots                  RestoreSnapshots
----------------                   ----------------
{Amazon.WorkSpaces.Model.Snapshot} {Amazon.WorkSpaces.Model.Snapshot}
```

For API details, see `DescribeWorkspaceSnapshots` in *AWS Tools for PowerShell Cmdlet Reference*.

**Get-WKSWorkspacesConnectionStatus**

The following code example shows how to use `Get-WKSWorkspacesConnectionStatus`.

**Tools for PowerShell**

*Example 1:* This sample fetches the connection status for the specified Workspace

```
Get-WKSWorkspacesConnectionStatus -WorkspaceId ws-w123s234r
```

**WorkSpaces**
• For API details, see DescribeWorkspacesConnectionStatus in AWS Tools for PowerShell Cmdlet Reference.

**New-WKSIpGroup**

The following code example shows how to use New-WKSIpGroup.

**Tools for PowerShell**

**Example 1:** This sample creates an empty Ip group named FreshEmptyIpGroup

```powershell
New-WKSIpGroup -GroupName "FreshNewIPGroup"
```

**Output:**

wsipg-w45rty4ty

• For API details, see CreateIpGroup in AWS Tools for PowerShell Cmdlet Reference.

**New-WKSTag**

The following code example shows how to use New-WKSTag.

**Tools for PowerShell**

**Example 1:** This example adds a new tag to a workspace named ws-wsname. The tag has a key of "Name", and a key value of AWS_Workspace.

```powershell
$tag = New-Object Amazon.WorkSpaces.Model.Tag
$tag.Key = "Name"
$tag.Value = "AWS_Workspace"
New-WKSTag -Region us-west-2 -WorkspaceId ws-wsname -Tag $tag
```

**Example 2:** This example adds multiple tags to a workspace named ws-wsname. One tag has a key of "Name" and a key value of AWS_Workspace; the other tag has a tag key of "Stage" and a key value of "Test".

```powershell
$tag = New-Object Amazon.WorkSpaces.Model.Tag
```
$tag.Key = "Name"
$tag.Value = "AWS_Workspace"

$tag2.Key = "Stage"
$tag2.Value = "Test"
New-WKSTag -Region us-west-2 -WorkspaceId ws-wsname -Tag $tag,$tag2

- For API details, see [CreateTags](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### New-WKSSWorkspace

The following code example shows how to use `New-WKSSWorkspace`.

#### Tools for PowerShell

**Example 1: Create a WorkSpace for the supplied bundle, directory, and user.**

New-WKSSWorkspace -Workspace @{"BundleID" = "wsb-1a2b3c4d"; "DirectoryId" = "d-1a2b3c4d"; "UserName" = "USERNAME"}

**Example 2: This example creates multiple WorkSpaces**

New-WKSSWorkspace -Workspace @{"BundleID" = "wsb-1a2b3c4d"; "DirectoryId" = "d-1a2b3c4d"; "UserName" = "USERNAME_1"},@{"BundleID" = "wsb-1a2b3c4d"; "DirectoryId" = "d-1a2b3c4d"; "UserName" = "USERNAME_2"}

- For API details, see [CreateWorkspaces](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Register-WKSIpGroup

The following code example shows how to use `Register-WKSIpGroup`.

#### Tools for PowerShell

**Example 1: This sample registers the specified IP Group with the specified Directory**

Register-WKSIpGroup -GroupId wsipg-23ahsdres -DirectoryId d-123412e123

- For API details, see [AssociateIpGroups](#) in *AWS Tools for PowerShell Cmdlet Reference*.
Register-WKSWorkspaceDirectory

The following code example shows how to use Register-WKSWorkspaceDirectory.

Tools for PowerShell

Example 1: This sample registers the specified directory for Workspaces Service

```
Register-WKSWorkspaceDirectory -DirectoryId d-123412a123 -EnableWorkDoc $false
```

- For API details, see [RegisterWorkspaceDirectory](#) in **AWS Tools for PowerShell Cmdlet Reference**.

Remove-WKSIpGroup

The following code example shows how to use Remove-WKSIpGroup.

Tools for PowerShell

Example 1: This sample deletes the specified IP Group

```
Remove-WKSIpGroup -GroupId wsipg-32fhgtred
```

Output:

```
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-WKSIpGroup (DeleteIpGroup)" on target "wsipg-32fhgtred".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): Y
```

- For API details, see [DeleteIpGroup](#) in **AWS Tools for PowerShell Cmdlet Reference**.

Remove-WKSTag

The following code example shows how to use Remove-WKSTag.

Tools for PowerShell

Example 1: This sample removes the tag associated with the Workspace
Remove-WKSTag -ResourceId ws-w10b3abcd -TagKey "Type"

Output:

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-WKSTag (DeleteTags)" on target "ws-w10b3abcd".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): Y

- For API details, see DeleteTags in AWS Tools for PowerShell Cmdlet Reference.

Remove-WKSWorkspace

The following code example shows how to use Remove-WKSWorkspace.

Tools for PowerShell

Example 1: Terminates multiple WorkSpaces. use of the -Force switch stops the cmdlet from prompting for confirmation.

Remove-WKSWorkspace -WorkspaceId "ws-1a2b3c4d5","ws-6a7b8c9d0" -Force

Example 2: Retrieves the collection of all your WorkSpaces and pipes the IDs to the -WorkspaceId parameter of Remove-WKSWorkspace, terminating all of the WorkSpaces. The cmdlet will prompt before each WorkSpace is terminated. To suppress the confirmation prompt add the -Force switch.

Get-WKSWorkspaces | Remove-WKSWorkspace

Example 3: This example shows how to pass TerminateRequest objects defining the WorkSpaces to be terminated. The cmdlet will prompt for confirmation before proceeding, unless the -Force switch parameter is also specified.

$arrRequest = @()
$request1 = New-Object Amazon.WorkSpaces.Model.TerminateRequest
$request1.WorkspaceId = 'ws-12345678'
$arrRequest += $request1
$request2 = New-Object Amazon.WorkSpaces.Model.TerminateRequest
$request2.WorkspaceId = 'ws-abcddefgh'
$arrRequest += $request2
Remove-WKSWorkspace -Request $arrRequest

- For API details, see [TerminateWorkspaces](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Reset-WKSWorkspace

The following code example shows how to use `Reset-WKSWorkspace`.

**Tools for PowerShell**

**Example 1:** Rebuilds the specified WorkSpace.

```powershell
Reset-WKSWorkspace -WorkspaceId "ws-1a2b3c4d"
```

**Example 2:** Retrieves the collection of all your WorkSpaces and pipes the IDs to the `-WorkspaceId` parameter of `Reset-WKSWorkspace`, causing the WorkSpaces to be rebuilt.

```powershell
Get-WKSWorkspaces | Reset-WKSWorkspace
```

- For API details, see [RebuildWorkspaces](#) in *AWS Tools for PowerShell Cmdlet Reference*.

### Restart-WKSWorkspace

The following code example shows how to use `Restart-WKSWorkspace`.

**Tools for PowerShell**

**Example 1:** Reboots the specified WorkSpace.

```powershell
Restart-WKSWorkspace -WorkspaceId "ws-1a2b3c4d"
```

**Example 2:** Reboots multiple WorkSpaces.

```powershell
Restart-WKSWorkspace -WorkspaceId "ws-1a2b3c4d","ws-5a6b7c8d"
```

**Example 3:** Retrieves the collection of all your WorkSpaces and pipes the IDs to the `-WorkspaceId` parameter of `Restart-WKSWorkspace`, causing the WorkSpaces to be restarted.
Get-WKSWorkspaces | Restart-WKSWorkspace

- For API details, see RebootWorkspaces in AWS Tools for PowerShell Cmdlet Reference.

Stop-WKSWorkspace

The following code example shows how to use Stop-WKSWorkspace.

Tools for PowerShell

Example 1: Stops multiple WorkSpaces.

Stop-WKSWorkspace -WorkspaceId "ws-1a2b3c4d5","ws-6a7b8c9d0"

Example 2: Retrieves the collection of all your WorkSpaces and pipes the IDs to the -WorkspaceId parameter of Stop-WKSWorkspace causing the WorkSpaces to be stopped.

Get-WKSWorkspaces | Stop-WKSWorkspace

Example 3: This example shows how to pass StopRequest objects defining the WorkSpaces to be stopped.

```
$arrRequest = @()
$request1 = New-Object Amazon.WorkSpaces.Model.StopRequest
$request1.WorkspaceId = 'ws-12345678'
$arrRequest += $request1
$request2 = New-Object Amazon.WorkSpaces.Model.StopRequest
,request2.WorkspaceId = 'ws-abcddefgh'
$arrRequest += $request2
Stop-WKSWorkspace -Request $arrRequest
```

- For API details, see StopWorkspaces in AWS Tools for PowerShell Cmdlet Reference.

Unregister-WKSIpGroup

The following code example shows how to use Unregister-WKSIpGroup.

Tools for PowerShell

Example 1: This sample unregisters the specified IP Group from the specified Directory
Unregister-WKSIpGroup -GroupId wsipg-12abcdphq -DirectoryId d-123454b123

- For API details, see [DisassociatelIpGroups](#) in *AWS Tools for PowerShell Cmdlet Reference*. 
Security for this AWS Product or Service

Cloud security at Amazon Web Services (AWS) is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations. Security is a shared responsibility between AWS and you. The Shared Responsibility Model describes this as Security of the Cloud and Security in the Cloud.

Security of the Cloud – AWS is responsible for protecting the infrastructure that runs all of the services offered in the AWS Cloud and providing you with services that you can use securely. Our security responsibility is the highest priority at AWS, and the effectiveness of our security is regularly tested and verified by third-party auditors as part of the AWS Compliance Programs.

Security in the Cloud – Your responsibility is determined by the AWS service you are using, and other factors including the sensitivity of your data, your organization’s requirements, and applicable laws and regulations.

This AWS product or service follows the shared responsibility model through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the AWS service security documentation page and AWS services that are in scope of AWS compliance efforts by compliance program.

Topics

- Data protection in this AWS product or service
- Identity and Access Management
- Compliance Validation for this AWS Product or Service
- Enforcing a minimum TLS version in the Tools for PowerShell
- Additional security considerations for the Tools for PowerShell

Data protection in this AWS product or service

The AWS shared responsibility model applies to data protection in this AWS product or service. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy
FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the AWS Security Blog.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a Name field. This includes when you work with this AWS product or service or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Data encryption

A key feature of any secure service is that information is encrypted when it is not being actively used.

Encryption at Rest

The AWS Tools for PowerShell does not itself store any customer data other than the credentials it needs to interact with the AWS services on the user's behalf.
If you use the AWS Tools for PowerShell to invoke an AWS service that transmits customer data to your local computer for storage, then refer to the Security & Compliance chapter in that service's User Guide for information on how that data is stored, protected, and encrypted.

**Encryption in Transit**

By default, all data transmitted from the client computer running the AWS Tools for PowerShell and AWS service endpoints is encrypted by sending everything through an HTTPS/TLS connection.

You don't need to do anything to enable the use of HTTPS/TLS. It is always enabled.

**Identity and Access Management**

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use AWS resources. IAM is an AWS service that you can use with no additional charge.

**Topics**
- Audience
- Authenticating with identities
- Managing access using policies
- How AWS services work with IAM
- Troubleshooting AWS identity and access

**Audience**

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in AWS.

**Service user** – If you use AWS services to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AWS features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in AWS, see [Troubleshooting AWS identity and access](#) or the user guide of the AWS service you are using.

**Service administrator** – If you're in charge of AWS resources at your company, you probably have full access to AWS. It's your job to determine which AWS features and resources your service users
should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with AWS, see the user guide of the AWS service you are using.

IAM administrator – If you’re an IAM administrator, you might want to learn details about how you can write policies to manage access to AWS. To view example AWS identity-based policies that you can use in IAM, see the user guide of the AWS service you are using.

**Authenticating with identities**

Authentication is how you sign in to AWS using your identity credentials. You must be authenticated (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see [How to sign in to your AWS account](#) in the [AWS Sign-In User Guide](#).

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see [Signing AWS API requests](#) in the [IAM User Guide](#).

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see [Multi-factor authentication](#) in the [AWS IAM Identity Center User Guide](#) and [Using multi-factor authentication (MFA) in AWS](#) in the IAM User Guide.
**AWS account root user**

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see [Tasks that require root user credentials](#) in the *IAM User Guide*.

**Federated identity**

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see [What is IAM Identity Center?](#) in the *AWS IAM Identity Center User Guide*.

**IAM users and groups**

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see [Rotate access keys regularly for use cases that require long-term credentials](#) in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.
Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the IAM User Guide.

IAM roles

An IAM role is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the IAM User Guide.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Creating a role for a third-party Identity Provider in the IAM User Guide. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the AWS IAM Identity Center User Guide.

- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.

- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the IAM User Guide.

- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.

- **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the
principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

- **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the IAM User Guide.

- **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the IAM User Guide.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the IAM User Guide.

**Managing access using policies**

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the IAM User Guide.

Administrators can use AWS JSON policies to specify who has access to what. That is, which principal can perform actions on what resources, and under what conditions.
By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

**Identity-based policies**

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Creating IAM policies](#) in the *IAM User Guide*.

Identity-based policies can be further categorized as inline policies or managed policies. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see [Choosing between managed policies and inline policies](#) in the *IAM User Guide*.

**Resource-based policies**

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM role trust policies and Amazon S3 bucket policies. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.
Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the Amazon Simple Storage Service Developer Guide.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the Principal field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the IAM User Guide.

- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the AWS Organizations User Guide.

- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the IAM User Guide.
Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the IAM User Guide.

How AWS services work with IAM

To get a high-level view of how AWS services work with most IAM features, see AWS services that work with IAM in the IAM User Guide.

To learn how to use a specific AWS service with IAM, see the security section of the relevant service's User Guide.

Troubleshooting AWS identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with AWS and IAM.

Topics
- I am not authorized to perform an action in AWS
- I am not authorized to perform iam:PassRole
- I want to allow people outside of my AWS account to access my AWS resources

I am not authorized to perform an action in AWS

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the mateojackson IAM user tries to use the console to view details about a fictional my-example-widget resource but doesn't have the fictional awes:GetWidget permissions.

```plaintext
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
awes:GetWidget on resource: my-example-widget
```

In this case, the policy for the mateojackson user must be updated to allow access to the my-example-widget resource by using the awes:GetWidget action.
If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

**I am not authorized to perform iam:PassRole**

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to AWS.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in AWS. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

**I want to allow people outside of my AWS account to access my AWS resources**

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether AWS supports these features, see [How AWS services work with IAM](https://docs.aws.amazon.com/IAM/latest/UserGuide/id-users-iam.html).
- To learn how to provide access to your resources across AWS accounts that you own, see [Providing access to an IAM user in another AWS account that you own](https://docs.aws.amazon.com/IAM/latest/UserGuide/id-provisioning-external-user.html) in the [IAM User Guide](https://docs.aws.amazon.com/IAM/latest/UserGuide/).  
- To learn how to provide access to your resources to third-party AWS accounts, see [Providing access to AWS accounts owned by third parties](https://docs.aws.amazon.com/IAM/latest/UserGuide/id-provisioning-external-account.html) in the [IAM User Guide](https://docs.aws.amazon.com/IAM/latest/UserGuide/).  
• To learn the difference between using roles and resource-based policies for cross-account access, see [Cross account resource access in IAM](https://docs.aws.amazon.com/iam/latest/userguide/cross-account-access.html) in the IAM User Guide.

### Compliance Validation for this AWS Product or Service

To learn whether an AWS service is within the scope of specific compliance programs, see [AWS services in Scope by Compliance Program](https://aws.amazon.com/compliance/services/) and choose the compliance program that you are interested in. For general information, see [AWS Compliance Programs](https://aws.amazon.com/compliance/).

You can download third-party audit reports using AWS Artifact. For more information, see [Downloading Reports in AWS Artifact](https://docs.aws.amazon.com/auditreport/latest/userguide/why-use-aws-artifact.html).

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- **Security and Compliance Quick Start Guides** – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.

- **Architecting for HIPAA Security and Compliance on Amazon Web Services** – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

  <i>Note</i>

  Not all AWS services are HIPAA eligible. For more information, see the [HIPAA Eligible Services Reference](https://aws.amazon.com/compliance/guides/hipaa-eligible-services/).

- **AWS Compliance Resources** – This collection of workbooks and guides might apply to your industry and location.

- **AWS Customer Compliance Guides** – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).

- **Evaluating Resources with Rules** in the [AWS Config Developer Guide](https://docs.aws.amazon.com/config/latest/developerguide/evaluating-resources.html) – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
• **AWS Security Hub** – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see [Security Hub controls reference](#).

• **Amazon GuardDuty** – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.

• **AWS Audit Manager** – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

This AWS product or service follows the [shared responsibility model](#) through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the [AWS service security documentation page](#) and [AWS services that are in scope of AWS compliance efforts by compliance program](#).

### Enforcing a minimum TLS version in the Tools for PowerShell

To increase security when communicating with AWS services, you should configure the Tools for PowerShell to use the appropriate TLS version. For information about how to do this, see [Enforcing a minimum TLS version](#) in the [AWS SDK for .NET Developer Guide](#).

### Additional security considerations for the Tools for PowerShell

This topic contains security considerations in addition to the security topics covered in earlier sections.

#### Logging of sensitive information

Some operations of this tool might return information that could be considered sensitive, including information from environment variables. The exposure of this information might represent a security risk in certain scenarios; for example, the information could be included in continuous integration and continuous deployment (CI/CD) logs. It is therefore important that you review when you are including such output as part of your logs, and suppress the output when not needed. For additional information about protecting sensitive data, see [Data protection in this AWS product or service](#).
Consider the following best practices:

- Do not use environment variables to store sensitive values for your serverless resources. Instead have your serverless code programmatically retrieve the secret from a secrets store (for example, AWS Secrets Manager).

- Review the contents of your build logs to ensure they do not contain sensitive information. Consider approaches such as piping to /dev/null or capturing the output as a bash or PowerShell variable to suppress command outputs.

- Consider the access of your logs and scope the access appropriately for your use case.
Cmdlet reference for the Tools for PowerShell

The Tools for PowerShell provides cmdlets that you can use to access AWS services. To see what cmdlets are available, see the AWS Tools for PowerShell Cmdlet Reference.
## Document history

This topic describes significant changes to the documentation for the AWS Tools for PowerShell.

We also update the documentation periodically in response to customer feedback. To send feedback about a topic, use the feedback buttons next to "Did this page help you?" located at the bottom of each page.

For additional information about changes and updates to the AWS Tools for PowerShell, see the release notes.

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Code Examples</strong></td>
<td>Included a chapter with cmdlet examples.</td>
<td>April 17, 2024</td>
</tr>
<tr>
<td>**Additional security considera</td>
<td>Including information about potential logging of sensitive data.</td>
<td>April 16, 2024</td>
</tr>
<tr>
<td><strong>Configure tool authentication with AWS</strong></td>
<td>Added information about support for SSO in the AWS Tools for PowerShell.</td>
<td>March 15, 2024</td>
</tr>
<tr>
<td><strong>Cmdlet reference for the Tools for PowerShell</strong></td>
<td>Added section with a link to the Tools for PowerShell cmdlet reference.</td>
<td>November 17, 2023</td>
</tr>
<tr>
<td><strong>Included more IAM best practices updates</strong></td>
<td>Updated guide to align with the IAM best practices. For more information, see Security best practices in IAM.</td>
<td>October 12, 2023</td>
</tr>
<tr>
<td><strong>Installing on Windows</strong></td>
<td>Removed information about installing the Tools for Windows PowerShell by using the MSI, which has been deprecated.</td>
<td>September 25, 2023</td>
</tr>
<tr>
<td>Update Description</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>--------------------------------------------------------------</td>
<td>-----------------</td>
<td></td>
</tr>
<tr>
<td><strong>IAM best practices updates</strong></td>
<td>September 8, 2023</td>
<td></td>
</tr>
<tr>
<td>Updated guide to align with the IAM best practices. For more information, see Security best practices in IAM.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Pipelining and $AWSHistory</strong></td>
<td>March 9, 2023</td>
<td></td>
</tr>
<tr>
<td>Added the IncludeSensitiveData parameter to the Set-AWSHistoryConfiguration cmdlet.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Using the ClientConfig parameter in cmdlets</strong></td>
<td>October 28, 2022</td>
<td></td>
</tr>
<tr>
<td>Added information about support for the ClientConfig parameter.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Launch an Amazon EC2 Instance Using Windows PowerShell</strong></td>
<td>July 26, 2022</td>
<td></td>
</tr>
<tr>
<td>Added notes about retiring EC2-Classic.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>AWS Tools for PowerShell Version 4</strong></td>
<td>November 21, 2019</td>
<td></td>
</tr>
<tr>
<td>Added information about version 4, including installation instructions for both Windows and Linux/macOS, and a migration topic that describes the differences from version 3 and introduces new features.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>AWS Tools for PowerShell 3.3.563</strong></td>
<td>October 18, 2019</td>
<td></td>
</tr>
<tr>
<td>Added information about how to install and use the preview version of the AWS.Tools.Common module. This new module breaks apart the older monolithic package into one shared module and one module per AWS service.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AWS Tools for PowerShell</td>
<td>Added information to the Using the AWS Tools for PowerShell section introducing the AWS Lambda Tools for PowerShell for PowerShell Core developers to build AWS Lambda functions.</td>
<td>September 11, 2018</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>AWS Tools for Windows PowerShell 3.1.31.0</td>
<td>Added information to the Getting Started section about new cmdlets that use Security Assertion Markup Language (SAML) to support configuring federated identity for users.</td>
<td>December 1, 2015</td>
</tr>
<tr>
<td>AWS Tools for Windows PowerShell 2.3.19</td>
<td>Added information to the Cmdlets Discovery and Aliases section about the new Get-AWSCmdletName cmdlet that can help users more easily find their desired AWS cmdlets.</td>
<td>February 5, 2015</td>
</tr>
</tbody>
</table>
Collection output from cmdlets is always enumerated to the PowerShell pipeline. Automatic support for pageable service calls. New $AWSHistory shell variable collects service responses and optionally service requests. AWSRegion instances use Region field instead of SystemName to aid piping. Remove-S3Bucket supports a -DeleteObjects switch option. Fixed usability issue with Set-AWSCredentials. Initialize-AWSDefaults reports from where it obtained credentials and region data. Stop-EC2Instance accepts Amazon.EC2.Model.Reservation instances as input. Generic List<T> parameter types replaced with array types (T[]). Cmdlets that delete or terminate resources prompt for confirmation prior to deletion. Write-S3Object supports in-line text content to upload to Amazon S3.
The install location of the Tools for Windows PowerShell module has changed so that environments using Windows PowerShell version 3 can take advantage of auto-loading. The module and supporting files are now installed to an AWSPowerShell subfolder beneath AWS ToolsPowerShell. Files from previous versions that exist in the AWS ToolsPowerShell folder are automatically removed by the installer. The PSModulePath for Windows PowerShell (all versions) is updated in this release to contain the parent folder of the module (AWS ToolsPowerShell). For systems with Windows PowerShell version 2, the Start Menu shortcut is updated to import the module from the new location and then run Initialize-AWSDefaults. For systems with Windows PowerShell version 3, the Start Menu shortcut is updated to remove the Import-Module command, leaving just Initialize-AWSDefaults. If you edited your PowerShell profile to perform an Import-Mo
dule of the AWSPowerShell.ps1 file, you will need to update it to point to the file's new location (or, if using PowerShell version 3, remove the Import-Module statement as it is no longer needed). As a result of these changes, the Tools for Windows PowerShell module is now listed as an available module when executing Get-Module -ListAvailable. In addition, for users of Windows PowerShell version 3, the execution of any cmdlet exported by the module will automatically load the module in the current PowerShell shell without needing to use Import-Module first. This enables interactive use of the cmdlets on a system with an execution policy that disallows script execution.

AWS Tools for Windows PowerShell 1.0.0.0

Initial release

December 6, 2012