
Amazon Tools for PowerShell

User Guide

Table of Contents

- What are the Amazon Tools for PowerShell? 1
 - Maintenance and support for SDK major versions 1
 - `AWS.Tools` 1
 - `AWSPowerShell.NetCore` 2
 - `AWSPowerShell` 2
 - How to use this guide 3
- Installation 4
 - Prerequisites 4
 - Installing on Windows 5
 - Prerequisites 6
 - Install `AWS.Tools` 6
 - Install `AWSPowerShell.NetCore` 7
 - Install `AWSPowerShell` 8
 - Enable Script Execution 9
 - Versioning 10
 - Updating Amazon Tools for PowerShell 11
 - Installing on Linux or macOS 12
 - Overview of Setup 12
 - Prerequisites 6
 - Install `AWS.Tools` 13
 - Install `AWSPowerShell.NetCore` 14
 - Script Execution 9
 - Configuring the PowerShell Console 16
 - Initialize Your PowerShell Session 16
 - Versioning 10
 - Updating the Amazon Tools for PowerShell on Linux or macOS 17
 - Related Information 18
- Migrating from Amazon Tools for PowerShell Version 3.3 to Version 4 18
 - New Fully Modularized `AWS.Tools` Version 18
 - New `Get-AWSService` cmdlet 18
 - New `-Select` Parameter to Control the Object Returned by a Cmdlet 19
 - More Consistent Limiting of the Number of Items in the Output 20
 - Easier to Use Stream Parameters 20
 - Extending the Pipe by Property Name 21
 - Static Common Parameters 21
 - `AWS.Tools` Declares and Enforces Mandatory Parameters 21
 - All Parameters Are Nullable 21
 - Removing Previously Deprecated Features 22
- Amazon Account and Access Keys 22
 - To get your access key ID and secret access key 22
- Getting Started 24
 - Amazon Credentials 24
 - Credentials Store Locations 24
 - Managing Profiles 25
 - Specifying Credentials 26
 - Credentials Search Order 28
 - Credential Handling in Amazon Tools for PowerShell Core 28
 - Shared Credentials 30
 - Using an IAM Role with Amazon Tools for PowerShell 30
 - Using the Credential Profile Types 31
 - The `ProfilesLocation` Common Parameter 32
 - Displaying Your Credential Profiles 32
 - Removing Credential Profiles 32
 - Important Notes 33

Amazon Regions	33
Specifying a Custom or Nonstandard Endpoint	34
Cmdlet Discovery and Aliases	34
Cmdlet Discovery	34
Cmdlet Naming and Aliases	39
Pipelining and \$AWSHistory	41
\$AWSHistory	41
Configuring Federated Identity	44
Prerequisites	44
How an Identity-Federated User Gets Federated Access to Amazon Service APIs	44
How SAML Support Works in the Amazon Tools for PowerShell	45
How to Use the PowerShell SAML Configuration Cmdlets	45
Additional Reading	48
Using the Amazon Tools for PowerShell	49
PowerShell File Concatenation Encoding	49
Returned Objects for the PowerShell Tools	50
Amazon EC2	50
Amazon S3	50
IAM and Amazon Tools for PowerShell	50
Amazon Lambda and Amazon Tools for PowerShell	51
Amazon SNS and Amazon SQS	51
CloudWatch	51
See Also	51
Amazon S3 and Tools for Windows PowerShell	51
See Also	51
Create an Amazon S3 Bucket, Verify Its Region, and Optionally Remove It	52
Configure an Amazon S3 Bucket as a Website and Enable Logging	53
Upload Objects to an Amazon S3 Bucket	53
Delete Amazon S3 Objects and Buckets	55
Upload In-Line Text Content to Amazon S3	55
IAM and Tools for PowerShell	56
Create New IAM Users and Groups	56
Set an IAM Policy for an IAM User	57
Set an Initial Password for an IAM User	58
Amazon EC2 and Tools for Windows PowerShell	58
Create a Key Pair	59
Create a Security Group	60
Find an AMI	63
Launch an Instance	65
Amazon Lambda and Amazon Tools for PowerShell	68
Prerequisites	6
Install the AWSLambdaPSCore Module	69
See Also	51
Amazon SQS, Amazon SNS and Tools for Windows PowerShell	69
Create an Amazon SQS queue and get queue ARN	70
Create an Amazon SNS topic	70
Give permissions to the SNS topic	70
Subscribe the queue to the SNS topic	71
Give permissions	71
Verify results	71
CloudWatch from the Amazon Tools for Windows PowerShell	72
Publish a Custom Metric to Your CloudWatch Dashboard	72
See Also	51
Security	74
Data protection	74
Data encryption	75
Identity and Access Management	75

Compliance Validation	76
Document History	77

What are the Amazon Tools for PowerShell?

The Amazon Tools for PowerShell are a set of PowerShell modules that are built on the functionality exposed by the Amazon SDK for .NET. The Amazon Tools for PowerShell enable you to script operations on your Amazon 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 Amazon service HTTP query APIs. For example, the cmdlets for the Amazon Tools for PowerShell support PowerShell pipelining—that is, you can pipe PowerShell objects in and out of the cmdlets.

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

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

Note

Amazon Tools for PowerShell version 4 is the latest major release, and is a backward-compatible update to Amazon 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 Amazon Tools for PowerShell Version 3.3 to Version 4 \(p. 18\)](#).

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

- [AWS.Tools \(p. 1\)](#)
- [AWSPowerShell.NetCore \(p. 2\)](#)
- [AWSPowerShell \(p. 2\)](#)

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 [Amazon SDKs and Tools Reference Guide](#):

- [Amazon SDKs and tools maintenance policy](#)
- [Amazon SDKs and tools version support matrix](#)

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

PowerShell Gallery [AWS.Tools](#) ZIP Archive [AWS.Tools](#)

This version of Amazon 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 enables auto-importing of `AWS.Tools` cmdlets with the need to manually call `Import-Module` first.

This is the latest version of Amazon 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 Amazon service, for example, `AWS.Tools.EC2`, `AWS.Tools.IAM`, `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 Amazon 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 Amazon service, you just run the command. PowerShell automatically imports the `AWS.Tools.Common` module and the module for the Amazon 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 Amazon 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 Amazon Tools for PowerShell



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

You can install this version of Amazon 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 Amazon 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 Amazon services.

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

How to use this guide

The guide is divided into the following major sections.

Installing the Amazon Tools for PowerShell (p. 4)

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

Getting Started with the Amazon Tools for Windows PowerShell (p. 24)

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

Using the Amazon Tools for PowerShell (p. 49)

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

Installing the Amazon Tools for PowerShell

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

Topics

- [Prerequisites for Setting up the Amazon Tools for PowerShell](#) (p. 4)
- [Installing the Amazon Tools for PowerShell on Windows](#) (p. 5)
- [Installing Amazon Tools for PowerShell on Linux or macOS](#) (p. 12)
- [Migrating from Amazon Tools for PowerShell Version 3.3 to Version 4](#) (p. 18)
- [Amazon Account and Access Keys](#) (p. 22)

Prerequisites for Setting up the Amazon Tools for PowerShell

To use the Amazon Tools for PowerShell, you must first complete the following steps.

1. Sign up for an Amazon account.

If you don't have an Amazon account, see the following topic for complete instructions on how to sign up:

<http://www.amazonaws.cn/premiumsupport/knowledge-center/create-and-activate-aws-account/>

2. Create an IAM user.

After you sign up for your account, you must create *users* in the Amazon Identity and Access Management (IAM) service. Each user has its own credentials and permissions. The *credentials* are used to authenticate the user making a request. The *permissions* determine which Amazon resources and operations are authorized for that user.

Creating a user is outside the scope of this topic. But if you're new to Amazon, we recommend that you read the following:

- To understand user credentials and best practices for managing them, see [Amazon Security Credentials](#) in the *Amazon Web Services General Reference*.
- For a step-by-step tutorial on creating a user with "administrator" permissions that you can use to run Amazon Tools for PowerShell commands, see [Creating Your First IAM Admin User and Group](#) in the *IAM User Guide*.

3. Create an access key for your IAM user.

The Amazon Tools for PowerShell require that each cmdlet is sent using appropriate security credentials. To do this, you typically must create an access key for each user that needs to use the Amazon Tools for PowerShell cmdlets. An access key consists of an *access key ID* and *secret access key*. These are used to sign (encrypt for the purpose of authentication) programmatic requests that you make to Amazon services. If you don't have an access key, you can create it by using the IAM console at <https://console.amazonaws.cn/iam/>. As described in [Amazon Security Credentials](#), we

recommend that you use access keys for IAM users instead of Amazon root account access keys. IAM lets you securely control access to Amazon services and resources in your Amazon account.

As with any Amazon operation, creating access keys requires that you have permissions to perform the related IAM actions. For more information, see [Permissions for Administering IAM Identities](#) in the *IAM User Guide*.

After you create the access key for your first user in the Amazon console, you can use that user and its access key to run Amazon Tools for PowerShell cmdlets to create access keys for your other users. The following example shows how to use the `New-IAMAccessKey` cmdlet to create an access key and secret key for an IAM user.

```
PS > New-IAMAccessKey -UserName alice

AccessKeyId      : AKIAIOSFODNN7EXAMPLE
CreateDate       : 9/4/19 12:46:18 PM
SecretAccessKey  : wJalrXUtnFEMI/K7MDENG/bPxrRficyEXAMPLEKEY
Status           : Active
UserName        : alice
```

Save these credentials in a safe place. You need them to configure the Amazon Tools for PowerShell credentials file later. For more information, see [Using Amazon Credentials \(p. 24\)](#).

Important

The only time you can see the secret access key (the equivalent of a password) is when you create the access key. You cannot retrieve it later. If you lose the secret key, you must delete the access key/secret key pair and recreate them.

An IAM user can have only two access keys at any one time. If you attempt to create a third set, the `New-IAMAccessKey` cmdlet returns an error. To create another, you must first delete one of the existing two.

You can use the `Remove-IAMAccessKey` cmdlet to delete a set of credentials for an IAM user. You must specify both the `UserName` and the `AccessKeyId`.

```
PS > Remove-IAMAccessKey -UserName alice -AccessKeyId -AccessKeyId AKIAIOSFODNN7EXAMPLE

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-IAMAccessKey (DeleteAccessKey)" on target
"AKIAIOSFODNN7EXAMPLE".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is
"Y"): y
```

Installing the Amazon Tools for PowerShell on Windows

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

- [AWS.Tools \(p. 6\)](#) - The modularized version of Amazon Tools for PowerShell. Each Amazon service is supported by its own individual, small module, with shared support modules `AWS.Tools.Common` and `AWS.Tools.Installer`.
- [AWSPowerShell.NetCore \(p. 7\)](#) - The single, large-module version of Amazon Tools for PowerShell. All Amazon services are supported by this single, large module.
- [AWSPowerShell \(p. 8\)](#) - The legacy Windows-specific, single, large-module version of Amazon Tools for PowerShell. All Amazon 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 (AWSPowerShell module) are installed by default on all Windows-based Amazon Machine Images (AMIs).

Setting up the Amazon Tools for PowerShell involves the following high-level tasks, described in detail in this topic.

1. Install the Amazon 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 Amazon Tools for PowerShell module into your PowerShell session.

Prerequisites

Ensure that you meet the requirements listed in [Prerequisites for Setting up the Amazon Tools for PowerShell](#) (p. 4).

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 Amazon 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` module. The `AWS.Tools.Installer` module simplifies the installation and update of other `AWS.Tools` modules. The `AWS.Tools.Installer` requires, automatically downloads and installs, an updated version of `PowerShellGet`. The `AWS.Tools.Installer` module and automatically keeps your module versions in sync. When you install or update to a newer version of one module, the cmdlets in the `AWS.Tools.Installer` automatically update all of your other `AWS.Tools` modules to the same version.
- Downloading the modules from [AWS.Tools.zip](#) and extracting them in one of the module folders. You can discover your module folders by printing the value of the `$Env:PSModulePath` variable.
- Installing each service module from the PowerShell Gallery using the `Install-Module` cmdlet, as described in the following procedure.

To install AWS.Tools on Windows using the Install-Module cmdlet

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 Amazon service that you want to use by using the `Install-AWSToolsModule` cmdlet. For example, the following command installs the IAM module. 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 Amazon 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"):  
  
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/>) and considers it a trusted source. Use the command `Get-PSRepository -Name PSGallery` for more information about this PSRepository.

By default, this command installs modules into the `$home\Documents\PowerShell\Modules` folder. To install the Amazon Tools for PowerShell for all users of a computer, you must run the following command in a PowerShell session that you started as an administrator. This installs modules to the `$env:ProgramFiles\PowerShell\Modules` folder that is accessible by all users.

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

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](#) 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 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 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 Amazon 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 Amazon 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 Amazon Tools for Windows PowerShell in one of three ways:

- Downloading the module from [AWSPowerShell.zip](#) and extracting it in one of the module directories. You can discover your module directories by printing the value of the `$Env:PSModulePath` variable.
- Running the [Tools for Windows PowerShell installer](#). This method of installing AWSPowerShell is deprecated and we recommend that you use `Install-Module` instead.
- 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 Amazon 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.

```
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 Amazon module's directory, the `Get-Module -ListAvailable` cmdlet shows the module.

```
PS > Get-Module -ListAvailable

ModuleType Name                ExportedCommands
-----
Manifest AppLocker             {}
Manifest BitsTransfer    {}
Manifest PSDiagnostics  {}
Manifest TroubleshootingPack {}
```

```
Manifest   AWSPowerShell           {Update-EBApplicationVersion, Set-DPStatus, Remove-
IAMGroupPol...
```

Versioning

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

```
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 Amazon 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.

```
PS > Get-AWSPowerShellVersion -ListServiceVersionInfo
...

Service                                Noun Prefix Module Name                                SDK
Assembly
Version
-----
-----
Alexa For Business                     ALXB           AWS.Tools.AlexaForBusiness                          3.7.0.11
Amplify Backend                         AMPB           AWS.Tools.AmplifyBackend                            3.7.0.11
Amazon API Gateway                     AG             AWS.Tools.APIGateway                                3.7.0.11
Amazon API Gateway Management API      AGM           AWS.Tools.ApiGatewayManagementApi                  3.7.0.11
Amazon API Gateway V2                  AG2           AWS.Tools.ApiGatewayV2                              3.7.0.11
Amazon Appflow                          AF            AWS.Tools.Appflow                                    3.7.1.4
Amazon Route 53                         R53           AWS.Tools.Route53                                    3.7.0.12
Amazon Route 53 Domains                 R53D          AWS.Tools.Route53Domains                            3.7.0.11
Amazon Route 53 Resolver                R53R          AWS.Tools.Route53Resolver                            3.7.1.5
Amazon Simple Storage Service (S3)     S3            AWS.Tools.S3                                         3.7.0.13
...
```

To determine the version of PowerShell that you are running, enter `$PSVersionTable` to view the contents of the `$PSVersionTable` [automatic variable](#).

```
PS > $PSVersionTable

Name          Value
----          -
PSVersion     6.2.2
PSEdition     Core
GitCommitId   6.2.2
```

```
OS 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
Platform Unix
PSCompatibleVersions {1.0, 2.0, 3.0, 4.0...}
PSRemotingProtocolVersion 2.3
SerializationVersion 1.1.0.1
WSManStackVersion 3.0
```

Updating the Amazon Tools for PowerShell on Windows

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

Update the Modularized `AWS.Tools`

To upgrade 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 Amazon 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](#) website. We suggest you check every two to three weeks. Support for new commands and Amazon 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 the `.msi` package installer or by using a downloaded ZIP file:
 1. Download the most recent version from the [Tools for PowerShell](#) 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 Amazon Tools for PowerShell console from your **Start** menu.

Installing Amazon Tools for PowerShell on Linux or macOS

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

Overview of Setup

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

- [AWS.Tools \(p. 13\)](#) – The modularized version of Amazon Tools for PowerShell. Each Amazon service is supported by its own individual, small module, with shared support modules `AWS.Tools.Common`.
- [AWSPowerShell.NetCore \(p. 14\)](#) – The single, large-module version of Amazon Tools for PowerShell. All Amazon services are supported by this single, large module.

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 Amazon credentials.

Prerequisites

Ensure that you meet the requirements listed on [Prerequisites for Setting up the Amazon Tools for PowerShell \(p. 4\)](#).

To run the Amazon 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](#) 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](#) on Microsoft's website.

Install `AWS.Tools` on Linux or macOS

You can install the modularized version of Amazon 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](#) on the Microsoft PowerShell website.

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

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

To install `AWS.Tools` on Linux or macOS using the `Install-Module` cmdlet

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

```
$ pwsh
```

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 using the `AWS.Tools.Installer` module, 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", 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 IAM module. 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 Amazon 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"):

  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/>) and considers the repository as a trusted source. Use the command `Get-PSRepository -Name PSGallery` for more information about this PSRepository.

By default, this installs modules into the `$home\Documents\PowerShell\Modules` folder. To install the `AWS.Tools` module for all users of a computer, you must run the following command in a PowerShell session that you started as an administrator. This installs modules to the `$env:ProgramFiles\PowerShell\Modules` folder that is accessible by all users.

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

Install AWSPowerShell.NetCore on Linux or macOS

To upgrade to a newer release of `AWSPowerShell.NetCore`, follow the instructions in [Updating the Amazon Tools for PowerShell on Linux or macOS \(p. 17\)](#). Uninstall earlier versions of `AWSPowerShell.NetCore` first.

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 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.

```
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.

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

You don't have to run this command as root, unless you want to install the Amazon 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`.

```
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](#) on the Microsoft Technet website.

Because the `PSModulePath` includes the location of the Amazon module's directory, the `Get-Module -ListAvailable` cmdlet shows the module that you installed.

AWS.Tools

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

Directory: /Users/*username*/.local/share/powershell/Modules

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

Directory: /Users/username/.local/share/powershell/Modules

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

Configure a PowerShell Console to Use the Amazon 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 Amazon Tools for PowerShell, you must run [Initialize-AWSDefaultConfiguration](#) to specify which Amazon access key to use. For more information about `Initialize-AWSDefaultConfiguration`, see [Using Amazon Credentials \(p. 24\)](#).

Note

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

Versioning

Amazon releases new versions of the Amazon Tools for PowerShell periodically to support new Amazon services and features. To determine the version of the Amazon 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 Amazon 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
Name                           Value
----                           -
PSVersion                      6.2.2
PSEdition                      Core
GitCommitId                    6.2.2
OS                              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_ARM_T8020
Platform                       Unix
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
PSRemotingProtocolVersion      2.3
SerializationVersion          1.1.0.1
WSManStackVersion              3.0
```

Updating the Amazon Tools for PowerShell on Linux or macOS

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

Update the Modularized `AWS.Tools.*`

To upgrade 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 Amazon 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

- [Getting Started with the Amazon Tools for Windows PowerShell \(p. 24\)](#)
- [Using the Amazon Tools for PowerShell \(p. 49\)](#)
- [Amazon Account and Access Keys \(p. 22\)](#)

Migrating from Amazon Tools for PowerShell Version 3.3 to Version 4

Amazon Tools for PowerShell version 4 is a backward-compatible update to Amazon 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 Amazon services were supported in the same module, making it very large, and growing larger as each new Amazon 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 Amazon 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.

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.

```
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 Amazon 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 Amazon service APIs for you using the Amazon SDK for .NET. Then the Amazon 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.

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

```
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.

```
PS > Get-S3Object -BucketName mybucket -Select S3Objects.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.

```
PS > Get-S3Object -BucketName mybucket -Select S3Objects.Key |
>> Write-S3ObjectTagSet -Select ^Key -BucketName mybucket -Tagging_TagSet @{ Key='key';
Value='value' }
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 Amazon 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-Items` cmdlet and specify the `-First n` parameter, where *n* is the maximum number of items to include in the final output.

```
PS > Get-S3Object -BucketName mybucket -Select S3Objects.Key | select -first 1*
file1.txt
```

Not all Amazon 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 \(p. 19\)](#) 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.

```
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",  
'})
```

Amazon 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.

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

Note

Some properties supported this in earlier versions of Amazon 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 Amazon Tools for PowerShell, all parameters are static.

In earlier versions of Amazon 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 Amazon 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 Amazon service still requires that information.

```
PS > Get-EC2InstanceAttribute -InstanceId $null -Attribute $null
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.
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.

Get-EC2InstanceAttribute : The request must contain the parameter instanceId
```

Removing Previously Deprecated Features

The following features were deprecated in previous releases of Amazon 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`.

Amazon Account and Access Keys

To access Amazon, you will need to sign up for an Amazon account.

Access keys consist of an *access key ID* and *secret access key*, which are used to sign programmatic requests that you make to Amazon. If you don't have access keys, you can create them by using the IAM console at <https://console.amazonaws.cn/iam/>. We recommend that you use IAM access keys instead of Amazon root account access keys. IAM lets you securely control access to Amazon services and resources in your Amazon account.

Note

To create access keys, you must have permissions to perform the required IAM actions. For more information, see [Granting IAM User Permission to Manage Password Policy and Credentials](#) in the *IAM User Guide*.

To get your access key ID and secret access key

1. Open the IAM console at <https://console.amazonaws.cn/iam/>.
2. On the navigation menu, choose **Users**.
3. Choose your IAM user name (not the check box).
4. Open the **Security credentials** tab, and then choose **Create access key**.
5. To see the new access key, choose **Show**. Your credentials resemble the following:
 - Access key ID: AKIAIOSFODNN7EXAMPLE
 - Secret access key: wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY

6. To download the key pair, choose **Download .csv file**. Store the .csv file with keys in a secure location.

Important

- Keep the keys confidential to protect your Amazon account, and never email them. Do not share them outside your organization, even if an inquiry appears to come from Amazon or Amazon.com. *No one who legitimately represents Amazon will ever ask you for your secret key.*
- You can retrieve the secret access key **only** when you initially create the key pair. Like a password, you can't retrieve it later. If you lose it, you must create a new key pair.

Related topics

- [What Is IAM?](#) in the *IAM User Guide*.
- [Amazon Security Credentials](#) in the *Amazon Web Services General Reference*.

Getting Started with the Amazon Tools for Windows PowerShell

This section describes fundamentals of using the Tools for Windows PowerShell. For example, it explains how to specify which credentials and Amazon Region the Tools for Windows PowerShell should use when interacting with Amazon. This section also provides guidance for using standard PowerShell cmdlets such as `Get-Command` to discover Amazon cmdlets.

Topics

- [Using Amazon Credentials \(p. 24\)](#)
- [Shared Credentials in Amazon Tools for PowerShell \(p. 30\)](#)
- [Specifying Amazon Regions \(p. 33\)](#)
- [Cmdlet Discovery and Aliases \(p. 34\)](#)
- [Pipelining and `\$AWSHistory` \(p. 41\)](#)
- [Configuring Federated Identity with the Amazon Tools for PowerShell \(p. 44\)](#)

Using Amazon Credentials

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

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 Amazon Tools for PowerShell retrieves the associated credentials. For a general discussion of how to safely manage Amazon credentials, see [Best Practices for Managing Amazon Access Keys](#) in the *Amazon Web Services General Reference*.

Note

You need an Amazon account to get credentials and use the Amazon Tools for PowerShell. For information about how to sign up for an account, see [Amazon Account and Access Keys \(p. 22\)](#).

Topics

- [Credentials Store Locations \(p. 24\)](#)
- [Managing Profiles \(p. 25\)](#)
- [Specifying Credentials \(p. 26\)](#)
- [Credentials Search Order \(p. 28\)](#)
- [Credential Handling in Amazon Tools for PowerShell Core \(p. 28\)](#)

Credentials Store Locations

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

- The Amazon 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 [Amazon SDK for .NET](#) and [Toolkit for Visual Studio](#) can also use the Amazon 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 Amazon SDKs and the Amazon Command Line Interface can also use the credentials file. If you're running a script outside of your Amazon 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 Amazon Tools for PowerShell. You can use Amazon Tools for PowerShell cmdlets to manage your profiles in the Amazon SDK store. You can also manage profiles in the Amazon SDK store by using the [Toolkit for Visual Studio](#) or programmatically by using the [Amazon SDK for .NET](#). For directions about how to manage profiles in the credentials file, see [Best Practices for Managing Amazon Access Keys](#).

Add a New profile

To add a new profile to the Amazon 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.

```
PS > Set-AWSCredential `
    -AccessKey AKIA0123456787EXAMPLE `
    -SecretKey wJalrXUtnFEMI/K7MDENG/bPxrRfiCYEXAMPLEKEY `
    -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 Amazon 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

ProfileName StoreTypeName ProfileLocation
-----
default     SharedCredentialsFile /Users/shirley/.aws/credentials
production  SharedCredentialsFile /Users/shirley/.aws/credentials
test        SharedCredentialsFile /Users/shirley/.aws/credentials
```

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. Amazon Tools for PowerShell locates the profile using a search order that is described in [Credentials Search Order](#) (p. 28).

On Windows, Amazon credentials stored in the Amazon 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 Amazon Tools for PowerShell releases. For backward compatibility, `-StoredCredentials` is still supported.

Default Profile (Recommended)

All Amazon 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 Amazon 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 Amazon 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 Amazon 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 [Specifying Amazon Regions \(p. 33\)](#). The following example specifies a default profile and Region.

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

By default, the Amazon 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 Amazon—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 Amazon 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 Amazon 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, Amazon 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 Amazon SDK store and, if that does not exist, the specified profile from the Amazon 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 Amazon SDK store.
 - b. The default profile in the Amazon shared credentials file.
 - c. The `AWS_PS_Default` profile in the Amazon 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 [Amazon SDK for .NET](#).

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

Credential Handling in Amazon Tools for PowerShell Core

Cmdlets in Amazon Tools for PowerShell Core accept Amazon access and secret keys or the names of credential profiles when they run, similarly to the Amazon Tools for Windows PowerShell. When they run

on Windows, both modules have access to the Amazon 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 Amazon Tools for PowerShell searches for a credential profile, and is also the file where the Amazon Tools for PowerShell stores credential profiles. For more information about the Amazon SDK for .NET credential store file, see [Configuring Amazon 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 Amazon shared credentials file that is used by other Amazon SDKs and the Amazon 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 Amazon Tools for PowerShell Core module uses the Amazon 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 Amazon Tools for PowerShell

The Tools for Windows PowerShell support the use of the Amazon shared credentials file, similarly to the Amazon CLI and other Amazon 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 Amazon shared credential file. This functionality is enabled by a new `Amazon.Runtime.CredentialManagement` namespace.

The new profile types and access to the Amazon 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 Amazon Tools for PowerShell

The Amazon shared credential file enables additional types of access. For example, you can access your Amazon 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 Amazon Tools for PowerShell to use a profile that specified a role, the Amazon 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 a multi-factor authentication (MFA) device or an `ExternalId` code when the role is assumed by a third party.

Parameter Name	Description
ExternalId	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 ExternalId as a parameter when assuming the assigned role. For more information, see How to Use an External ID When Granting Access to Your Amazon Resources to a Third Party in the <i>IAM User Guide</i> .
MfaSerial	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 Amazon in the <i>IAM User Guide</i> .
RoleArn	The ARN of the role to assume for assume role credentials. For more information about creating and using roles, see IAM Roles in the <i>IAM User Guide</i> .
SourceProfile	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 <code>RoleArn</code> parameter.

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	Options
RoleSessionName		
-----	-----	-----
Amazon.Runtime.BasicAWSCredentials	arn:aws:iam::123456789012:role/role-i-want-to-assume	Amazon.Runtime.AssumeRoleAWSCredentialsOptions
aws-dotnet-sdk-session-636238288466144357		

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. Amazon 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`.

```
PS > Get-S3Bucket -ProfileName my_role_profile
```

CreationDate	BucketName
-----	-----
2/27/2017 8:57:53 AM	4ba3578c-f88f-4d8b-b95f-92a8858dac58-bucket1
2/27/2017 10:44:37 AM	2091a504-66a9-4d69-8981-aaef812a02c3-bucket2

Using the Credential Profile Types

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

Credentials Type	Parameters you must use
Basic These are the long term credentials for an IAM user	-AccessKey -SecretKey
Session: These are the short term credentials for an IAM role that you retrieve manually, such as by directly calling the Use-STSRole cmdlet.	-AccessKey -SecretKey -SessionToken
Role: These are are short term credentials for an IAM role that Amazon Tools for PowerShell retrieve for you.	-SourceProfile -RoleArn optional: -ExternalId

Credentials Type	Parameters you must use
	optional: -MfaSerial

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.

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

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\user\.aws\credentials -ProfileName
    basic_profile -AccessKey access_key2 -SecretKey secret_key2
PS > Get-Content C:\Users\user\.aws\credentials

aws_access_key_id=access_key2
aws_secret_access_key=secret_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\user\.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.

Specifying Amazon Regions

There are two ways to specify the Amazon Region to use when running Amazon 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 Amazon cmdlets fail if the Tools for Windows PowerShell can't figure out what Region to use. Exceptions include cmdlets for [Amazon S3 \(p. 51\)](#), Amazon SES, and [IAM and Tools for PowerShell \(p. 56\)](#), which automatically default to a global endpoint.

To specify the region for a single Amazon command

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

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

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

From the PowerShell command prompt, type the following command.

```
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 Amazon CLI commands

From the PowerShell command prompt, type the following command.

```
PS > Get-DefaultAWSRegion

Region      Name                IsShellDefault
-----      -
us-west-2   US West (Oregon)   True
```

To clear the current default Region for all Amazon CLI commands

From the PowerShell command prompt, type the following command.

```
PS > Clear-DefaultAWSRegion
```

To view a list of all available Amazon 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

Region      Name                               IsShellDefault
-----
ap-east-1   Asia Pacific (Hong Kong)         False
ap-northeast-1 Asia Pacific (Tokyo)             False
...
us-east-2   US East (Ohio)                   False
us-west-1   US West (N. California)          False
us-west-2   US West (Oregon)                 True
...
```

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 Amazon 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"
```

Cmdlet Discovery and Aliases

This section shows you how to list services that are supported by the Amazon Tools for PowerShell, how to show the set of cmdlets provided by the Amazon 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 Amazon 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 Amazon service API and an Amazon PowerShell cmdlet. To get the cmdlet name that corresponds to an Amazon service API name, run the Amazon `Get-AWSCmdletName` cmdlet with the `-ApiOperation` parameter and the Amazon service API name. For example, to get all possible cmdlet names that are based on any available `DescribeInstances` Amazon service API, run the following command:

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

CmdletName	ServiceOperation	ServiceName	CmdletNounPrefix
Get-EC2Instance	DescribeInstances	Amazon Elastic Compute Cloud	EC2
Get-GMLInstance	DescribeInstances	Amazon GameLift Service	GML

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 Amazon 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 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"
```

CmdletName	ServiceOperation	ServiceName	CmdletNounPrefix
Get-EC2Instance	DescribeInstances	Amazon Elastic Compute Cloud	EC2

Parameter values in these commands are case-insensitive.

If you do not know the name of either the desired Amazon service API or the Amazon 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
```

CmdletName	ServiceOperation
Approve-ECCacheSecurityGroupIngress	AuthorizeCacheSecurityGroupIngress
Get-ECCacheSecurityGroup	DescribeCacheSecurityGroups
New-ECCacheSecurityGroup	CreateCacheSecurityGroup
Remove-ECCacheSecurityGroup	DeleteCacheSecurityGroup
Revoke-ECCacheSecurityGroupIngress	RevokeCacheSecurityGroupIngress
Add-EC2SecurityGroupToClientVpnTargetNetwork	ApplySecurityGroupsToClientVpnTargetNetwork
Get-EC2SecurityGroup	DescribeSecurityGroups
Get-EC2SecurityGroupReference	DescribeSecurityGroupReferences
Get-EC2StaleSecurityGroup	DescribeStaleSecurityGroups
Grant-EC2SecurityGroupEgress	AuthorizeSecurityGroupEgress
Grant-EC2SecurityGroupIngress	AuthorizeSecurityGroupIngress

New-EC2SecurityGroup		CreateSecurityGroup
Amazon Elastic Compute Cloud	EC2	
Remove-EC2SecurityGroup		DeleteSecurityGroup
Amazon Elastic Compute Cloud	EC2	
Revoke-EC2SecurityGroupEgress		RevokeSecurityGroupEgress
Amazon Elastic Compute Cloud	EC2	
Revoke-EC2SecurityGroupIngress		RevokeSecurityGroupIngress
Amazon Elastic Compute Cloud	EC2	
Update-EC2SecurityGroupRuleEgressDescription		UpdateSecurityGroupRuleDescriptionsEgress
Amazon Elastic Compute Cloud	EC2	
Update-EC2SecurityGroupRuleIngressDescription		UpdateSecurityGroupRuleDescriptionsIngress
Amazon Elastic Compute Cloud	EC2	
Edit-EFSMountTargetSecurityGroup		ModifyMountTargetSecurityGroups
Amazon Elastic File System	EFS	
Get-EFSMountTargetSecurityGroup		DescribeMountTargetSecurityGroups
Amazon Elastic File System	EFS	
Join-ELBSecurityGroupToLoadBalancer		ApplySecurityGroupsToLoadBalancer
Elastic Load Balancing	ELB	
Set-ELB2SecurityGroup		SetSecurityGroups
Elastic Load Balancing V2	ELB2	
Enable-RDSDBSecurityGroupIngress		AuthorizeDBSecurityGroupIngress
Amazon Relational Database Service	RDS	
Get-RDSDBSecurityGroup		DescribeDBSecurityGroups
Amazon Relational Database Service	RDS	
New-RDSDBSecurityGroup		CreateDBSecurityGroup
Amazon Relational Database Service	RDS	
Remove-RDSDBSecurityGroup		DeleteDBSecurityGroup
Amazon Relational Database Service	RDS	
Revoke-RDSDBSecurityGroupIngress		RevokeDBSecurityGroupIngress
Amazon Relational Database Service	RDS	
Approve-RSClusterSecurityGroupIngress		AuthorizeClusterSecurityGroupIngress
Amazon Redshift	RS	
Get-RSClusterSecurityGroup		DescribeClusterSecurityGroups
Amazon Redshift	RS	
New-RSClusterSecurityGroup		CreateClusterSecurityGroup
Amazon Redshift	RS	
Remove-RSClusterSecurityGroup		DeleteClusterSecurityGroup
Amazon Redshift	RS	
Revoke-RSClusterSecurityGroupIngress		RevokeClusterSecurityGroupIngress
Amazon Redshift	RS	

If you know the name of the Amazon service but not the Amazon 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

CmdletName                ServiceOperation
-----
ServiceName                CmdletNounPrefix
-----
Add-EC2SecurityGroupToClientVpnTargetNetwrk  ApplySecurityGroupsToClientVpnTargetNetwork
  Amazon Elastic Compute Cloud EC2
Get-EC2SecurityGroup                DescribeSecurityGroups
  Amazon Elastic Compute Cloud EC2
Get-EC2SecurityGroupReference        DescribeSecurityGroupReferences
  Amazon Elastic Compute Cloud EC2
Get-EC2StaleSecurityGroup            DescribeStaleSecurityGroups
  Amazon Elastic Compute Cloud EC2
Grant-EC2SecurityGroupEgress         AuthorizeSecurityGroupEgress
  Amazon Elastic Compute Cloud EC2
Grant-EC2SecurityGroupIngress        AuthorizeSecurityGroupIngress
  Amazon Elastic Compute Cloud EC2
```


New-EC2SecurityGroup Amazon Elastic Compute Cloud EC2	CreateSecurityGroup
Remove-EC2SecurityGroup Amazon Elastic Compute Cloud EC2	DeleteSecurityGroup
Revoke-EC2SecurityGroupEgress Amazon Elastic Compute Cloud EC2	RevokeSecurityGroupEgress
Revoke-EC2SecurityGroupIngress Amazon Elastic Compute Cloud EC2	RevokeSecurityGroupIngress
Update-EC2SecurityGroupRuleEgressDescription Amazon Elastic Compute Cloud EC2	UpdateSecurityGroupRuleDescriptionsEgress
Update-EC2SecurityGroupRuleIngressDescription Amazon Elastic Compute Cloud EC2	UpdateSecurityGroupRuleDescriptionsIngress

If you know the name of the Amazon Command Line Interface (Amazon CLI) command, you can use the `-AwsCliCommand` parameter and the desired Amazon 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` Amazon CLI command call in the Amazon EC2 service, run the following command:

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

CmdletName          ServiceOperation      ServiceName
-----
Grant-EC2SecurityGroupIngress AuthorizeSecurityGroupIngress Amazon Elastic Compute Cloud
EC2
```

The `Get-AWSCmdletName` cmdlet needs only enough of the Amazon CLI command name to identify the service and the Amazon 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 Amazon 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 Amazon 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*`

CommandType	Name	Version	Source
-----	----	-----	-----
Cmdlet	Add-CWEResourceTag AWSPowerShell.NetCore	3.3.563.1	
Cmdlet	Disable-CWEEventSource AWSPowerShell.NetCore	3.3.563.1	
Cmdlet	Disable-CWERule AWSPowerShell.NetCore	3.3.563.1	
Cmdlet	Enable-CWEEventSource AWSPowerShell.NetCore	3.3.563.1	
Cmdlet	Enable-CWERule AWSPowerShell.NetCore	3.3.563.1	
Cmdlet	Get-CWEEventBus AWSPowerShell.NetCore	3.3.563.1	
Cmdlet	Get-CWEEventBusList AWSPowerShell.NetCore	3.3.563.1	
Cmdlet	Get-CWEEventSource AWSPowerShell.NetCore	3.3.563.1	
Cmdlet	Get-CWEEventSourceList AWSPowerShell.NetCore	3.3.563.1	
Cmdlet	Get-CWEPartnerEventSource AWSPowerShell.NetCore	3.3.563.1	
Cmdlet	Get-CWEPartnerEventSourceAccountList AWSPowerShell.NetCore	3.3.563.1	
Cmdlet	Get-CWEPartnerEventSourceList AWSPowerShell.NetCore	3.3.563.1	
Cmdlet	Get-CWEResourceTag AWSPowerShell.NetCore	3.3.563.1	
Cmdlet	Get-CWERule AWSPowerShell.NetCore	3.3.563.1	
Cmdlet	Get-CWERuleDetail AWSPowerShell.NetCore	3.3.563.1	
Cmdlet	Get-CWERuleNamesByTarget AWSPowerShell.NetCore	3.3.563.1	
Cmdlet	Get-CWETargetsByRule AWSPowerShell.NetCore	3.3.563.1	
Cmdlet	New-CWEEventBus AWSPowerShell.NetCore	3.3.563.1	
Cmdlet	New-CWEPartnerEventSource AWSPowerShell.NetCore	3.3.563.1	
Cmdlet	Remove-CWEEventBus AWSPowerShell.NetCore	3.3.563.1	
Cmdlet	Remove-CWEPartnerEventSource AWSPowerShell.NetCore	3.3.563.1	
Cmdlet	Remove-CWEPermission AWSPowerShell.NetCore	3.3.563.1	
Cmdlet	Remove-CWEResourceTag AWSPowerShell.NetCore	3.3.563.1	
Cmdlet	Remove-CWERule AWSPowerShell.NetCore	3.3.563.1	
Cmdlet	Remove-CWETarget AWSPowerShell.NetCore	3.3.563.1	
Cmdlet	Test-CWEEventPattern AWSPowerShell.NetCore	3.3.563.1	
Cmdlet	Write-CWEEvent AWSPowerShell.NetCore	3.3.563.1	

Cmdlet	Write-CWEPartnerEvent	3.3.563.1
	AWSPowerShell.NetCore	
Cmdlet	Write-CWEPermission	3.3.563.1
	AWSPowerShell.NetCore	
Cmdlet	Write-CWERule	3.3.563.1
	AWSPowerShell.NetCore	
Cmdlet	Write-CWETarget	3.3.563.1
	AWSPowerShell.NetCore	

Cmdlet Naming and Aliases

The cmdlets in the Amazon Tools for PowerShell for each service are based on the methods provided by the Amazon 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 `S3GetObject` method retrieves an Amazon S3 object. However, the `Get-S3Object` cmdlet returns *information* about an Amazon S3 object rather than the object itself.

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

ETag          : "df000002a0fe0000f3c000004EXAMPLE"
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 Amazon Tools for PowerShell, run the `Read-S3Object` cmdlet:

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

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

Note

The cmdlet help for an Amazon cmdlet provides the name of the Amazon 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 Amazon 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

Amazon cmdlets do not support the `-WhatIf` switch.

Aliases

Setup of the Amazon Tools for PowerShell installs an aliases file that contains aliases for many of the Amazon cmdlets. You might find these aliases to be more intuitive than the cmdlet names. For example, service names and Amazon 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 Amazon 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 Amazon 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"
```

CommandType	Name	Version	Source
-----	----	-----	-----
Alias	Add-ASInstances	3.3.343.0	AWSPowerShell
Alias	Add-CTTag	3.3.343.0	AWSPowerShell
Alias	Add-DPTags	3.3.343.0	AWSPowerShell
Alias	Add-DSIpRoutes	3.3.343.0	AWSPowerShell
Alias	Add-ELBTags	3.3.343.0	AWSPowerShell
Alias	Add-EMRTag	3.3.343.0	AWSPowerShell
Alias	Add-ESTag	3.3.343.0	AWSPowerShell
Alias	Add-MLTag	3.3.343.0	AWSPowerShell
Alias	Clear-AWSCredentials	3.3.343.0	AWSPowerShell
Alias	Clear-AWSDefaults	3.3.343.0	AWSPowerShell
Alias	Dismount-ASInstances	3.3.343.0	AWSPowerShell
Alias	Edit-EC2Hosts	3.3.343.0	AWSPowerShell
Alias	Edit-RSCLusterIamRoles	3.3.343.0	AWSPowerShell
Alias	Enable-ORGAllFeatures	3.3.343.0	AWSPowerShell
Alias	Find-CTEvents	3.3.343.0	AWSPowerShell
Alias	Get-ASACases	3.3.343.0	AWSPowerShell
Alias	Get-ASAccountLimits	3.3.343.0	AWSPowerShell
Alias	Get-ASACommunications	3.3.343.0	AWSPowerShell
Alias	Get-ASAServices	3.3.343.0	AWSPowerShell
Alias	Get-ASASeverityLevels	3.3.343.0	AWSPowerShell
Alias	Get-ASATrustedAdvisorCheckRefreshStatuses	3.3.343.0	AWSPowerShell
Alias	Get-ASATrustedAdvisorChecks	3.3.343.0	AWSPowerShell
Alias	Get-ASATrustedAdvisorCheckSummaries	3.3.343.0	AWSPowerShell
Alias	Get-ASLifecycleHooks	3.3.343.0	AWSPowerShell
Alias	Get-ASLifecycleHookTypes	3.3.343.0	AWSPowerShell
Alias	Get-AWSCredentials	3.3.343.0	AWSPowerShell
Alias	Get-CDApplications	3.3.343.0	AWSPowerShell
Alias	Get-CDDeployments	3.3.343.0	AWSPowerShell
Alias	Get-CFCloudFrontOriginAccessIdentities	3.3.343.0	AWSPowerShell
Alias	Get-CFDistributions	3.3.343.0	AWSPowerShell
Alias	Get-CFGConfigRules	3.3.343.0	AWSPowerShell
Alias	Get-CFGConfigurationRecorders	3.3.343.0	AWSPowerShell
Alias	Get-CFGDeliveryChannels	3.3.343.0	AWSPowerShell
Alias	Get-CFInvalidations	3.3.343.0	AWSPowerShell
Alias	Get-CFNAccountLimits	3.3.343.0	AWSPowerShell
Alias	Get-CFNStackEvents	3.3.343.0	AWSPowerShell

```
...
```

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 Amazon 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.

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 Amazon Regions and your Amazon EC2 machine images (AMIs) in each Region.

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

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

```
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 Amazon 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 Amazon 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 Amazon 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 Amazon 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.

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

CmdletName

Name of the cmdlet.

CmdletStart

Date/Time that the cmdlet was run.

CmdletEnd

Date/Time 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 `$AWSHistory` variable is not created until an Amazon cmdlet making a service call is used. It evaluates to `$null` until that time.

Note

Earlier versions of the Tools for Windows PowerShell emitted data related to service responses as `Note` 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
```

The `-MaxCmdletHistory` parameter sets the maximum number of cmdlets that can be tracked at any time. A value of 0 turns off recording of Amazon 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. All parameters are optional.

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 Amazon cmdlets that are being held in the list to the pipeline.

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

Access the details of the last Amazon cmdlet that was run:

```
PS > $AWSHistory.LastCommand
```

Access the details of the last service response received by the last Amazon cmdlet that was run. If an Amazon 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).

```
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.

```
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.

```
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.

```
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`:

```
#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
```

Configuring Federated Identity with the Amazon Tools for PowerShell

To let users in your organization access Amazon 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 Amazon APIs, without federated API access, you would also have to create Amazon Identity and Access Management (IAM) users, which defeats the purpose of using federation. This topic describes SAML (Security Assertion Markup Language) support in the Amazon Tools for PowerShell that eases your federated access solution.

SAML support in the Amazon Tools for PowerShell lets you provide your users federated access to Amazon 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 Amazon). 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 Amazon 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 Amazon 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 Amazon 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 Amazon Tools for PowerShell installed on your local workstation.

How an Identity-Federated User Gets Federated Access to Amazon 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 Amazon 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. The user's client sends the SAML assertion to the Amazon Security Token Service (STS) as part of a SAML federation request.

4. STS returns a SAML response that contains Amazon temporary credentials for a role the user can assume.
5. The user accesses Amazon service APIs by including those temporary credentials in request made by Amazon Tools for PowerShell.

How SAML Support Works in the Amazon Tools for PowerShell

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

1. Amazon 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 Amazon.
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. Amazon Tools for PowerShell extracts the list of the user's authorized roles from the SAML assertion.
4. Amazon 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 `AmazonAccessKeyId`, `SecretAccessKey`, and `SessionToken`. These credentials last for 3,600 seconds (1 hour).
6. The user now has valid credentials to work with any Amazon service APIs that the user's role is authorized to access. Amazon Tools for PowerShell automatically applies these credentials for any subsequent Amazon API calls, and renews them automatically when they expire.

Note

When the credentials expire, and new credentials are required, Amazon Tools for PowerShell automatically reauthenticates 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, Amazon Tools for PowerShell prompts users to enter their credentials before they can reauthenticate.

How to Use the PowerShell SAML Configuration Cmdlets

Amazon 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 Amazon 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 Amazon service APIs.

Open a new Amazon Tools for PowerShell session. If you are running PowerShell 3.0 or newer, the Amazon 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.

```
PS > Import-Module "C:\Program Files (x86)\AWS Tools\PowerShell\AWSPowerShell\AWSPowerShell.psd1"
```

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.

```
PS > $endpoint = "https://adfs.example.com/adfs/ls/IdpInitiatedSignOn.aspx?loginToRp=urn:amazon:webservices"
```

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`.

```
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.

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

Alternatively, for non-domain-joined accounts, you can provide Active Directory credentials, and then select an Amazon 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).

```
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
[1] 1 - ADFS-Dev [2] 2 - ADFS-Production [?] Help (default is "1"):
```

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.

```
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.

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

How to Use Role Profiles to Run Cmdlets that Require Amazon Credentials

To run cmdlets that require Amazon credentials, you can use role profiles defined in the Amazon shared credential file. Provide the name of a role profile to `Set-AWSCredential` (or as the value for any `ProfileName` parameter in the Amazon Tools for PowerShell) to get temporary Amazon 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 Amazon credentials, no authentication or request for credentials occurs.

You can now use the temporary Amazon credentials that you obtained with the `SAMLDemoProfile` profile to work with Amazon 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 Amazon 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.

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

Instances                                     GroupNames
-----
{TestInstance1}                             {default}
{TestInstance2}                             {}
{TestInstance3}                             {launch-wizard-6}
{TestInstance4}                             {default}
{TestInstance5}                             {}
{TestInstance6}                             {AWS-OpsWorks-Default-Server}
```

Example 2: Change Role Profiles During a PowerShell Session

This example lists all available Amazon S3 buckets in the Amazon account of the role associated with the `SAMLDemoProfile` profile. The example shows that although you might have been using another profile earlier in your Amazon 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.

```
PS > Get-S3Bucket -ProfileName SAMLDemoProfile

CreationDate                BucketName
-----
7/25/2013 3:16:56 AM        mybucket1
4/15/2015 12:46:50 AM       mybucket2
4/15/2015 6:15:53 AM        mybucket3
1/12/2015 11:20:16 PM       mybucket4
```

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), Amazon 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, Amazon 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.

```
PS > (Get-Ec2Instance -ProfileName ADFS-Production -Region ap-southeast-2).Instances |
Select InstanceType, @{Name="Servername";Expression={$_.tags | where key -eq "Name" |
Select Value -Expand Value}}
```

InstanceType	Servername
t2.small	DC2
t1.micro	NAT1
t1.micro	RDGW1
t1.micro	RDGW2
t1.micro	NAT2
t2.small	DC1
t2.micro	BUILD

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 Amazon Developer Forums for [PowerShell Scripting](#) or [.NET Development](#).

Using the Amazon Tools for PowerShell

Topics

- [PowerShell File Concatenation Encoding \(p. 49\)](#)
- [Returned Objects for the PowerShell Tools \(p. 50\)](#)
- [Amazon EC2 \(p. 50\)](#)
- [Amazon S3 \(p. 50\)](#)
- [IAM and Amazon Tools for PowerShell \(p. 50\)](#)
- [Amazon Lambda and Amazon Tools for PowerShell \(p. 51\)](#)
- [Amazon SNS and Amazon SQS \(p. 51\)](#)
- [CloudWatch \(p. 51\)](#)
- [See Also \(p. 51\)](#)
- [Amazon S3 and Tools for Windows PowerShell \(p. 51\)](#)
- [IAM and Tools for PowerShell \(p. 56\)](#)
- [Amazon EC2 and Tools for Windows PowerShell \(p. 58\)](#)
- [Amazon Lambda and Amazon Tools for PowerShell \(p. 68\)](#)
- [Amazon SQS, Amazon SNS and Tools for Windows PowerShell \(p. 69\)](#)
- [CloudWatch from the Amazon Tools for Windows PowerShell \(p. 72\)](#)

This section provides examples of using the Amazon Tools for PowerShell to access Amazon services. These examples help demonstrate how to use the cmdlets to perform actual Amazon tasks.

PowerShell File Concatenation Encoding

Some cmdlets in the Amazon Tools for PowerShell edit existing files or records that you have in Amazon. 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 Amazon CLI commands from within the PowerShell console, the same behavior applies. You can pipe the output of an Amazon 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 Amazon Tools for PowerShell more useful in a native PowerShell environment, the object returned by a Amazon Tools for PowerShell cmdlet is a .NET object, not the JSON text object that is typically returned from the corresponding API in the Amazon 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 Amazon 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 (p. 58)

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 (p. 51)

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.

IAM and Amazon Tools for PowerShell (p. 56)

This section demonstrates basic operations in Amazon Identity and Access Management (IAM) including how to:

- Create an IAM group.
- Create an IAM user.

- Add an IAM user to an IAM group.
- Specify a policy for an IAM user.
- Set a password and credentials for an IAM user.

Amazon Lambda and Amazon Tools for PowerShell (p. 68)

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

Amazon SNS and Amazon SQS (p. 69)

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 (p. 72)

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

- Publish a Custom Metric to Your CloudWatch Dashboard.

See Also

- [Getting Started with the Amazon Tools for Windows PowerShell \(p. 24\)](#)

Amazon S3 and Tools for Windows PowerShell

Topics

- [See Also \(p. 51\)](#)
- [Create an Amazon S3 Bucket, Verify Its Region, and Optionally Remove It \(p. 52\)](#)
- [Configure an Amazon S3 Bucket as a Website and Enable Logging \(p. 53\)](#)
- [Upload Objects to an Amazon S3 Bucket \(p. 53\)](#)
- [Delete Amazon S3 Objects and Buckets \(p. 55\)](#)
- [Upload In-Line Text Content to Amazon S3 \(p. 55\)](#)

In this section, we create a static website using the Amazon 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 [Amazon 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

- [Using the Amazon Tools for PowerShell \(p. 49\)](#)
- [Hosting a Static Website on Amazon S3](#)
- [Amazon S3 Console](#)

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-2

CreationDate      BucketName
-----
8/16/19 8:45:38 PM website-example
```

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

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

Value
-----
us-west-2
```

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

- [Using the Amazon Tools for PowerShell \(p. 49\)](#)
- [Put Bucket \(Amazon S3 Service Reference\)](#)
- [Amazon 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](#) (p. 53).

```
PS > Write-S3BucketWebsite -BucketName website-example -  
WebsiteConfiguration_IndexDocumentSuffix index.html -WebsiteConfiguration_ErrorDocument  
error.html  
RequestId      : A1813E27995FFDDD  
AmazonId2      : T7hlDOeLqA5Q2XfTe8j2q3SLoP3/5XwhUU3RyJBGHU/LnC+CIWLeGgP0MY24xAlI  
ResponseStream :  
Headers        : {x-amz-id-2, x-amz-request-id, Content-Length, Date...}  
Metadata       : {}  
ResponseXml    :
```

See Also

- [Using the Amazon Tools for PowerShell](#) (p. 49)
- [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>  
>>       Hello, World!  
>>     </p>  
>>   </body>  
>> </html>  
>> @"  
>>  
PS > $index_html | Set-Content index.html  
PS > $error_html = @"  
>> <html>  
>>   <body>  
>>     <p>  
>>       This is an error page.  
>>     </p>  
>>   </body>  
>> </html>  
>> @"  
>>
```

```
>>#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
>> }
>>
PS > # Verify that the files were uploaded
PS > 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 Amazon SDK for .NET. Note, however, that these are different from the values used by the Amazon S3Put Object 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

- [Using the Amazon Tools for PowerShell \(p. 49\)](#)
- [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 > 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.

```
PS > Remove-S3Bucket -BucketName website-example  
  
RequestId      : E480ED92A2EC703D  
AmazonId2      : k6tqaqC1nMkoeYwbuJXUx1/UDa49BJd6dfLN0Ls1mWYNPHjbc8/Nyvm6AGbWcc2P  
ResponseStream :  
Headers        : {x-amz-id-2, x-amz-request-id, Date, Server}  
Metadata       : {}  
ResponseXml    :
```

In 1.1 and newer versions of the Amazon 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, Amazon Tools for PowerShell prompts you for confirmation before the cmdlet runs.

See Also

- [Using the Amazon Tools for PowerShell \(p. 49\)](#)
- [Delete Object \(Amazon S3 API Reference\)](#)
- [DeleteBucket \(Amazon S3 API Reference\)](#)

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
```

IAM and Tools for PowerShell

This section describes some common tasks related to Amazon Identity and Access Management (IAM) and how to perform them using the Amazon Tools for PowerShell.

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.

Topics

- [Create New IAM Users and Groups \(p. 56\)](#)
- [Set an IAM Policy for an IAM User \(p. 57\)](#)
- [Set an Initial Password for an IAM User \(p. 58\)](#)

Create New IAM Users and Groups

This section describes how to create a new IAM group and a new IAM user and then add the user to the group.

First, use the `New-IAMGroup` cmdlet to create the group. Although we've included it here, the `-Path` parameter is optional.

```
PS > New-IAMGroup -Path "/ps-created-groups/" -GroupName "powerUsers"

Path          : /ps-created-groups/
GroupName     : powerUsers
GroupId      : AGPAJPHUEYD5XPCGIUH3E
Arn          : arn:aws:iam::455364113843:group/ps-created-groups/powerUsers
CreateDate   : 11/20/2012 3:32:50 PM
```

Next, use the `New-IAMUser` cmdlet to create the user. Similar to the preceding example, the `-Path` parameter is optional.

```
PS > New-IAMUser -Path "/ps-created-users/" -UserName "myNewUser"

Path          : /ps-created-users/
```

```
UserName : myNewUser
UserId   : AIDAJOJSPSPXADHBT7IN6
Arn      : arn:aws:iam::455364113843:user/ps-created-users/myNewUser
CreateDate : 11/20/2012 3:26:31 PM
```

Finally, use the `Add-IAMUserToGroup` cmdlet to add the user to the group.

```
PS > Add-IAMUserToGroup -UserName myNewUser -GroupName powerUsers

ServiceResponse
-----
Amazon.IdentityManagement.Model.AddUserToGroupResponse
```

To verify that the `powerUsers` group contains the `myNewUser`, use the `Get-IAMGroup` cmdlet.

```
PS > Get-IAMGroup -GroupName powerUsers

Group           Users           IsTruncated
-----
Group Marker
-----
Amazon.IdentityManagement.... {myNewUser}    False
```

You can also view IAM users and groups with the Amazon Web Services Management Console

- [Users View](#)
- [Groups View](#)

See Also

- [Using the Amazon Tools for PowerShell \(p. 49\)](#)
- [Adding a New User to Your Amazon Account \(IAM User Guide\)](#)
- [CreateGroup \(IAM Service Reference\)](#)

Set an IAM Policy for an IAM User

The following commands show how to assign an IAM policy to an IAM user. The policy specified below provides the user with "Power User Access". This policy is identical to the *Power User Access* policy template provided in the IAM console. The name for the policy shown below follows the naming convention used for IAM policy templates such as the template for *Power User Access*. The convention is

```
<template name>+<user name>+<date stamp>
```

In order to specify the policy document, we use a PowerShell here-string. We assign the contents of the here-string to a variable and then use the variable as a parameter value in `Write-IAMUserPolicy`.

```
PS > $policyDoc = @"
>> {
>>   "Version": "2012-10-17",
>>   "Statement": [
>>     {
>>       "Effect": "Allow",
>>       "NotAction": "iam:*",
>>       "Resource": "*"
>>     }
>>   ]
>> }
```

```
>>     }
>>   ]
>> }
>> "@
>>

PS > Write-IAMUserPolicy -UserName myNewUser -PolicyName "PowerUserAccess-
myNewUser-201211201605" -PolicyDocument $policyDoc

ServiceResponse
-----
Amazon.IdentityManagement.Model.PutUserPolicyResponse
```

See Also

- [Using the Amazon Tools for PowerShell \(p. 49\)](#)
- [Using Windows PowerShell "Here-Strings"](#)
- [PutUserPolicy](#)

Set an Initial Password for an IAM User

The following example demonstrates how to use the `New-IAMLoginProfile` cmdlet to set an initial password for an IAM user. For more information about character limits and recommendations for passwords, see [Password Policy Options](#) in the *IAM User Guide*.

```
PS > New-IAMLoginProfile -UserName myNewUser -Password "&!123!&"

UserName                               CreateDate
-----                               -
myNewUser                               11/20/2012 4:23:05 PM
```

Use the `Update-IAMLoginProfile` cmdlet to change the password for an IAM user.

See Also

- [Using the Amazon Tools for PowerShell \(p. 49\)](#)
- [Managing Passwords](#)
- [CreateLoginProfile](#)

Amazon EC2 and Tools for Windows PowerShell

You can perform common tasks related to Amazon EC2 using the Amazon 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 [Getting Started with the Amazon Tools for Windows PowerShell \(p. 24\)](#).

Topics

- [Creating a Key Pair \(p. 59\)](#)
- [Create a Security Group Using Windows PowerShell \(p. 60\)](#)
- [Find an Amazon Machine Image Using Windows PowerShell \(p. 63\)](#)

- [Launch an Amazon EC2 Instance Using Windows PowerShell \(p. 65\)](#)

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

Name      MemberType Definition
----      -
Equals    Method      bool Equals(System.Object obj)
GetHashCode Method      int GetHashCode()
GetType   Method      type GetType()
ToString  Method      string ToString()
KeyFingerprint Property    System.String KeyFingerprint {get;set;}
KeyMaterial Property    System.String KeyMaterial {get;set;}
KeyName   Property    System.String KeyName {get;set;}
```

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
KeyFingerprint : 09:06:70:8e:26:b6:e7:ef:8f:fe:4a:1d:bc:9c:6a:63:11:ac:ad:3c
KeyMaterial  : ----BEGIN RSA PRIVATE KEY----
MIIEogIBAAKCAQEAK+ANYUS9c7niNjYfaCn6KYj/D0I6djnFoQE...
Mz6btoxPcE7EMeH1wySup8nouAS9xb19l7+VkD74bN9KmNcPa/Mu...
Zyn4vVe0Q5il/MpkrRogHqOB0rigeTeV5Yc3lv00RFFPu0Kz4kcm...
w3Jg8dKsWn0plOpX7V3sRC02KgJIbejQUvBFGi50QK9bm4tXBIEC...
daxKIAQMtDUdmBDrhR1/YMv8itFe5DiLLbq7Ga+FDcS85NstBa3h...
iuskGkcvGwkcFQkLmRHRoDpPb+OdFsZtjHZDpMVfMA9tT8EdbkEF...
3SrNeqZPsxJJIXoodb3CxLJpg75JU5kyWnb0+sDNVHoJiZCULCr...
GGLLfEgB95KjGik7zEv2Q7K6s+DHclrDeMZwa7KFNRZuCuX7jssC...
xO98abxMr3o3TNU6p1ZYRJEQ0oJrOW+kc+/8SWb8NIwfltwmJJEy...
1BX9X8WFX/A8VLHrT1elrKmlkNECgYEAwltkV1pOJAFhz9p7ZFEv...
vvVsPaF0Ev9bk9pqhx269PB50x2KokwCagDMMaYvasWobuLmNu/1...
lmwRx7KTeQ7WlJ3OLGxHA1QNMkip9c4Tb3q9vVc3t/fPf8vwfJ8C...
63g6N6rk2FkHZX1E62BgbewUd3eZOS05Ip4VUdvtGcuc8/qa+e5C...
KXgyt9nl64pMv+VaXfXkZhdLAdY0Khc9TGB9++VMSG5Trd15YJId...
gYALEI7m1jJKpHWAESOhiemw5VmKyIZpzGstSJsFStERLajiETDH...
YAtnI4J8dRyP9I7BOVOn3wnfIjk85gi1/00c+j8S65giLafndWGR...
9R9wIkm5BMUcSRRcDyOyuwKBgEbkOnGGSD0ah4HkvrUkepIbUDTD...
AnEBM1cXI5UT7BfKInpUihZi59Qhgdk/hkOSmWhlZGwikJ5VizBf...
drkBr/vTKVRMTi31VFB7KkIV1xJxC5E/BZ+YdZEpWocZAoGAC/Cd...
TTld5N6opgOXAcQJwzqoGa9ZMwc5Q9f4bfRc67emkwoZAAwSsvWR...
x302duuy7/smTwWwskEWRK5IrUxoMv/VVYaqdzcOajwieNbrl7c...
-----END RSA PRIVATE KEY-----
```

The `KeyMaterial` member stores the private key for the key pair. The public key is stored in Amazon. You can't retrieve the public key from Amazon, but you can verify the public key by comparing the `KeyFingerprint` for the private key to that returned from Amazon 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.

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

KeyName           : myPSKeyPair
KeyFingerprint    : 09:06:70:8e:26:b6:e7:ef:8f:fe:4a:1d:bc:9c:6a:63:11:ac:ad:3c
```

Storing Your Private Key

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

```
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:

```
PS > openssl rsa -check < myPSKeyPair.pem
```

(The `openssl` tool is not included with the Amazon Tools for PowerShell or the Amazon SDK for .NET.)

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 Amazon, use the `Remove-EC2KeyPair` cmdlet. When prompted, press `Enter` to remove the key pair.

```
PS > Remove-EC2KeyPair -KeyName myPSKeyPair

Confirm
Performing the operation "Remove-EC2KeyPair (DeleteKeyPair)" on target "myPSKeyPair".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):
```

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 Amazon.

Create a Security Group Using Windows PowerShell

You can use the Amazon 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 \(p. 61\)](#)
- [Creating a Security Group for EC2-Classic \(p. 61\)](#)
- [Creating a Security Group for EC2-VPC \(p. 62\)](#)

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/> or <https://checkip.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 for Linux Instances](#) or the [Amazon EC2 User Guide for Windows Instances](#). 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}
OwnerId         : 123456789012
Tags             : {}
VpcId           : vpc-9668ddef
```

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`.

```
#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.

```
PS > Get-EC2SecurityGroup -GroupNames myPSSecurityGroup

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.

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

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.

```
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.

```
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.

```
$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.

```
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.

```
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 Amazon Windows AMIs change frequently because Amazon 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](#) (p. 63)
- [Get-EC2ImageByName](#) (p. 64)

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  
ImageLocation          : amazon/Windows_Server-2019-English-Full-SQL_2017_Web-2019.06.12  
ImageOwnerAlias        : amazon  
ImageType              : machine  
KernelId              :  
Name                  : Windows_Server-2019-English-Full-SQL_2017_Web-2019.06.12  
OwnerId               : 801119661308  
Platform              : Windows  
ProductCodes          : {}  
Public                : True  
RamdiskId             :  
RootDeviceName        : /dev/sda1  
RootDeviceType        : ebs  
SriovNetSupport        : simple  
State                 : available  
StateReason           :  
Tags                  : {}  
VirtualizationType    : hvm
```

Get-EC2ImageByName

The `Get-EC2ImageByName` cmdlet enables you to filter the list of Amazon 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_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
WINDOWS_2008RTM_SQL_SERVER_EXPRESS_2008
WINDOWS_2008RTM_SQL_SERVER_STANDARD_2008
WINDOWS_2008_BEANSTALK_IIS75
WINDOWS_2012_BEANSTALK_IIS8
VPC_NAT
```

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, xvdc, xvdc...}
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 \(p. 59\)](#)
- [Create a Security Group Using Windows PowerShell \(p. 60\)](#)

- [Find an Amazon Machine Image Using Windows PowerShell \(p. 63\)](#)

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 \(p. 66\)](#)
- [Launching an Instance in a VPC \(p. 67\)](#)
- [Launching a Spot Instance in a VPC \(p. 68\)](#)

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 for Linux Instances](#) or the [Amazon EC2 User Guide for Windows Instances](#). 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 `
  -MaxCount 1 `
  -KeyName myPSKeyPair `
  -SecurityGroups myPSSecurityGroup `
  -InstanceType t1.micro

ReservationId : r-b70a0ef1
OwnerId       : 123456789012
RequesterId   :
Groups        : {myPSSecurityGroup}
GroupName     : {myPSSecurityGroup}
Instances     : {}
```

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.

```
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

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     :
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           :
Tags               : {}
VirtualizationType : hvm
VpcId              :
  
```

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.

```

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

ReservationId      : r-b70a0ef1
OwnerId            : 123456789012
RequesterId        :
Groups             : {}
GroupName          : {}
Instances          : {}
  
```

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.

```

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

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  
Tags                 : {}  
VirtualizationType   : hvm  
VpcId                : vpc-a01106c2
```

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.

```
$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
```

Amazon Lambda and Amazon Tools for PowerShell

By using the [AWSLambdaPSCore](#) module, you can develop Amazon Lambda functions in PowerShell Core 6.0 using the .NET Core 2.1 runtime. PowerShell developers can manage Amazon 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 Amazon module for PowerShell; it is not part of the Amazon Tools for PowerShell, nor does installing the AWSLambdaPSCore module install the Amazon 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 Amazon Lambda Tools for PowerShell module includes project templates for PowerShell-based serverless applications, and tools to publish projects to Amazon.

AWSLambdaPSCore module support is available in all regions that support Lambda. For more information about supported regions, see the [Amazon 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 Amazon 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 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 Amazon Lambda Developer Guide.

See Also

- [Announcing Lambda Support for PowerShell Core on the Amazon Developer Blog](#)
- [AWSLambdaPSCore module on the PowerShell Gallery website](#)
- [Setting Up a PowerShell Development Environment](#)
- [Amazon Lambda Tools for Powershell on GitHub](#)
- [Amazon 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 Amazon 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
arn:aws:sns:us-west-2:123456789012: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:

```
# 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:

```
PS > Connect-SNSNotification `
  -TopicARN arn:aws:sns:us-west-2:123456789012:myTopic `
  -Protocol SQS `
  -Endpoint arn:aws:sqs:us-west-2:123456789012:myQueue
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`

```
PS > Add-SNSPermission `
  -TopicArn arn:aws:sns:us-west-2:123456789012:myTopic `
  -Label ps-cmdlet-topic `
  -AWSAccountIds 123456789012 `
  -ActionNames publish
```

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`.

```
PS > Publish-SNSMessage `
  -TopicArn arn:aws:sns:us-west-2:123456789012:myTopic `
  -Message "Have A Nice Day!"
728180b6-f62b-49d5-b4d3-3824bb2e77f4
```

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

```
PS > Receive-SQSMessage -QueueUrl https://sqs.us-west-2.amazonaws.com/123456789012/myQueue
Attributes          : {}
```

```
Body      : {
            "Type" : "Notification",
            "MessageId" : "491c687d-b78d-5c48-b7a0-3d8d769ee91b",
            "TopicArn" : "arn:aws:sns:us-west-2:123456789012:myTopic",
            "Message" : "Have A Nice Day!",
            "Timestamp" : "2019-09-09T21:06:27.201Z",
            "SignatureVersion" : "1",
            "Signature" : "11E17A2+XOuJZnw3TlgcXz4C4KPLXZxbxoEMiirelhl3u/
oxkWmz5+9tJKFMns1ZOqQvKxk+ExfEZcD5yWt6biVuBb8pyRmZ1b03hUEN13ayv2WQIQ1T1vpLpM7VEQN5m+hLIiPFcs
vyuGkJReV71OJWPHnCN
+qTE21Id2RPkFOeGtLGawTsSPTWEvJdDbLlf7E0zZ0q1niXTUtpsZ8Swx01X3Q06u9i9qBFt0ekJFZNJp6Avu05hIk1b4yoRs1IkblV
y0a8Yl9lWp7a7EoWaBn0zhCESe7o
            kZC6ncBJWphX7KCGVYD0qhVf/5VDgBuv9w8T+higJyvr3WbaSvg==",
            "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/
SimpleNotificationService-6aad65c2f9911b05cd53efda11f913f9.pem",
            "UnsubscribeURL" :
            "https://sns.us-west-2.amazonaws.com/?
Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-west-2:123456789012:myTopic:22b77de7-
a216-4000-9a23-bf465744ca84"
        }
MD5OfBody      : 5b5ee4f073e9c618eda3718b594fa257
MD5OfMessageAttributes :
MessageAttributes : {}
MessageId      : 728180b6-f62b-49d5-b4d3-3824bb2e77f4
ReceiptHandle  :
    AQEB2vvk1e5cOKFjeIWJticabkc664yuDEjhucnIOqdVUmie7bX7GiJb17F0enABUGaI2XjEcNPxixhVc/
wfsAJZLNhnl8SlbQaOR/kD+Saqa40Ivfj8x3M4OhlyM1cVKpYmhAzsYrAwAD5g5FvxNBD6zs
    +HmXdkax2Wd+9AxrHlQZV5ur1MoByKWWbDbsqoYJTJquCclOgWiak/sBx/
daBRMTiVQ4GHsrQWMMVhNC14q7Jy/0L2dkmb4dzJfJq0VbFSX1G+u/lrSLpgae+Dfux646y8yFiPFzY4ua4mCF/
SVUn63Spy
    sHN12776axknhg3j9K/Xwj54DixdsegrKoLx+ctI
+0jzAetBR66Q1VhIoJAq7s0a2MseyOem/Jjucg6Sr9VUnTWWVhV8ErXmotoiEg==
```

CloudWatch from the Amazon 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 `MetricDatum` object and posts it to the service. You can see the result of this operation by navigating to the [CloudWatch console](#).

```
$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

- [Using the Amazon Tools for PowerShell \(p. 49\)](#)
- [AmazonCloudWatchClient.PutMetricData \(.NET SDK Reference\)](#)
- [MetricDatum \(Service API Reference\)](#)
- [Amazon CloudWatch Console](#)

Security in the Amazon Tools for PowerShell

Cloud security at Amazon is the highest priority. As an Amazon 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 Amazon and you. The [shared responsibility model](#) describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – Amazon is responsible for protecting the infrastructure that runs Amazon services in the Amazon Cloud. Amazon also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the [Amazon Compliance Programs](#). To learn about the compliance programs that apply to Amazon Tools for PowerShell, see [Amazon Services in Scope by Compliance Program](#).
- **Security in the cloud** – Your responsibility is determined by the Amazon service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using the Amazon Tools for PowerShell. The following topics show you how to configure the Amazon Tools for PowerShell to meet your security and compliance objectives. You also learn how to use the Amazon Tools for PowerShell to help you to monitor and secure your Amazon resources.

Topics

- [Data protection in the Amazon Tools for PowerShell](#) (p. 74)
- [Identity and Access Management for the Amazon Tools for PowerShell](#) (p. 75)
- [Compliance Validation for the Amazon Tools for PowerShell](#) (p. 76)

Data protection in the Amazon Tools for PowerShell

The Amazon [shared responsibility model](#) applies to data protection in the Amazon Tools for PowerShell. As described in this model, Amazon is responsible for protecting the global infrastructure that runs all of the Amazon Web Services Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the Amazon Web Services that you use. For more information about data privacy, see the [Data Privacy FAQ](#).

For data protection purposes, we recommend that you protect Amazon Web Services account credentials and set up individual user accounts with Amazon 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 Amazon resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with Amazon CloudTrail.

- Use Amazon encryption solutions, along with all default security controls within Amazon services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing Amazon 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 fields such as a **Name** field. This includes when you work with the Amazon Tools for PowerShell or other Amazon services using the console, API, Amazon CLI, or Amazon SDKs. Any data that you enter into tags or free-form 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 Amazon Tools for PowerShell does not itself store any customer data other than the credentials it needs to interact with the Amazon services on the user's behalf.

If you use the Amazon Tools for PowerShell to invoke an Amazon 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 Amazon Tools for PowerShell and Amazon 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 for the Amazon Tools for PowerShell

The Amazon Tools for PowerShell uses the same IAM users and roles that you use to access your Amazon resources and their services with the Amazon Web Services Management Console. The policies that grant permissions are also the same because the Amazon Tools for PowerShell calls the same API operations that are used by the service console. For more information, see the "Identity and Access Management" section in the "Security" chapter for the Amazon service that you want to use.

The only major difference is how you authenticate when using a standard IAM user and long-term credentials. Although an IAM user requires a password to access an Amazon service's console, that same IAM user requires an access key instead of a password to perform the same operations using the Amazon Tools for PowerShell. All other short-term credentials are used in the same way they are used with the console.

The credentials used by the Amazon Tools for PowerShell are typically stored in plaintext files and are **not** encrypted. However, you do have an option to use the encrypted .NET SDK credential store when you run on Windows.

- The `$HOME/.aws/credentials` file stores long-term credentials required to access your Amazon resources. These include your access key ID and secret access key.

Mitigation of Risk

- We strongly recommend that you configure your file system permissions on the `$HOME/.aws` folder and its child folders and files to restrict access to only authorized users.
- Use roles with temporary credentials wherever possible to reduce the opportunity for damage if the credentials are compromised. Use long-term credentials only to request and refresh short-term role credentials.

Compliance Validation for the Amazon Tools for PowerShell

Third-party auditors assess the security and compliance of Amazon services as part of multiple Amazon compliance programs. Using the Amazon Tools for PowerShell to access a service does not alter that service's compliance.

For a list of Amazon services in scope of specific compliance programs, see [Amazon Services in Scope by Compliance Program](#). For general information, see [Amazon Compliance Programs](#).

You can download third-party audit reports using the Amazon Artifact. For more information, see [Downloading Reports in Amazon Artifact](#).

Your compliance responsibility when using Amazon Tools for PowerShell is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. Amazon 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 security- and compliance-focused baseline environments on Amazon.
- [Architecting for HIPAA Security and Compliance Whitepaper](#) – This whitepaper describes how companies can use Amazon to create HIPAA-compliant applications.
- [Amazon Compliance Resources](#) – This collection of workbooks and guides might apply to your industry and location.
- [Evaluating Resources with Rules](#) in the *Amazon Config Developer Guide* – The Amazon Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- [Amazon Security Hub](#) – This Amazon service provides a comprehensive view of your security state within Amazon that helps you check your compliance with security industry standards and best practices.

Document History

This topic describes significant changes to the documentation for the Amazon 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 Amazon Tools for PowerShell, see the [release notes](#).

Change	Description	Date
Launch an Amazon EC2 Instance Using Windows PowerShell (p. 65)	Added notes about retiring EC2-Classic.	July 26, 2022
Amazon Tools for PowerShell Version 4 (p. 77)	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.	November 21, 2019
Amazon Tools for PowerShell 3.3.563 (p. 77)	Added information about how to install and use the preview version of the <code>AWS.Tools.Common</code> module. This new module breaks apart the older monolithic package into one shared module and one module per Amazon service.	October 18, 2019
Amazon Tools for PowerShell 3.3.343.0 (p. 77)	Added information to the Using the Amazon Tools for PowerShell section introducing the Amazon Lambda Tools for PowerShell for PowerShell Core developers to build Amazon Lambda functions.	September 11, 2018
Amazon Tools for Windows PowerShell 3.1.31.0 (p. 77)	Added information to the Getting Started section about new cmdlets that use Security Assertion Markup Language (SAML) to support configuring federated identity for users.	December 1, 2015
Amazon Tools for Windows PowerShell 2.3.19 (p. 77)	Added information to the Cmdlets Discovery and Aliases section about the new <code>Get-AWSCmdletName</code> cmdlet that can help users more easily find their desired Amazon cmdlets.	February 5, 2015

[Amazon Tools for Windows PowerShell 1.1.1.0 \(p. 77\)](#)

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 pipelining. 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.

May 15, 2013

[Amazon Tools for Windows PowerShell 1.0.1.0 \(p. 77\)](#)

December 21, 2012

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-Module` of the `AWSPowerShell.psd1` 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

[Amazon Tools for Windows PowerShell 1.0.0.0 \(p. 77\)](#)

with an execution policy that disallows script execution.

Initial release

December 6, 2012