AWS Tools for Windows PowerShell

User Guide

Table of Contents

What are the AWS Tools for PowerShell?

The AWS Tools for Windows PowerShell and AWS Tools for PowerShell Core are PowerShell modules that are built on the functionality exposed by the AWS SDK for .NET. The AWS PowerShell Tools enable you to script operations on your AWS resources from the PowerShell command line. Although the cmdlets are implemented using the service clients and methods from the SDK, the cmdlets provide an idiomatic PowerShell experience for specifying parameters and handling results. For example, the cmdlets for the Tools for Windows PowerShell support PowerShell pipelining—that is, you can pipeline PowerShell objects both into and out of the cmdlets.

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

The AWS Tools for Windows PowerShell support the same set of services and regions that are supported by the SDK.

How to Use this Guide

The guide is divided into the following major sections:

Setting up the AWS Tools for Windows PowerShell or AWS Tools for PowerShell Core (p. 2)

This section explains how to install the AWS Tools for Windows PowerShell and AWS Tools for PowerShell Core. It also covers how to sign up for AWS if you don't already have an account. (An AWS account is required in order to use the Tools for Windows PowerShell.)

Getting Started with the AWS Tools for Windows PowerShell (p. 16)

This section describes the fundamentals of using the tools, such as specifying credentials and regions, finding cmdlets for a particular service, and using aliases for cmdlets.

Using the AWS Tools for Windows PowerShell (p. 40)

This section includes information about using the AWS Tools for Windows PowerShell and AWS Tools for PowerShell Core to perform common AWS tasks.

Setting up the AWS Tools for Windows PowerShell or AWS Tools for PowerShell Core

Topics

Prerequisites

To use the AWS Tools for Windows PowerShell or the AWS Tools for PowerShell Core, you must have an AWS account. If you do not yet have an AWS account, see AWS Account and Access Keys (p. 14) for instructions.

Setting up the AWS Tools for PowerShell on a Windows-based Computer

Overview of Setup

A Windows-based computer can run the AWS Tools for PowerShell, the AWS Tools for PowerShell Core, or both, depending on the release and edition of Windows that you are running. Setting up the AWS Tools for PowerShell or AWS Tools for PowerShell Core involves the following tasks, described in this topic.

1. Installing Windows PowerShell 2.0 or newer (Microsoft PowerShell Core 6.0 or newer if you are installing the AWS Tools for PowerShell Core).
2. After installing PowerShell, either downloading and running the AWS Tools for Windows PowerShell MSI installer, or starting PowerShell.
3. If you did not run the MSI installer, running `Install-Module` in a PowerShell session to install the AWS Tools for PowerShell or PowerShell Core.
4. Verifying that script execution is enabled by running the `Get-ExecutionPolicy` cmdlet.
5. If you are not running the custom AWS Tools for PowerShell console, or running Windows PowerShell 3.0 or newer, explicitly loading the AWS Tools for PowerShell module into your PowerShell session by running an `Import-Module AWSPowerShell` command.

Prerequisites

To use the AWS Tools for Windows PowerShell or the AWS Tools for PowerShell Core, you must have an AWS account. If you do not yet have an AWS account, see AWS Account and Access Keys (p. 14) for instructions.

To use the AWS Tools for Windows PowerShell, your system must meet the following prerequisites.

- Microsoft Windows XP or later
- Windows PowerShell 2.0 or later (PowerShell Core 6.0 or later for the Tools for PowerShell Core).

The following table shows the versions of PowerShell that are installed on Windows releases by default.

Windows Release	Included PowerShell Release
Windows 7 and Windows Server 2008 R2	Windows PowerShell 2.0
Windows 8 and Windows Server 2012	Windows PowerShell 3.0
Windows 8.1 and Windows Server 2012 R2	Windows PowerShell 4.0
Windows 10 and Windows Server 2016	Windows PowerShell 5.0
Windows 10 and Windows Server 2016 January 2017 Anniversary Update	Windows PowerShell 5.1

Server Core installation options for the preceding server releases include PowerShell. The Nano Server installation option of Windows Server 2016 includes PowerShell Core. For earlier releases of Windows, such as Windows XP, Windows Vista, Windows Server 2003, and Windows Server 2008, you can get PowerShell 2.0 by installing the Windows Management Framework.

- Windows Management Framework (Windows PowerShell 2.0, WinRM 2.0, and BITS 4.0)

Install the AWS Tools for PowerShell on a Windows-based Computer

To upgrade to a newer release of the AWS Tools for PowerShell, follow instructions in pstools-updating (p. 9). Uninstall older versions of PowerShell first.

The AWS Tools for Windows PowerShell is one of the optional components that you can install by running the AWS Tools for Windows installer `.msi`. Download the installer by opening the following webpage, and then choosing **AWS Tools for Windows**.

- http://aws.amazon.com/powershell/

The installer for the Tools for Windows PowerShell installs the most recent versions of the AWS SDK for .NET assemblies for the .NET 3.5 and 4.5 Frameworks. If you have Microsoft Visual Studio 2013 or 2015 installed, the installer can also install the AWS Toolkit for Visual Studio.

Users who are running PowerShell 5.0 or newer can also install and update the Tools for Windows PowerShell from Microsoft's PowerShell Gallery website by running the following command.

```
PS C:\> Install-Module -Name AWSPowerShell
```

If you are running PowerShell 3.0 or newer, and add the module installation path to the value of the `PSModulePath` environment variable, the Tools for Windows PowerShell are automatically loaded into your session when you run any Tools for Windows PowerShell cmdlet or function.

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

Install the AWS Tools for PowerShell Core on a Windows-based Computer

You can install the AWS Tools for PowerShell Core on computers that are running Microsoft PowerShell Core 6.0 or newer. AWS Tools for PowerShell Core is supported on the following Windows-based operating systems.

- Windows 8.1 Enterprise
- Windows Server 2012 R2
- Windows 10 for Business or Windows 10 Pro
- Windows Server 2016

For more information about how to install PowerShell Core on computers that run Windows 8.1 or Windows 10, see Package installation instructions (Windows), also in the GitHub repository for PowerShell.

After you install PowerShell Core, you can find the AWS Tools for PowerShell Core on Microsoft's PowerShell Gallery website. The simplest way to install the Tools for PowerShell Core is by running the `Install-Module` cmdlet.

```
PS C:\> Install-Module -Name AWSPowerShell.NetCore -AllowClobber
```

It is not necessary to run this command as Administrator, unless you want to install the AWS Tools for PowerShell Core for all users of a computer. To do this, run the following command in a PowerShell session that is running as Administrator:

```
PS C:\> Install-Module -Scope CurrentUser -Name AWSPowerShell.NetCore -Force
```

To install both AWSPowerShell and AWSPowerShell.NetCore on a Windows-based computer, add `-AllowClobber` to the second installation command, because the modules have cmdlets with the same names.

For more information about the release of AWS Tools for PowerShell Core, see the AWS blog post, Introducing AWS Tools for PowerShell Core Edition.

Installation Troubleshooting Tips

Some users have reported issues with the Install-Module cmdlet that is included with older releases of PowerShell Core, including errors related to semantic versioning (see https://github.com/OneGet/oneget/issues/202). Using the NuGet provider appears to resolve the issue. Newer versions of PowerShell Core have resolved this issue.

To install AWS Tools for PowerShell Core by using NuGet, run the following command. Specify an appropriate destination folder (on Linux, try -Destination ~/.local/share/powershell/Modules):

```
PS C:\> Install-Package -Name AWSPowerShell.NetCore -Source
https://www.powershellgallery.com/api/v2/ -ProviderName NuGet -ExcludeVersion
-Destination <path to destination folder>
```

Enable Script Execution

To load the AWS Tools for Windows PowerShell or AWS Tools for PowerShell Core modules, enable PowerShell script execution if you have not already done so. To enable script execution, run the `Set-`

`ExecutionPolicy` cmdlet to set a policy of `RemoteSigned`. By default, PowerShell script execution policy is set to `Restricted`. For more information about execution policies, see About Execution Policies on the Microsoft Technet website.

To enable script execution

1. Administrator rights are required to set the execution policy. If you are not logged on as a user with administrator rights, open a PowerShell session as Administrator by doing the following: Click **Start** and then click **All Programs**. Click **Accessories**, and then click **Windows PowerShell**. Right-click **Windows PowerShell**, and then choose **Run as administrator** from the context menu.
2. At the command prompt, type: `Set-ExecutionPolicy RemoteSigned`

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

If you do not have the execution policy set correctly, PowerShell shows the following error.

```
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 updates the PSModulePath to include the location of the directory that contains the AWSPowerShell module. If you are running PowerShell 3.0 or newer, the AWSPowerShell module is loaded automatically whenever you run one of the AWS cmdlets. This lets you use the AWS cmdlets even if the execution policy on your system is set to disallow script execution.

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

```
PS C:\> Get-Module -ListAvailable

ModuleType Name                      ExportedCommands
---------- ----                      ----------------
Manifest   AppLocker                 {}
Manifest   BitsTransfer              {}
Manifest   PSDiagnostics             {}
Manifest   TroubleshootingPack       {}
Manifest   AWSPowerShell             {Update-EBApplicationVersion, Set-DPStatus, Remove-
IAMGroupPol...
```

Configure a PowerShell Console to Use the AWS Tools for Windows PowerShell

The installer creates a **Start Menu** group called **Amazon Web Services**, which contains a shortcut called **Windows PowerShell for AWS**. In PowerShell 2.0, this shortcut automatically imports the AWSPowerShell module and runs the `Initialize-AWSDefaultConfiguration` cmdlet for you. Because PowerShell 3.0 and newer automatically load the AWSPowerShell module whenever you run an AWS cmdlet, in PowerShell 3.0 and newer, the shortcut created by the AWS Tools for PowerShell installer runs only the `Initialize-AWSDefaultConfiguration` cmdlet. For more information

about `Initialize-AWSDefaultConfiguration`, see Using AWS Credentials (p. 16). In older (before 3.3.96.0) releases of the Tools for Windows PowerShell, this cmdlet was named `Initialize-AWSDefaults`.

The installer creates another shortcut titled **AWS Tools for Windows**, which opens a visual display of AWS resources for Windows developers.

If you run PowerShell 3.0 or newer, or if you only use the custom-console shortcut that is installed by the installer, there is no need to configure a PowerShell window to use the AWS Tools for Windows PowerShell. But if you run PowerShell 2.0 with a specially-configured PowerShell console, and you want to add support for the AWS Tools for PowerShell, you must load the AWS module manually by running `Import-Module` as described in the following sections.

How to Load the AWS Tools for Windows PowerShell Module (PowerShell 2.0)

To load the Powershell Tools module into your current session

1. Open a PowerShell session, type the following command, and press Enter.

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

   Note
   In PowerShell 4.0 and later, Import-Module also searches the Program Files folder for installed modules, so it is not necessary to provide the full path to the module. You can run the following command to import the AWSPowerShell module. In PowerShell 3.0 and later, running a cmdlet in the module also automatically imports a module into your session.

```
PS C:\> Import-Module AWSPowerShell
```

2. To verify that the module was loaded, type the following command:

```
PS C:\> Get-Module
```

   Look for an entry in the list named **AWSPowerShell** to verify that the Tools for Windows PowerShell module was loaded successfully.

```
ModuleType Version    Name            ExportedCommands
---------- -------    ----            ----------------
Binary     3.3.96.0   AWSPowerShell   {Add-AASScalableTarget, Add-ACMCertificateTag, Add-
ADSConfigurationItemsToApplication, Add-ASAAttachmentsToSet...}
...
```

Load the AWS Tools for Windows PowerShell Module into Every Session (PowerShell 2.0)

To load the AWSPowerShell module automatically every time you start a PowerShell session, add it to your PowerShell profile. Note, however, that adding commands to your PowerShell profile can slow the startup of PowerShell.

The PowerShell `$profile` variable stores the full path to the text file containing your PowerShell profile. This variable is available only in a PowerShell session; it is not a Windows environment variable. To view the value of this variable, run `echo`.

```
echo $profile C:\Users\{username}\Documents\WindowsPowerShell
\Microsoft.PowerShell_profile.ps1
```

You can edit this file with any text editor, such as notepad.exe.

```
notepad $profile
```

You might need to create both the profile directory and the profile itself, if they do not already exist.

Versioning

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

```
PS C:\> Get-AWSPowerShellVersion

AWS Tools for Windows PowerShell
Version 3.3.96.0
Copyright 2012-2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.

Amazon Web Services SDK for .NET
Core Runtime Version 3.3.14.0
Copyright 2009-2015 Amazon.com, Inc. or its affiliates. All Rights Reserved.

Release notes: https://aws.amazon.com/releasenotes/PowerShell

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 which AWS services are supported in the current version of the tools.

```
PS C:\> Get-AWSPowerShellVersion -ListServiceVersionInfo

AWS Tools for Windows PowerShell
Version 3.3.96.0
Copyright 2012-2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.

Amazon Web Services SDK for .NET
Core Runtime Version 3.3.14.0
Copyright 2009-2015 Amazon.com, Inc. or its affiliates. All Rights Reserved.

Release notes: https://aws.amazon.com/releasenotes/PowerShell

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


Service                          Noun Prefix Version
-------                          ----------- -------
AWS AppStream                     APS         2016-12-01
AWS Batch                         BAT         2016-08-10
AWS Budgets                       BGT         2016-10-20
AWS Certificate Manager           ACM         2015-12-08
AWS Cloud Directory               CDIR        2016-05-10
AWS Cloud HSM                     HSM         2014-05-30
AWS CloudFormation                CFN         2010-05-15
```

```
AWS CloudTrail                      CT          2013-11-01
AWS CodeBuild                       CB          2016-10-06
AWS CodeCommit                      CC          2015-04-13
AWS CodeDeploy                      CD          2014-10-06
AWS CodePipeline                    CP          2015-07-09
AWS CodeStar                        CST         2017-04-19
AWS Config                          CFG         2014-11-12
AWS Cost and Usage Report           CUR         2017-01-06
AWS Data Pipeline                   DP          2012-10-29
AWS Database Migration Service      DMS         2016-01-01
AWS Device Farm                     DF          2015-06-23
AWS Direct Connect                  DC          2012-10-25
AWS Directory Service               DS          2015-04-16
AWS Elastic Beanstalk               EB          2010-12-01
AWS Health                          HLTH        2016-08-04
AWS Identity and Access Management  IAM         2010-05-08
AWS Import/Export                   IE          2010-06-01
AWS Import/Export Snowball          SNOW        2016-06-30
AWS IoT                             IOT         2015-05-28
AWS Key Management Service          KMS         2014-11-01
AWS Marketplace Commerce Analytics  MCA         2015-07-01
AWS Marketplace Entitlement Service MES         2017-01-11
AWS Marketplace Metering            MM          2016-01-14
AWS OpsWorks                        OPS         2013-02-18
AWS OpsWorksCM                      OWCM        2016-11-01
AWS Organizations                   ORG         2016-11-28
AWS Resource Groups Tagging API     RGT         2017-01-26
AWS Security Token Service          STS         2011-06-15
AWS Service Catalog                 SC          2015-12-10
AWS Shield                          SHLD        2016-06-02
AWS Storage Gateway                 SG          2013-06-30
AWS Support API                     ASA         2013-04-15
AWS WAF                             WAF         2015-08-24
AWS WAF Regional                    WAFR        2016-11-28
AWS X-Ray                           XR          2016-04-12
Amazon API Gateway                  AG          2015-07-09
Amazon Athena                       ATH         2017-05-18
Amazon CloudFront                   CF          2017-03-25
Amazon CloudSearch                  CS          2013-01-01
Amazon CloudSearchDomain            CSD         2013-01-01
Amazon CloudWatch                   CW          2010-08-01
Amazon CloudWatch Events            CWE         2015-10-07
Amazon CloudWatch Logs              CWL         2014-03-28
Amazon Cognito Identity             CGI         2014-06-30
Amazon Cognito Identity Provider    CGIP        2016-04-18
Amazon DynamoDB                     DDB         2012-08-10
Amazon EC2 Container Registry       ECR         2015-09-21
Amazon EC2 Container Service        ECS         2014-11-13
Amazon ElastiCache                  EC          2015-02-02
Amazon Elastic Compute Cloud        EC2         2016-11-15
Amazon Elastic File System          EFS         2015-02-01
Amazon Elastic MapReduce            EMR         2009-03-31
Amazon Elastic Transcoder           ETS         2012-09-25
Amazon Elasticsearch                ES          2015-01-01
Amazon GameLift Service             GML         2015-10-01
Amazon Inspector                    INS         2016-02-16
Amazon Kinesis                      KIN         2013-12-02
Amazon Kinesis Analytics            KINA        2015-08-14
Amazon Kinesis Firehose             KINF        2015-08-04
Amazon Lambda                       LM          2015-03-31
Amazon Lex                          LEX         2016-11-28
Amazon Lex Model Building Service   LMB         2017-04-19
Amazon Lightsail                    LS          2016-11-28
Amazon MTurk Service                MTR         2017-01-17
Amazon Machine Learning             ML          2014-12-12
Amazon Pinpoint                     PIN         2016-12-01
```

```
Amazon Polly                          POL         2016-06-10
Amazon Redshift                       RS          2012-12-01
Amazon Rekognition                    REK         2016-06-27
Amazon Relational Database Service    RDS         2014-10-31
Amazon Route 53                       R53         2013-04-01
Amazon Route 53 Domains               R53D        2014-05-15
Amazon Server Migration Service       SMS         2016-10-24
Amazon Simple Email Service           SES         2010-12-01
Amazon Simple Notification Service    SNS         2010-03-31
Amazon Simple Queue Service           SQS         2012-11-05
Amazon Simple Storage Service         S3          2006-03-01
Amazon Simple Systems Management      SSM         2014-11-06
Amazon Step Functions                 SFN         2016-11-23
Amazon WorkDocs                       WD          2016-05-01
Amazon WorkSpaces                     WKS         2015-04-08
Application Auto Scaling              AAS         2016-02-06
Application Discovery Service         ADS         2015-11-01
Auto Scaling                          AS          2011-01-01
Elastic Load Balancing                ELB         2012-06-01
Elastic Load Balancing V2             ELB2        2015-12-01
```

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

```
PS C:\> $PSVersionTable

Name                           Value
----                           -----
PSVersion                      5.0.10586.117
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
BuildVersion                   10.0.10586.117
CLRVersion                     4.0.30319.34209
WSManStackVersion              3.0
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1
```

Updating the AWS Tools for Windows PowerShell and AWS Tools for PowerShell Core

Periodically, as updated versions of the Tools for Windows PowerShell or Tools for PowerShell Core are released, you should update the version that you are running locally. 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 at AWS Tools for Windows PowerShell or on the PowerShell Gallery website. A suggested time period for checking for an updated AWS Tools for PowerShell package is every two to three weeks.

Update the Tools for Windows PowerShell

Update your installed Tools for Windows PowerShell by downloading the most recent version of the MSI package from AWS Tools for Windows PowerShell and comparing the package version number in the MSI file name with the version number you get when you run the `Get-AWSPowerShellVersion` cmdlet.

If the download version is a higher number than the version you have installed, close all Tools for Windows PowerShell consoles, then uninstall **AWS Tools for Windows** by selecting it in the **Control Panel | Programs and Features | Uninstall a program** dialog box, and then clicking **Uninstall**. Wait for uninstallation to finish.

If you installed the existing version of the AWS Tools for PowerShell by running `Install-Module`, you can uninstall the existing version by running `Uninstall-Module`.

Install the newer version of the Tools for Windows PowerShell by running the MSI package you downloaded.

Update the Tools for PowerShell Core

Before you install a newer release of the AWS Tools for PowerShell Core, uninstall the existing module. Close any open PowerShell or AWS Tools for PowerShell sessions before you uninstall the existing Tools for PowerShell Core package. Run the following command to uninstall the package.

```
PS C:\> Uninstall-Module -Name AWSPowerShell.NetCore -AllVersions
```

When uninstallation is finished, install the updated module by running the following command. By default, this command installs the latest version of the AWS Tools for PowerShell Core. This module is available on the PowerShell Gallery, but the easiest method of installation is to run `Install-Module`.

```
PS C:\> Install-Module -Name AWSPowerShell.NetCore
```

See Also

- Getting Started with the AWS Tools for Windows PowerShell (p. 16)
- Using the AWS Tools for Windows PowerShell (p. 40)
- AWS Account and Access Keys (p. 14)

Setting up the AWS Tools for PowerShell Core on Linux or macOS X

Overview of Setup

Note
You can skip AWS Tools for PowerShell Core installation on the .NET Core with Ubuntu Server 16.04 and .NET Core with Amazon Linux 2 LTS Candidate Amazon Machine Images (AMIs). These AMIs are preconfigured with .NET Core 2.0, PowerShell Core 6.0, the AWS Tools for PowerShell Core, and the AWS CLI.

A non-Windows-based computer can run only the AWS Tools for PowerShell Core (AWSPowerShell.NetCore). Setting up the AWS Tools for PowerShell Core involves the following tasks, described in this topic.

1. Installing Microsoft PowerShell Core 6.0 or newer on a supported non-Windows system.
2. After installing Microsoft PowerShell Core, starting PowerShell by running `pwsh` in your system shell.
3. Installing the AWS Tools for PowerShell Core.
4. Running the Initialize-AWSDefaultConfiguration cmdlet to provide your AWS credentials.

Prerequisites

To use the the AWS Tools for PowerShell Core, you must have an AWS account. If you do not yet have an AWS account, see AWS Account and Access Keys (p. 14) for instructions.

To run the AWS Tools for PowerShell Core, your system must be running Microsoft PowerShell Core 6.0 or newer. For more information about how to install PowerShell Core 6.0 or newer on a Linux-based computer, see PowerShell Package Installation Instructions.

Install the AWS Tools for PowerShell Core on Linux, macOS X, and Other Non-Windows Systems

To upgrade to a newer release of the AWS Tools for PowerShell Core, follow instructions in pstools-updating-core (p. 14). Uninstall older versions of PowerShell first.

You can install the AWS Tools for PowerShell Core on computers that are running Microsoft PowerShell Core 6.0 or newer. Microsoft PowerShell Core 6.0 is supported on the following non-Windows-based operating systems.

- Ubuntu 14.04 LTS and newer
- CentOS Linux 7
- Arch Linux
- Debian 8.7 and newer
- Red Hat Enterprise Linux 7
- OpenSUSE 42.2
- Fedora 25 and 26
- macOS 10.12

Some Linux-based operating systems, such as Arch and Kali, are not officially supported, but have community support. For more information about how to install PowerShell Core on computers that do not run Windows, see Package installation instructions (Linux) in the GitHub repository for the Microsoft PowerShell project.

After you install PowerShell Core, you can find the AWS Tools for PowerShell Core on Microsoft's PowerShell Gallery website. The simplest way to install the AWS Tools for PowerShell Core is by running the `Install-Module` cmdlet. First, start your PowerShell session by running `pwsh` in a shell.

> **Note**
> Although you can start PowerShell by running `sudo pwsh` to run PowerShell with elevated rights, be aware that this is a potential security risk, and not consistent with the principle of least privilege.

Next, run `Install-Module` as shown in the following command.

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

It is not necessary to run this command as Administrator, unless you want to install the AWS Tools for PowerShell Core 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 CurrentUser -Name AWSPowerShell.NetCore -Force
```

For more information about the release of AWS Tools for PowerShell Core, see the AWS blog post, Introducing AWS Tools for PowerShell Core Edition.

Installation Troubleshooting Tips

Some users have reported issues with the Install-Module cmdlet that is included with older releases of PowerShell Core, including errors related to semantic versioning (see https://github.com/OneGet/oneget/issues/202). Using the NuGet provider appears to resolve the issue. Newer versions of PowerShell Core have resolved this issue.

To install AWS Tools for PowerShell Core by using NuGet, run the following command. Specify an appropriate destination folder (on Linux, try `-Destination ~/.local/share/powershell/Modules`).

```
PS> Install-Package -Name AWSPowerShell.NetCore -Source
https://www.powershellgallery.com/api/v2/ -ProviderName NuGet -ExcludeVersion
-Destination <path to destination folder>
```

Script Execution

The `Set-ExecutionPolicy` command is not available in PowerShell Core running 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 about execution policies, see About Execution Policies on the Microsoft Technet website.

The AWS Tools installer updates the PSModulePath to include the location of the directory that contains the AWSPowerShell module.

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

```
PS> Get-Module -ListAvailable

Directory: /home/ubuntu/.local/share/powershell/Modules

ModuleType Version    Name                                ExportedCommands
---------- -------    ----                                ----------------
Binary     3.3.219.0  AWSPowerShell.NetCore               {Add-AASScalableTarget, Add-
ACMCertificateTag, Add-ADSC...
```

Configure a PowerShell Console to Use the AWS Tools for PowerShell Core

Because PowerShell 3.0 and newer automatically load the AWSPowerShell module whenever you run an AWS cmdlet, and AWSPowerShell.NetCore requires at least PowerShell 6.0, there is no need to configure PowerShell to use the AWS PowerShell Tools. When you start PowerShell on a Linux-based system after you have installed the AWS Tools for PowerShell Core, run Initialize-AWSDefaultConfiguration to specify your AWS access and secret keys. For more information about `Initialize-AWSDefaultConfiguration`, see Using AWS Credentials (p. 16). In older (before 3.3.96.0) releases of the AWS Tools for PowerShell, this cmdlet was named `Initialize-AWSDefaults`.

Versioning

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

```
PS> Get-AWSPowerShellVersion

AWS Tools for PowerShell Core
Version 3.3.219.0
Copyright 2012-2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.

Amazon Web Services SDK for .NET
Core Runtime Version 3.3.21.6
```

```
Copyright 2009-2015 Amazon.com, Inc. or its affiliates. All Rights Reserved.

Release notes: https://aws.amazon.com/releasenotes/PowerShell

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 which AWS services are supported in the current version of the tools.

```
PS> Get-AWSPowerShellVersion -ListServiceVersionInfo

AWS Tools for PowerShell Core
Version 3.3.219.0
Copyright 2012-2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.

Amazon Web Services SDK for .NET
Core Runtime Version 3.3.21.6
Copyright 2009-2015 Amazon.com, Inc. or its affiliates. All Rights Reserved.

Release notes: https://aws.amazon.com/releasenotes/PowerShell

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


Service                                  Noun Prefix API Version
-------                                  ----------- -----------
AWS AppStream                            APS         2016-12-01
AWS AppSync                              ASYN        2017-07-25
AWS Batch                                BAT         2016-08-10
AWS Budgets                              BGT         2016-10-20
AWS Certificate Manager                  ACM         2015-12-08
AWS Cloud Directory                      CDIR        2016-05-10
AWS Cloud HSM                            HSM         2014-05-30
AWS Cloud HSM V2                         HSM2        2017-04-28
AWS Cloud9                               C9          2017-09-23
AWS CloudFormation                       CFN         2010-05-15
AWS CloudTrail                           CT          2013-11-01
AWS CodeBuild                            CB          2016-10-06
AWS CodeCommit                           CC          2015-04-13
AWS CodeDeploy                           CD          2014-10-06
AWS CodePipeline                         CP          2015-07-09
AWS CodeStar                             CST         2017-04-19
AWS Config                               CFG         2014-11-12
AWS Cost Explorer                        CE          2017-10-25
AWS Cost and Usage Report                CUR         2017-01-06
AWS Data Pipeline                        DP          2012-10-29
AWS Database Migration Service           DMS         2016-01-01
AWS Device Farm                          DF          2015-06-23
AWS Direct Connect                       DC          2012-10-25
AWS Directory Service                    DS          2015-04-16
AWS Elastic Beanstalk                    EB          2010-12-01
AWS Elemental MediaConvert               EMC         2017-08-29
AWS Elemental MediaLive                  EML         2017-10-14
AWS Elemental MediaPackage               EMP         2017-10-12
AWS Elemental MediaStore                 EMS         2017-09-01
AWS Elemental MediaStore Data Plane      EMSD        2017-09-01
AWS Greengrass                           GG          2017-06-07
AWS Health                               HLTH        2016-08-04
AWS Identity and Access Management       IAM         2010-05-08
...
```

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                      6.0.0
PSEdition                      Core
GitCommitId                    v6.0.0
OS                             Linux 4.4.0-1047-aws #56-Ubuntu SMP Sat Jan 6 19:39:06 UTC
 2018
Platform                       Unix
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1
WSManStackVersion              3.0
```

Updating the AWS Tools for Windows PowerShell and AWS Tools for PowerShell Core

Periodically, as updated versions of the AWS Tools for PowerShell Core are released, you should update the version that you are running locally. Run the `Get-AWSPowerShellVersion` cmdlet to determine the version that you are running, and compare that with the version of AWS Tools for PowerShell Core that is available at AWS Tools for Windows PowerShell or on the PowerShell Gallery website. A suggested time period for checking for an updated AWS Tools for PowerShell package is every two to three weeks.

Update the Tools for PowerShell Core (All systems)

Before you install a newer release of the AWS Tools for PowerShell Core, close any open PowerShell or AWS Tools for PowerShell Core sessions before you uninstall the existing Tools for PowerShell Core package. You can exit a PowerShell session on a Linux-based system by pressing **Ctrl+D**. Run the following command to uninstall the package.

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

When uninstallation is finished, install the updated module by running the following command. By default, this command installs the latest version of the AWS Tools for PowerShell Core. This module is available on the PowerShell Gallery, but the easiest method of installation is to run `Install-Module`.

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

See Also

- Getting Started with the AWS Tools for Windows PowerShell (p. 16)
- Using the AWS Tools for Windows PowerShell (p. 40)
- AWS Account and Access Keys (p. 14)

AWS Account and Access Keys

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

Access keys consist of an *access key ID* and *secret access key*, which are used to sign programmatic requests that you make to AWS. If you don't have access keys, you can create them by using the AWS Management Console. We recommend that you use IAM access keys instead of AWS root account access keys. IAM lets you securely control access to AWS services and resources in your AWS 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.
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 keys

in a secure location.

> **Important**
> Keep the keys confidential to protect your
> <problematic>|AWS|</problematic>
> account, and never email them. Do not share them outside your organization, even if an inquiry appears to come from
> <problematic>|AWS|</problematic>
> or Amazon.com. *No one who legitimately represents Amazon will ever ask you for your secret key.*

Related topics

- What Is IAM? in *IAM User Guide*.
- AWS Security Credentials in *Amazon Web Services General Reference*.

Getting Started with the AWS 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 region the Tools for Windows PowerShell should use when interacting with AWS. This section also provides guidance for using standard PowerShell cmdlets such as `Get-Command` to discover AWS cmdlets.

Topics

Using AWS Credentials

Each Tools for Windows PowerShell command must include a set of AWS credentials, which are used to cryptographically sign the corresponding web service request. You can specify credentials per-command, per-session, or for all sessions. 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 credentials stores. Specify the correct profile by name in your command, and the Tools for Windows PowerShell retrieve the associated credentials. For a general discussion of how to safely manage AWS credentials, see Best Practices for Managing AWS Access Keys.

> **Note**
> If you do not yet have an AWS account, you will need one in order to obtain credentials and use the Tools for Windows PowerShell. For information about how to sign up for an account, see AWS Account and Access Keys (p. 14).

Topics

Managing Profiles

The Tools for Windows PowerShell can use either of two credentials stores.

- The AWS SDK store, which encrypts your credentials and stores them in your home folder.

  The AWS SDK for .NET and Toolkit for Visual Studio can also use the AWS SDK store.

- The credentials file, which is also located in your home folder, but stores credentials as plain text.

  By default, the credentials file is stored here: `C:\Users\username\.aws\credentials`. The AWS SDKs and the AWS Command Line Interface can also use the credentials file. If you are running a script outside of your AWS user context, be sure that the file that contains your credentials is copied to a location where all user accounts (local system and user) can access your credentials.

This topic describes how to use the Tools for Windows PowerShell to manage your profiles in the AWS SDK store. You can also manage the AWS SDK store by using the Toolkit for Visual Studio or programmatically by using the AWS SDK for .NET. For directions about how to manage profiles in the credentials file, see Best Practices for Managing AWS Access Keys.

Add a new profile

To add a new profile to the AWS SDK store, run `Set-AWSCredential`. Your access key and secret key are stored in your default credentials file.

```
PS C:\> Set-AWSCredential -AccessKey AKIAIOSFODNN7EXAMPLE -SecretKey wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY -StoreAs MyProfileName
```

- `-AccessKey`– The access key.
- `-SecretKey`– The secret key.
- `-StoreAs`– The profile name, which must be unique.

  To specify the default profile, set the profile name to `default`.

Update a profile

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

```
The AWS 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 as follows:

```
PS C:\> Get-AWSCredential -ListProfileDetail
```

Remove a profile

To remove a profile, use the following command:

```
PS C:\> Remove-AWSCredentialProfile -ProfileName MyProfileName
```

The `-ProfileName` parameter specifies the profile name.

You can continue to use Clear-AWSCredential for backward compatibility, but `Remove-AWSCredentialProfile` is preferred.

Specifying Credentials

There are several ways to specify credentials. The preferred approach is to use a profile rather than incorporating literal credentials into your command line. The Tools for Windows PowerShell locates the profile using a search order that is described in Credentials Search Order (p. 19). This section describes the most common ways to specify a profile.

AWS credentials are encrypted with the logged-on Windows user identity; they cannot be decrypted by using another account, or used on a different device from the one on which they were originally created. To perform tasks in the context of another user, such as a user account under which a scheduled task will run, set up an encrypted credential profile, as described in the preceding section, that you can use when you log on to the computer as that user. Log on as the task-performing user to complete the credential setup steps, create a profile that will work for that user, and then log off and log on again by using 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 Tools for Windows PowerShell releases. For backward compatibility, `-StoredCredentials` is still supported.

Default profile (recommended)

Use `Initialize-AWSDefaultConfiguration` to specify a default profile for every PowerShell session.

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

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

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.

```
PS C:\> Set-AWSCredential -ProfileName MyProfileName
```

Note
In versions of the Tools for Windows PowerShell that are older than 1.1, the `Set-AWSCredential` command 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

Add the `-ProfileName` parameter to specify a profile for a particular command. This profile overrides any default or session profiles. For example:

```
PS C:\> Get-EC2Instance -ProfileName MyProfileName
```

Note
When you specify a default or session profile, you can also add a `-Region` parameter to specify a default or session region. For more information, see Specifying AWS Regions (p. 23). The following example specifies a default profile and region.

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

By default, the credentials file is assumed to be in the user's home folder (`C:\Users\username\.aws`). To specify a credentials file in another location, include a `-ProfileLocation` parameter, set to the credentials file path. The following example specifies a non-default credentials file for a specific command.

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

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

Credentials Search Order

When you run a command, the Tools for Windows PowerShell search for credentials in the following order, and uses the first available set.

1. Use literal credentials that are embedded in the command line.

   We strongly recommend using profiles rather than putting literal credentials in your command lines.
2. Use a specified profile name or profile location.
   - If you specify only a profile name, use a specified profile from the AWS SDK store and, if that does not exist, the specified profile from the credentials file in the default location.
   - If you specify only a profile location, use the `default` profile from that credentials file.
   - If you specify a name and a location, use the specified profile from 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 have not specified a profile or location.
3. Use credentials specified by the `-Credential` parameter.
4. Use a session profile.
5. Use a default profile, in the following order:
   a. The `default` profile in the AWS SDK store.
   b. The `default` profile in the credentials file.
   c. Use the `AWS PS Default` profile in the AWS SDK store.
6. If you are running the command on an Amazon EC2 instance that is configured for an IAM role, use EC2 instance credentials stored in an instance profile.

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

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

Credential Handling in AWS Tools for PowerShell Core

Cmdlets in AWS Tools for PowerShell Core accept AWS access and secret keys or the names of credential profiles when they run, similarly to the AWS Tools for Windows PowerShell. When they run on Windows, both modules have access to the AWS SDK for .NET credential store file (stored in the per-user `AppData\Local\AWSToolkit\RegisteredAccounts.json` file). This file stores your keys in encrypted format, and cannot be used on a different computer. It is the first file that the AWS Tools for PowerShell searches for a credential profile, and is also the file where the AWS Tools for PowerShell stores credential profiles. For more information about the AWS SDK for .NET credential store file, see Configuring AWS Credentials. The AWS Tools for PowerShell module does not currently support writing credentials to other files or locations.

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

The SDK credential store holds your credentials in encrypted form by using Windows cryptographic APIs. These APIs are not available on other platforms, so the AWS Tools for PowerShell Core module uses the `ini`-format shared credentials file exclusively, and supports writing new credential profiles to the shared credential file. This support is slated for a future release of the AWS Tools for Windows PowerShell.

The following examples 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 Mac OS X operating systems:

```
# Writes a new (or updates existing) profile with name "myProfileName"
# in the default shared credentials file ~/.aws/credentials

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

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

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

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

Set-AWSCredential -ProfileName myProfileName

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

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

Shared Credentials in AWS Tools for PowerShell

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

The new profile types and access to the 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 new profiles by adding the common parameter, `-ProfileName`.

Parameter Name	Description
ExternalId	The user-defined external ID to be used when assuming a role, if required by the role.
MfaSerial	The MFA serial number to be used when assuming a role, if required by the role.
RoleArn	The ARN of the role to assume for assume role credentials.
SourceProfile	The name of the source profile to be used by assume role credentials.

The `ProfilesLocation` Common Parameter

The behavior of the `ProfileLocation` common parameter also changes. 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 `(user's home directory)\.aws\credentials`.
The path to a file in the shared credential file format	Search only the specified file for a profile with the given name.

Using the Credential Profile Types

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

Credentials Type	Parameter Combination
Basic	-AccessKey and -SecretKey values
Session	-AccessKey, -SecretKey, -SessionToken
AssumeRole	-SourceProfile, -RoleArn (optionally, -ExternalId and -MfaSerial)

Setup

The following is an example showing how to set up a source profile for an assume_role_profile credential profile type. In the first line, you set up a source profile for an assume role profile. In the second line, you set up the assume role profile itself. The third line shows the credentials from the new profile.

```
PS C:\> Set-AWSCredential -StoreAs source_profile -AccessKey access_key -SecretKey
 secret_key
PS C:\> Set-AWSCredential -StoreAs assume_role_profile -SourceProfile source-profile -
RoleArn arn:aws:iam::999999999999:role/some-role
PS C:\> Get-AWSCredential -ProfileName assume_role_profile

SourceCredentials                   RoleArn                                 RoleSessionName
                         Options
-----------------                   -------                                 ---------------
                         -------
Amazon.Runtime.BasicAWSCredentials arn:aws:iam::999999999999:role/some-role aws-dotnet-sdk-
session-636238288466144357 Amazon.Runtime.AssumeRoleAWSCredentialsOptions
```

To use a new credential type with the Tools for Windows PowerShell service cmdlets, do the following.

1. First, store credentials in one of the credential files, either the .NET credential file, or the AWS CLI shared credential file.
2. Add the -ProfileName common parameter to a Tools for Windows PowerShell cmdlet to reference the credentials.

The following is an example showing how to configure `assume role` credentials with the Get-S3Bucket cmdlet.

```
PS C:\> Set-AWSCredential -StoreAs source_profile -AccessKey access_key -SecretKey
 secret_key
PS C:\> Set-AWSCredential -StoreAs assume_role_profile -SourceProfile source_profile -
RoleArn arn:aws:iam::999999999999:role/some-role
PS C:\> Get-S3Bucket -ProfileName assume_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
```

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. In the first line, run `Set-AWSCredential` to use the new `-ProfileLocation` write functionality to add access and secret keys to a profile named `basic_profile` by adding the `-ProfileName` parameter. In the second line, run the Get-Content cmdlet to display the contents of the credentials file.

```
PS C:\> Set-AWSCredential -ProfileLocation C:\Users\auser\.aws\credentials -ProfileName
 basic_profile -AccessKey access_key2 -SecretKey secret_key2
PS C:\> Get-Content C:\Users\auser\.aws\credentials

aws_access_key_id=access_key2
aws_secret_access_key=secret_key2
```

Showing Credential Profiles

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

```
PS C:\> Get-AWSCredential -ListProfileDetail

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

Removing Credential Profiles

To remove credential profiles, run the new Remove-AWSCredentialProfile cmdlet. You can continue to use Clear-AWSCredential for backward compatibility, but `Remove-AWSCredentialProfile` is preferred.

Important Notes

If you rely on Exception types or Exception messages from the three credentials cmdlets to control script flow, you might need to update existing scripts to account for the changes.

Only Initialize-AWSDefaultConfiguration, New-AWSCredential, and Set-AWSCredential have the four new parameters. A command such as `Get-S3Bucket -AccessKey access_key -SecretKey secret_key` will continue to work. However, `Get-S3Bucket -SourceProfile source_profile_name -RoleArn arn:aws:iam::999999999999:role/role_name` will not work because the `Get-S3Bucket` cmdlet does not support the `SourceProfile` or `RoleArn` parameters.

Specifying AWS Regions

There are two ways to specify the AWS region to use when running AWS CLI commands, the `-Region` common parameter, or the `Set-DefaultAWSRegion` command.

Most AWS cmdlets fail if you do not specify a region. The exceptions are cmdlets for Amazon S3 (p. 41), Amazon SES, and AWS Identity and Access Management (IAM) (p. 46).

In the absence of a specified region, Amazon S3 and Amazon SES use US East (N. Virginia).

Amazon SES and IAM are services that do not require a region to be specified.

To specify the region for a single AWS command

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

```
PS C:\> Get-EC2Image -Region us-west-1
```

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

From the PowerShell command prompt, type the following command:

```
PS C:\> Set-DefaultAWSRegion -Region us-west-1
```

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

To view the current default region for all AWS CLI commands

From the PowerShell command prompt, type the following command:

```
PS C:\> Get-DefaultAWSRegion

Region                Name                           IsShellDefault
----------            ----                           --------------
us-west-1             US West (N. California)         True
```

To clear the current default region for all AWS CLI commands

From the PowerShell command prompt, type the following command:

```
PS C:\> Clear-DefaultAWSRegion
```

To view a list of all available AWS regions

From the PowerShell command prompt, type the following command. Note that the third column identifies which region is the default for your current session.

```
PS C:\> Get-AWSRegion

Region         Name                      IsShellDefault
------         ----                      --------------
ap-northeast-1 Asia Pacific (Tokyo)      False
ap-northeast-2 Asia Pacific (Seoul)      False
ap-south-1     Asia Pacific (Mumbai)     False
ap-southeast-1 Asia Pacific (Singapore)  False
ap-southeast-2 Asia Pacific (Sydney)     False
ca-central-1   Canada (Central)          False
eu-central-1   EU Central (Frankfurt)    False
eu-west-1      EU West (Ireland)         False
eu-west-2      EU West (London)          False
eu-west-3      EU West (Paris)           False
sa-east-1      South America (Sao Paulo) False
us-east-1      US East (Virginia)        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 might not be returned in the results of the `Get-AWSRegion` cmdlet. An example is the Asia Pacific (Osaka) Region (ap-northeast-3). If you are not able to specify a region by adding the `-Region` parameter, try specifying the region in a custom endpoint instead, as shown in the next section.

Specifying a Custom or Nonstandard Endpoint

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

```
PS C:\> AWS-PowerShellCmdlet -EndpointUrl "custom endpoint URL" -Other -Parameters
```

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

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

Cmdlet Discovery and Aliases

This section discusses which services are supported by the AWS Tools for Windows PowerShell, the set of cmdlets provided by the Tools for Windows PowerShell in support of those services, and alternative names (aliases) for accessing those services.

Cmdlet Discovery

To get the cmdlet name that corresponds to an AWS service API name, use the AWS `Get-AWSCmdletName` cmdlet along with the `-ApiOperation` parameter and the AWS service API name. For example, to get all of the possible cmdlet names that correspond to any available `DescribeInstances` AWS service API:

```
PS C:\> Get-AWSCmdletName -ApiOperation DescribeInstances

CmdletName              ServiceOperation         ServiceName
 CmdletNounPrefix
----------              ----------------         -----------
 ----------------
Get-EC2Instance         DescribeInstances        Amazon Elastic Compute Cloud        EC2
Get-OPSInstance         DescribeInstances        AWS OpsWorks                        OPS
```

Note that you can omit the `-ApiOperation` parameter name in the preceding call, so the following is equivalent:

```
PS C:\> Get-AWSCmdletName DescribeInstances
```

If you know the names of both the desired AWS service API and the AWS service, add the `-Service` parameter along with either the cmdlet noun prefix or some portion of the AWS service name. (For

instance, the cmdlet noun prefix for Amazon EC2 is `EC2`.) For example, to get the cmdlet name that corresponds to the `DescribeInstances` API in the Amazon EC2 service, call one of the following:

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

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

Note that the parameter values in these calls are case-insensitive.

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

```
PS C:\> Get-AWSCmdletName -ApiOperation SecurityGroup -MatchWithRegex

CmdletName                              ServiceOperation                         ServiceName
                          CmdletNounPrefix
----------                              ----------------                         -----------
                          ----------------
Approve-ECCacheSecurityGroupIngress     AuthorizeCacheSecurityGroupIngress       Amazon
 ElastiCache                     EC
Get-ECCacheSecurityGroup                DescribeCacheSecurityGroups              Amazon
 ElastiCache                     EC
New-ECCacheSecurityGroup                CreateCacheSecurityGroup                 Amazon
 ElastiCache                     EC
Remove-ECCacheSecurityGroup             DeleteCacheSecurityGroup                 Amazon
 ElastiCache                     EC
Revoke-ECCacheSecurityGroupIngress      RevokeCacheSecurityGroupIngress          Amazon
 ElastiCache                     EC
Get-EC2SecurityGroup                    DescribeSecurityGroups                   Amazon
 Elastic Compute Cloud           EC2
Grant-EC2SecurityGroupEgress            AuthorizeSecurityGroupEgress             Amazon
 Elastic Compute Cloud           EC2
Grant-EC2SecurityGroupIngress           AuthorizeSecurityGroupIngress            Amazon
 Elastic Compute Cloud           EC2
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
Join-ELBSecurityGroupToLoadBalancer     ApplySecurityGroupsToLoadBalancer        Elastic
 Load Balancing                 ELB
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 desired AWS service but not the AWS service API, use the `-MatchWithRegex` parameter along with the `-Service` parameter to scope the search to a single service. For example, to get all of the possible cmdlet names that contain `SecurityGroup` in just the Amazon EC2 service:

```
PS C:\> Get-AWSCmdletName -ApiOperation SecurityGroup -MatchWithRegex -Service EC2

CmdletName                              ServiceOperation                        ServiceName
                          CmdletNounPrefix
----------                              ----------------                        -----------
                          ----------------
Get-EC2SecurityGroup                    DescribeSecurityGroups                  Amazon
 Elastic Compute Cloud           EC2
Grant-EC2SecurityGroupEgress            AuthorizeSecurityGroupEgress            Amazon
 Elastic Compute Cloud           EC2
Grant-EC2SecurityGroupIngress           AuthorizeSecurityGroupIngress           Amazon
 Elastic Compute Cloud           EC2
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
```

If you know the name of the AWS Command Line Interface (AWS CLI) command, use the `-AwsCliCommand` parameter along with the desired AWS CLI command call to get the corresponding cmdlet name. For example, to get the cmdlet name that corresponds to the `authorize-security-group-ingress`AWS CLI command call in the Amazon EC2 service:

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

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

Note that the Get-AWSCmdletName cmdlet needs only enough of the AWS CLI command to be able to identify the service and the AWS API. For example, you could omit the `aws` portion of `aws ec2 authorize-security-group-ingress`.

To get a list of all of the cmdlets that are provided by the Tools for Windows PowerShell, use the PowerShell `Get-Command` cmdlet, for example:

```
PS C:\> Get-Command -Module AWSPowerShell
```

The `Get-Command` cmdlet generates this list in alphabetical order. Therefore, the list of cmdlets is sorted by PowerShell verb rather than PowerShell noun.

The following script generates a list of the cmdlets sorted by the PowerShell nouns that correspond to the supported AWS services.

```
$services =
"AS",   # Auto Scaling
"ASA",  # AWS Support API
"CC",   # AWS CodeCommit
"CD",   # AWS CodeDeploy
"CF",   # Amazon CloudFront
"CFG",  # Amazon Config
"CFN",  # AWS CloudFormation
"CGI",  # Amazon Cognito Identity
"CP",   # Amazon CodePipeline
"CS",   # Amazon CloudSearch
"CSD",  # Amazon CloudSearchDomain
"CT",   # AWS CloudTrail
"CW",   # Amazon CloudWatch
"CWL",  # Amazon CloudWatch Logs
"DC",   # AWS Direct Connect
"DDB",  # Amazon DynamoDB
"DF",   # AWS Device Farm
"DP",   # AWS Data Pipeline
"DS",   # AWS Directory Service
"EB",   # AWS Elastic Beanstalk
"EC",   # Amazon ElastiCache
"EC2",  # Amazon Elastic Compute Cloud
"ECS",  # Amazon EC2 Container Service
"EFS",  # Amazon Elastic File System
"ELB",  # Amazon Elastic Load Balancing
"EMR",  # Amazon Elastic MapReduce
"ETS",  # Amazon Elastic Transcoder
"HSM",  # AWS Cloud HSM
"IAM",  # AWS Identity and Access Management
"IE",   # AWS Import/Export
"KIN",  # AWS Kinesis
"KMS",  # AWS Key Management Service
"LM",   # Amazon Lambda
"ML",   # Amazon Machine Learning
"OPS",  # AWS OpsWorks
"R53",  # AWS Route 53
"R53D", # AWS Route 53 Domains
"RDS",  # Amazon Relational Database Service
"RS",   # Amazon Redshift
"S3",   # Amazon Simple Storage Service
"SES",  # Amazon Simple Email Service
"SG",   # AWS Storage Gateway
"SNS",  # Amazon Simple Notification Service
"SQS",  # Amazon Simple Queue Service
"SSM",  # Amazon Simple Systems Management
"STS",  # AWS Security Token Service
"WKS",  # Amazon WorkSpaces

foreach ($s in $services)
{
  "-----"; Get-Command -Noun ${s}*
}
```

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

```
PS C:\> Get-Command -Module AWSPowerShell | Select-String region

Clear-DefaultAWSRegion
Get-AWSRegion
```

```
Get-DefaultAWSRegion
Get-EC2Region
Set-DefaultAWSRegion
```

You can also find cmdlets for a specific service by filtering for the service prefix of cmdlet nouns. Service prefixes are shown in quotation marks in the previous script example. The following example returns cmdlets that support the Amazon CloudWatch service.

```
PS C:\> Get-Command -Module AWSPowerShell -Noun CW*
```

Cmdlet Naming and Aliases

The cmdlets provided by the Tools for Windows PowerShell for a given service correspond approximately to the methods provided by the AWS SDK for that service. However, because of PowerShell's naming conventions, the name of a cmdlet may be somewhat different than the name of the corresponding method. For example, the `Get-EC2Instance` cmdlet performs a similar function to the Amazon EC2`DescribeInstances` method.

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

```
PS C:\> Get-S3Object -BucketName text-content -Key text-object

Key          : text-object.txt
BucketName   : text-content
LastModified : Mon, 27 Aug 2012 19:39:34 GMT
ETag         : "f738612c5e842b39819c6d8fc4eb5b9b"
Size         : 20622
Owner        : Amazon.S3.Model.Owner
StorageClass : STANDARD
```

To retrieve the object with the Tools for Windows PowerShell, use the `Read-S3Object` cmdlet.

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

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

Note
The cmdlet help for an AWS cmdlet provides the name of the AWS SDK API that corresponds to the cmdlet. For more information about the standard PowerShell verbs and their expected meanings, go to the Windows DevCenter.

All AWS cmdlets that use the Remove verb, and the `Stop-EC2Instance` cmdlet when used with the `-Terminate` switch, now prompt for confirmation before proceeding. To bypass confirmation, use the `-Force` switch.

The AWS cmdlets do not support the `-WhatIf` switch.

Aliases

The setup program for the Tools for Windows PowerShell installs an aliases file that contains aliases for many of the Tools for Windows PowerShell cmdlets. You may find these aliases to be more intuitive

than the cmdlet names. For example, aliases are provided that are prefixed with the service name—rather than a PowerShell verb—and followed by an AWS SDK method name. An example is the `EC2-DescribeInstances` alias.

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

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

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

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

Pipelining and $AWSHistory

For service calls that return collections, the objects within the collection are now always 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

Return a collection of Amazon EC2 machine images (AMIs) across all regions.

```
PS C:\> Get-AWSRegion | % { Get-EC2Image -Owner self -Region $_ }
```

Stop all Amazon EC2 instances in the current default region.

```
PS C:\> Get-EC2Instance | Stop-EC2Instance
```

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

$AWSHistory

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

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

Starting with version 1.1, this data and more is now available in a new shell variable named `$AWSHistory`. This variable maintains a record of AWS cmdlet invocations and the service responses that were received for each invocation. Optionally, this history can be configured to also record the service requests that each cmdlet made. Additional useful data, such as the overall execution time of the cmdlet, can also be obtained from each entry.

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

CmdletName

Name of the cmdlet.

CmdletStart

DateTime that the cmdlet was run.

CmdletEnd

DateTime that the cmdlet finished all processing.

Requests

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

Responses

List of last service responses received.

LastServiceResponse

Helper to return the most recent service response.

LastServiceRequest

Helper to return the most recent service response, if available.

Note that the `$AWSHistory` variable is not created until an AWS cmdlet making a service call is used. It evaluates to $null until that point.

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 C:\> 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 AWS cmdlet activity. The `-MaxServiceCallHistory` parameter sets the maximum number of service responses (and/or requests) that are tracked for each cmdlet. The `-RecordServiceRequests` parameter, if specified, turns on tracking of service requests for each cmdlet. 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 AWS cmdlets that are being held in the list to the pipeline.

```
PS C:\> $AWSHistory.Commands
```

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

```
PS C:\> $AWSHistory.LastCommand
```

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

```
PS C:\> $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 C:\> $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.

```
$c = Get-S3Object -BucketName test
```

If you want to retain control of the amount of data returned, you can continue to use parameters on the individual cmdlets (e.g. `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.

```
$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.

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

See Also

- Using the AWS Tools for Windows PowerShell (p. 40)

Configuring Federated Identity with the AWS Tools for Windows PowerShell

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

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

Topics

Prerequisites

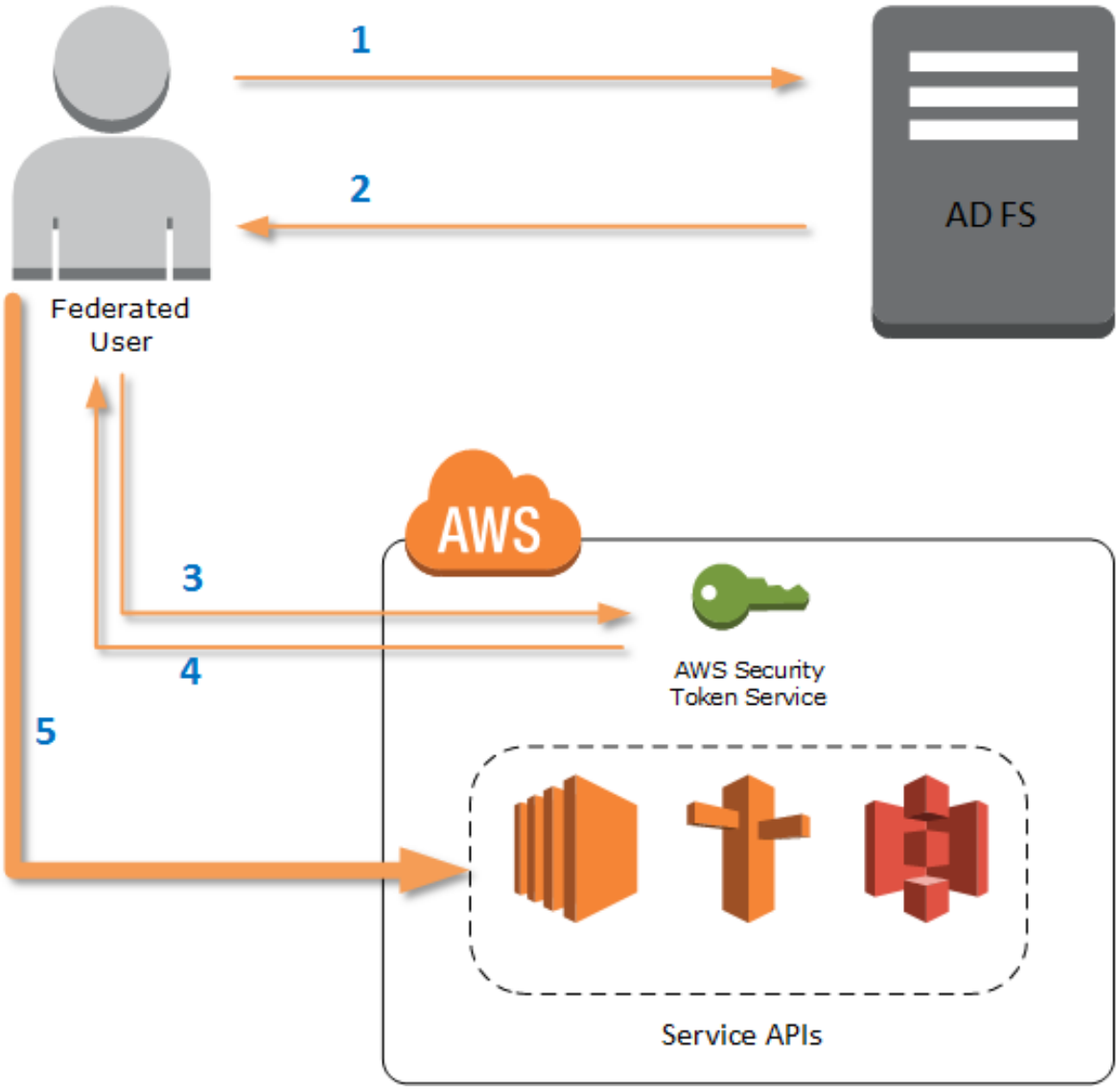
You must have the following in place before you try to use SAML support for the first time.

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

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

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

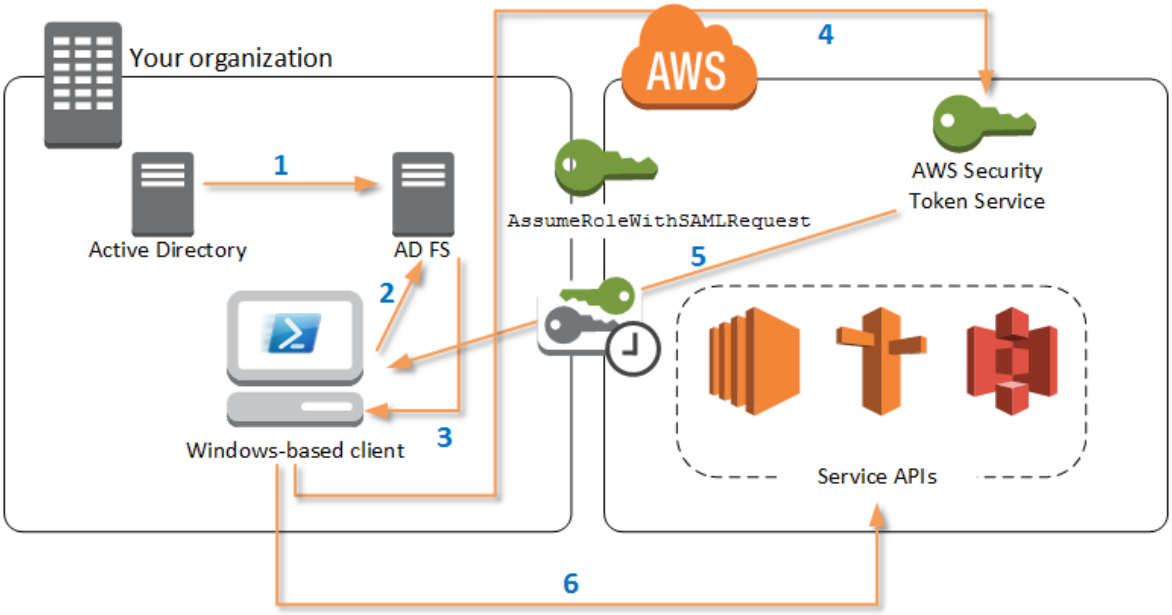
The following process describes, at a high level, how an Active Directory (AD) user is federated by AD FS to gain access to AWS resources.



1. The federated user authenticates against AD FS.
2. If authentication is successful, the user is sent a SAML assertion.
3. The SAML assertion is sent to the AWS Security Token Service (STS) in the form of a SAML request.
4. If the SAML request is valid, STS returns a SAML response that contains the user's AWS temporary credentials.
5. The AWS temporary credentials can be used to work with AWS service APIs by using tools including the Tools for Windows PowerShell.

How SAML Support Works in the Tools for Windows PowerShell

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



1. The Tools for Windows PowerShell authenticate against AD FS by using the Windows user's current credentials, or interactively, when a cmdlet that requires credentials to call into AWS is run.
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. The PowerShell cmdlet extracts the list of the user's authorized roles from the SAML assertion.
4. The PowerShell cmdlet forwards the SAML request, including the requested role Amazon Resource Names (ARN), to STS by making the `AssumeRoleWithSAMLRequest` API call.
5. If the SAML request is valid, STS returns a response that contains the AWS `AccessKeyId`, `SecretAccessKey`, and `SessionToken`. These credentials last for 3,600 seconds (1 hour).
6. The Tools for Windows PowerShell user now has valid credentials to work with any AWS service APIs that the user's role is authorized to access. The Tools for Windows PowerShell automatically apply these credentials for any subsequent AWS API calls, and renew them automatically when they expire.

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

How to Use the PowerShell SAML Configuration Cmdlets

The Tools for Windows PowerShell include two new cmdlets that provide SAML support.

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

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

Open a new Tools for Windows PowerShell session. If you are running PowerShell 3.0 or newer, the Tools for Windows 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. To do this, run the Import-Module cmdlet, as shown in the following example.

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

1. 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 C:\> $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.
2. Now, you 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 C:\> Set-AWSSamlRoleProfile -StoreAs SAMLDemoProfile -EndpointName $epName
   ```

   Alternatively, for non-domain-joined accounts, you can provide Active Directory credentials, and then select an AWS role to which the user has access, as shown in the following line. This is useful if

you have different Active Directory user accounts to differentiate roles within your organization (for example, administration functions).

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

3. In either case, the `Set-AWSSamlRoleProfile` cmdlet prompts you to choose which role should be stored in the profile. The following example uses the `ADFS-Dev` role.

```
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"): 1
```

   You can also specify a role without waiting for the prompt, by entering the `RoleARN`, `PrincipalARN`, and optional `NetworkCredential` parameters (provided the role exists in the assertion returned by authentication. If it does not exist, the user is prompted to choose from available roles.)

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

4. 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 C:\> Set-AWSSamlRoleProfile -EndpointName $epName -StoreAllRoles
ADFS-Dev
ADFS-Production
```

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

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

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

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

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

This example sets a default role for a Tools for Windows 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 C:\> Set-AWSCredential -ProfileName SAMLDemoProfile
PS C:\> 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 Simple Storage Service buckets in the AWS account of the role associated with the `SAMLDemoProfile` profile. The example shows that although you might have been using another profile earlier in your Tools for Windows 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 C:\> 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.

Though the credentials expire after 1 hour (a limit enforced by STS), the Tools for Windows PowerShell automatically refresh the credentials by requesting a new SAML assertion when the tools detect 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, the Tools for Windows PowerShell show 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 `ADFS-Production` profile. This is a useful command for returning all Amazon EC2 instances in a region.

```
PS C:\> (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 questions or comments, be sure to visit the AWS Developer Forums for PowerShell Scripting or .NET Development.

Using the AWS Tools for Windows PowerShell

Topics

This section provides examples of using the AWS Tools for Windows PowerShell to access AWS services. These examples are intended to demonstrate how to use the cmdlets to perform actual administrative tasks.

Note Regarding Returned Objects for the Powershell Tools

In some cases, the object returned from an Tools for Windows PowerShell cmdlet does not mirror what is returned from the corresponding API in the AWS SDK for .NET. For example, `Get-S3Bucket` emits a `Buckets` collection, not an Amazon S3 response object. Similarly, `Get-EC2Instance` emits a `Reservation` collection, not a `DescribeEC2Instances` result object. This behavior is by design and is intended to have the Tools for Windows PowerShell experience be more consistent with idiomatic PowerShell.

The actual service responses are stored in `note` properties on the returned objects and are therefore available if you need to access them. For API actions that support `NextToken` fields, these are also attached as `note` properties.

Amazon EC2 (p. 49)

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.
- Create and configure a security group.
- Launch the instance and retrieve information about it.

Amazon S3 (p. 41)

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

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

AWS Identity and Access Management (p. 46)

This section demonstrates basic operations in AWS 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 SNS and Amazon SQS (p. 58)

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. 62)

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 AWS Tools for Windows PowerShell (p. 16)

Amazon S3 and Tools for Windows PowerShell

Topics

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

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

There is currently no Amazon S3 API for renaming buckets and objects, and therefore, no single Tools for Windows PowerShell cmdlet for performing this task. To rename objects in S3, we recommend that you copy the object using 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 AWS Tools for Windows PowerShell (p. 40)
- 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 C:\> New-S3Bucket -BucketName website-example -Region us-west-1

BucketName                                                  CreationDate
----------                                                  ------------
website-example                                    Mon, 26 Nov 2012 00:41:08 GMT
```

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

```
PS C:\> Get-S3BucketLocation -BucketName website-example
us-west-1
```

You could use the following line to remove this bucket. We suggest that you leave this bucket in place as we use it in subsequent examples.

```
Remove-S3Bucket -BucketName website-example
```

Note that the bucket removal process takes some time to finish. If you try to create a same-named bucket immediately, the `New-S3Bucket` cmdlet might fail for a period of time.

See Also

- Using the AWS Tools for Windows PowerShell (p. 40)
- Put Bucket (Amazon S3 Service Reference)
- AWS PowerShell Regions for Amazon S3

Configure an Amazon S3 Bucket as a Website and Enable Logging

Use the `Write-S3BucketWebsite` cmdlet to configure an Amazon S3 bucket as a static website. The following example specifies a name of `index.html` for the default content web page and a name of

`error.html` for the default error web page. Note that this cmdlet does not create those pages. They need to be uploaded as Amazon S3 objects (p. 43).

```
PS C:\> 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 AWS Tools for Windows PowerShell (p. 40)
- 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 C:\> # Create the two files using here-strings and the Set-Content cmdlet
PS C:\> $index_html = @"
>> <html>
>>   <body>
>>     <p>
>>       Hello, World!
>>     </p>
>>   </body>
>> </html>
>> "@
>>
PS C:\> $index_html | Set-Content index.html
PS C:\> $error_html = @"
>> <html>
>>   <body>
>>     <p>
>>       This is an error page.
>>     </p>
>>   </body>
>> </html>
>> "@
>>
PS C:\> $error_html | Set-Content error.html
PS C:\> # Upload the files to Amazon S3 using a foreach loop
PS C:\> foreach ($f in "index.html", "error.html") {
>> Write-S3Object -BucketName website-example -File $f -Key $f -CannedACLName public-read
>> }
>>
```

```
PS C:\> # Verify that the files were uploaded
PS C:\> Get-S3BucketWebsite -BucketName website-example

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

Canned ACL Options

The values for specifying canned ACLs with the Tools for Windows PowerShell are the same as those used by the AWS SDK for .NET. Note, however, that these are different from the values used by the Amazon S3`Put 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 AWS Tools for Windows PowerShell (p. 40)
- 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.

Run the `Remove-S3Object` cmdlet to delete the objects for the HTML files from the Amazon S3 bucket.

```
PS C:\> 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.

Run the `Remove-S3Bucket` cmdlet to delete the now-empty Amazon S3 bucket for the site.

```
PS C:\> 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 Tools for Windows 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 for `Remove-S3Bucket` to delete it.

Note that unless you add the -Force parameter, you are prompted for confirmation before the cmdlet runs.

See Also

- Using the AWS Tools for Windows PowerShell (p. 40)
- 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.

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

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

```
>>

# Specifying content from a variable: (note final newline needed to end in-line here-
string)
$x = @"
>> line 1
>> line 2
>> line 3
>> "@
>>
write-s3object mybucket -key myobject.txt -content $x
```

IAM and Tools for Windows PowerShell

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

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 C:\> 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 C:\> 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 C:\> 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 C:\> Get-IAMGroup -GroupName powerUsers

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

You can also view IAM users and groups with the AWS Management Console

http://console.aws.amazon.com/iam/home?#s=Users [Users View]

http://console.aws.amazon.com/iam/home?#s=Groups [Groups View]

See Also

- Using the AWS Tools for Windows PowerShell (p. 40)
- Adding a New User to Your AWS 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 C:\> $policyDoc = @"
>> {
>>   "Version": "2012-10-17",
>>   "Statement": [
>>     {
>>       "Effect": "Allow",
>>       "NotAction": "iam:*",
>>       "Resource": "*"
>>     }
>>   ]
>> }
>> "@
>>

PS C:\> Write-IAMUserPolicy -UserName myNewUser -PolicyName "PowerUserAccess-
myNewUser-201211201605" -PolicyDocument $policyDoc
```

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

See Also

- Using the AWS Tools for Windows PowerShell (p. 40)
- 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 C:\> New-IAMLoginProfile -UserName myNewUser -Password "&!123!&"

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

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

See Also

- Using the AWS Tools for Windows PowerShell (p. 40)
- Managing Passwords
- CreateLoginProfile

Create Security Credentials for an IAM User

The following example uses the `New-IAMAccesskey` cmdlet to create security credentials for an IAM user. A set of security credentials comprises an Access Key ID and a Secret Key. Note that an IAM user can have no more than two sets of credentials at any given time. If you attempt to create a third set, the `New-IAMAccessKey` cmdlet returns an error.

```
PS C:\> New-IAMAccessKey -UserName myNewUser

UserName        : myNewUser
AccessKeyId     : AKIAIOSFODNN7EXAMPLE
Status          : Active
SecretAccessKey : wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKE
CreateDate      : 11/20/2012 4:30:04 PM
```

Use the `Remove-IAMAccessKey` cmdlet to delete a set of credentials for an IAM user. Specify credentials to delete using the Access Key ID.

```
PS C:\> Remove-IAMAccessKey -UserName myNewUser -AccessKeyId AKIAIOSFODNN7EXAMPLE

ServiceResponse
```

```
---------------
Amazon.IdentityManagement.Model.DeleteAccessKeyResponse
```

Amazon EC2 and Tools for Windows PowerShell

You can perform common tasks related to Amazon EC2 using the AWS Tools for Windows 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 AWS Tools for Windows PowerShell (p. 16).

Topics

Creating a Key Pair

The following example uses the `New-EC2KeyPair` cmdlet to create a key pair. The returned object is stored in the PowerShell variable `$myPSKeyPair`

```
PS C:\> $myPSKeyPair = New-EC2KeyPair -KeyName myPSKeyPair
```

Pipe the key pair object into the `Get-Member` cmdlet to view the object's members.

```
PS C:\> $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 C:\> $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----
                   MIIEogIBAAKCAQEAkK+ANYUS9c7niNjYfaCn6KYj/D0I6djnFoQE...
                   Mz6btoxPcE7EMeH1wySUp8nouAS9xbl9l7+VkD74bN9KmNcPa/Mu...
                   Zyn4vVe0Q5il/MpkrRogHqOB0rigeTeV5Yc3lvO0RFFPu0Kz4kcm...
                   w3Jg8dKsWn0plOpX7V3sRC02KgJIbejQUvBFGi5OQK9bm4tXBIeC...
                   daxKIAQMtDUdmBDrhR1/YMv8itFe5DiLLbq7Ga+FDcS85NstBa3h...
```

```
                    iuskGkcvgWkcFQkLmRHRoDpPb+OdFsZtjHZDpMVFmA9tT8EdbkEF...
                    3SrNeqZPsxJJIxOodb3CxLJpg75JU5kyWnb0+sDNVHoJiZCULCr0...
                    GGlLfEgB95KjGIk7zEv2Q7K6s+DHclrDeMZWa7KFNRZuCuX7jssC...
                    xO98abxMr3o3TNU6p1ZYRJEQ0oJr0W+kc+/8SWb8NIwfLtwhmJEy...
                    1BX9X8WFX/A8VLHrT1elrKmLkNECgYEAwltkV1pOJAFhz9p7ZFEv...
                    vvVsPaF0Ev9bk9pqhx269PB5Ox2KokwCagDMMaYvasWobuLmNu/1...
                    lmwRx7KTeQ7W1J3OLgxHA1QNMkip9c4Tb3q9vVc3t/fPf8vwfJ8C...
                    63g6N6rk2FkHZX1E62BgbewUd3eZOS05Ip4VUdvtGcuc8/qa+e5C...
                    KXgyt9nl64pMv+VaXfXkZhdLAdY0Khc9TGB9++VMSG5TrD15YJId...
                    gYALEI7m1jJKpHWAEs0hiemw5VmKyIZpzGstSJsFStERlAjiETDH...
                    YAtnI4J8dRyP9I7BOVOn3wNfIjk85gi1/0Oc+j8S65giLAfndWGR...
                    9R9wIkm5BMUcSRRcDy0yuwKBgEbkOnGGSD0ah4HkvrUkepIbUDTD...
                    AnEBM1cXI5UT7BfKInpUihZi59QhgdK/hkOSmWhlZGWikJ5VizBf...
                    drkBr/vTKVRMTi3lVFB7KkIV1xJxC5E/BZ+YdZEpWoCZAoGAC/Cd...
                    TTld5N6opgOXAcQJwzqoGa9ZMwc5Q9f4bfRc67emkw0ZAAwSsvWR...
                    x3O2duuy7/smTwWwskEWRK5IrUxoMv/VVYaqdzcOajwieNrblr7c...
                    -----END RSA PRIVATE KEY-----
```

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

Viewing the Fingerprint of Your Key Pair

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

```
PS C:\> 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 C:\> $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` may 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:

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

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

Removing Your Key Pair

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

```
PS C:\> Remove-EC2KeyPair -KeyName myPSKeyPair

Remove-EC2KeyPair
Are you sure you want to remove keypair 'myPSKeyPair'?
```

```
[Y] Yes  [N]  [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 AWS.

Create a Security Group Using Windows PowerShell

You can use the AWS Tools for Windows 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

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, we provide 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 used by client computers.

If you use `0.0.0.0/0`, you enable all IP addresses to access your instance. For the SSH and RDP protocols, this is acceptable for a short time in a test environment, but it's unsafe for production environments. In production, you'll authorize only a specific IP address or range of addresses to access your instance.

Creating a Security Group for EC2-Classic

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

```
PS C:\> New-EC2SecurityGroup -GroupName myPSSecurityGroup -GroupDescription "EC2-Classic
 from PowerShell"

sg-9cf9e5d9
```

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

```
PS C:\> Get-EC2SecurityGroup -GroupNames myPSSecurityGroup

OwnerId             : 123456789012
GroupName           : myPSSecurityGroup
GroupId             : sg-9cf9e5d9
Description         : EC2-Classic from PowerShell
IpPermissions       : {}
IpPermissionsEgress : {}
```

```
VpcId               :
Tags                : {}
```

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 script shows how you enable SSH traffic from a single IP address, `203.0.113.25/32`.

```
PS C:\> $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 has been updated, run the `Get-EC2SecurityGroup` cmdlet again. You can't specify an outbound rule for EC2-Classic.

```
PS C:\> Get-EC2SecurityGroup -GroupNames myPSSecurityGroup

OwnerId             : 123456789012
GroupName           : myPSSecurityGroup
GroupId             : sg-9cf9e5d9
Description         : EC2-Classic from PowerShell
IpPermissions       : {Amazon.EC2.Model.IpPermission}
IpPermissionsEgress : {}
VpcId               :
Tags                : {}
```

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

```
PS C:\> (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 example uses the `New-EC2SecurityGroup` cmdlet to create a security group for the specified VPC.

```
PS C:\> $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 C:\> 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, which works with PowerShell 2.0 and later. For example, here's how you define permissions for TCP ports 22 and 3389 from a single IP address, `203.0.113.25/32`.

```
PS C:\> $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 C:\> 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, use the `IpPermissions` property.

```
PS C:\> ($groupid | Get-EC2SecurityGroup).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 need to specify an Amazon Machine Image (AMI) to serve as a template for the instance. However, the IDs for the AWS Windows AMIs change monthly because AWS provides new AMIs with the latest updates and security enhancements. You can use the Get-EC2Image and Get-EC2ImageByName cmdlets to find the current Windows AMIs and get their IDs.

Topics

Get-EC2Image

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 who corresponds to the credentials with which the cmdlet is invoked.

```
PS C:\> 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. (To test this example, copy all four lines and paste them into the Windows PowerShell for AWS window.)

```
PS C:\> $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...}
Description         : Microsoft Windows Server 2012 RTM 64-bit Locale English Base AMI
 provided by Amazon
Hypervisor          : xen
ImageId             : ami-e49a0b8c
ImageLocation       : amazon/Windows_Server-2012-RTM-English-64Bit-Base-2014.11.19
ImageOwnerAlias     : amazon
ImageType           : machine
KernelId            :
Name                : Windows_Server-2012-RTM-English-64Bit-Base-2014.11.19
OwnerId             : 801119661308
Platform            : Windows
ProductCodes        : {}
Public              : True
RamdiskId           :
RootDeviceName      : /dev/sda1
RootDeviceType      : ebs
SriovNetSupport     :
State               : available
StateReason         :
Tags                : {}
VirtualizationType  : hvm
```

Get-EC2ImageByName

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

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

```
PS C:\> 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 C:\> Get-EC2ImageByName -Names WINDOWS_2012R2_SQL_SERVER_EXPRESS_2014

Architecture        : x86_64
BlockDeviceMappings : {/dev/sda1, xvdca, xvdcb, xvdcc...}
Description         : Microsoft Windows Server 2012 R2 RTM 64-bit Locale English with SQL
 2014 Express AMI provided by Amazon
Hypervisor          : xen
ImageId             : ami-de9c0db6
ImageLocation       : amazon/Windows_Server-2012-R2_RTM-English-64Bit-
SQL_2014_RTM_Express-2014.11.19
ImageOwnerAlias     : amazon
ImageType           : machine
KernelId            :
Name                : Windows_Server-2012-R2_RTM-English-64Bit-
SQL_2014_RTM_Express-2014.11.19
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. You also need the ID of an Amazon Machine Image (AMI). For more information, see the following documentation:

- Creating a Key Pair (p. 49)
- Create a Security Group Using Windows PowerShell (p. 51)
- Find an Amazon Machine Image Using Windows PowerShell (p. 53)

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

The following command creates a single `t1.micro` instance.

```
PS C:\> 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 will be in the `running` state in 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` with the filter and view the `Instances` property.

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

The following is example output.

```
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            : 12/2/2014 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 one you created for the VPC that contains the specified subnet.

```
PS C:\> 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 will be in the `running` state in 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` with the filter and view the `Instances` property.

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

The following is example output.

```
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            : 12/2/2014 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 command requests a Spot Instance in the specified subnet. The security group must be one you created for the VPC that contains the specified subnet.

```
PS C:\> $interface1 = New-Object Amazon.EC2.Model.InstanceNetworkInterfaceSpecification
PS C:\> $interface1.DeviceIndex = 0
PS C:\> $interface1.SubnetId = "subnet-b61f49f0"
PS C:\> $interface1.PrivateIpAddress = "10.0.1.5"
PS C:\> $interface1.Groups.Add("sg-5d293231")
PS C:\> 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 SQS, Amazon SNS and Tools for Windows PowerShell

This section provides instructions to:

- Create an Amazon SQS queue and get queue ARN (Amazon Resource Name).
- Create an Amazon SNS topic.

- Give permissions to the SNS topic so that it can send messages to the queue.
- Subscribe the queue to the SNS topic
- Give IAM users or AWS accounts permissions to publish to the SNS topic and read messages from the SQS queue.
- Verify results by publishing a message to the topic and reading the message from the queue.

Create an Amazon SQS queue and get queue ARN

The following command creates an SQS queue:

```
New-SQSQueue -QueueName MyQueue -Region us-west-2
```

The URL of the created queue is returned:

```
https://sqs.us-west-2.amazonaws.com/123456789012/MyQueue
```

The following command gets the queue ARN

```
Get-SQSQueueAttribute -QueueUrl https://sqs.us-west-2.amazonaws.com/123456789012/MyQueue -
AttributeName QueueArn -Region us-west-2
```

The ARN of the created queue is returned:

```
...
QueueARN : arn:aws:sqs:us-west-2:123456789012:MyQueue
...
```

Create an Amazon SNS topic

The following command creates an SNS topic:

```
New-SNSTopic -Name MyTopic -Region us-west-2
```

The ARN of the created topic is returned:

```
arn:aws:sns:us-west-2:123456789012:MyTopic
```

Give permissions to the SNS topic

The following command gives permissions to the SNS topic so that it can send messages to the 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",
       "Id": "$qarn/SQSPOLICY",
       "Statement": [
           {
               "Sid": "1",
               "Effect": "Allow",
               "Principal": "*"
               },
               "Action": "SQS:SendMessage",
               "Resource": "$qarn",
               "Condition": {
                   "ArnEquals": {
                       "aws:SourceArn": "$topicarn"
                   }
               }
           }
       ]
   }
   "@

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

The following is returned:

```
ServiceResponse
---------------
<?xml version="1.0" encoding="utf-16"?>...
```

Subscribe the queue to the SNS topic

The following command subscribes the queue *MyQueue* to the SNS topic *MyTopic*:

```
Connect-SNSNotification -TopicARN arn:aws:sns:us-west-2:123456789012:MyTopic -Protocol SQS
 -Endpoint arn:aws:sqs:us-west-2:123456789012:MyQueue -Region us-west-2
```

The Subscription Id is returned:

```
arn:aws:sns:us-west-2:123456789012:ps-cmdlet-topic:f8ff77c6-e719-4d70-8e5c-a54d41feb754
```

Give permissions

The following command gives permission to perform the `sns:Publish` action on the topic *MyTopic*

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

The following is returned:

```
ServiceResponse
---------------
<?xml version="1.0" encoding="utf-16"?>...
```

The following command gives permission to perform the `sqs:ReceiveMessage` and `sqs:DeleteMessage` actions on the queue *MyQueue*

```
Add-SQSPermission -QueueUrl https://sqs.us-west-2.amazonaws.com/123456789012/MyQueue
 -Region US-West-2 -AWSAccountId "123456789012" -Label queue-permission -ActionName
 SendMessage, ReceiveMessage
```

The following is returned:

```
ServiceResponse
---------------
<?xml version="1.0" encoding="utf-16"?>...
```

Verify results

The following command publishes a message to the SNS topic *MyTopic*

```
Publish-SNSMessage -TopicArn arn:aws:sns:us-west-2:123456789012:MyTopic -Message "Have A
 Nice Day!" -Region us-west-2
```

The `MessageId` is returned:

```
4914beb6-f8d2-5568-989f-f7909cefab79
```

The following command retrieves the message from the SQS queue *MyQueue*

```
Receive-SQSMessage -QueueUrl https://sqs.us-west-2.amazonaws.com/123456789012/MyQueue -
Region us-west-2
```

The following is returned:

```
MessageId     : 03204f1d-1d65-4733-9eed-fc9cd514873a
ReceiptHandle : uUk89DYFzt3SjcTMtVq9VLAxpcJU5hHOKkInt
+Hq6AxnWLGl1Eg1RLnPlIrkrflNmujk8+p2HrTCw0+1nLHAA+rfcy0m0f7Hxvm9iGR
                WMcFcCp4woccvYwQJW/if62D8R14v4JtSltEiY2ukxl/Zb4xqC9WN3+M0YZ/HW1euFb/
tIE0qLQnKcOyoQ4Hj1d5WGc/IFo0cYNvOuM
                x8pRxeyOHKpah8OTrFiQFcCXbMKiuTqOI6yceInyAJ8YWwfKpjatc2zUcq5PqcrYMtbs4jK/
zJc4uVhZNMUmCu2fA5EM4=
MD5OfBody     : 60509281ad1bfd6980e84f9d64bbf9ab
Body          : {
                  "Type" : "Notification",
                  "MessageId" : "4914beb6-f8d2-5568-989f-f7909cefab79",
                  "TopicArn" : "arn:aws:sns:us-west-2:803981987763:MyTopic",
                  "Message" : "Have A Nice Day!",
                  "Timestamp" : "2012-11-21T05:09:17.905Z",
                  "SignatureVersion" : "1",
                  "Signature" : "GpF4Dhb5GotbtK883ccm1s59+7vnZMdcjxrAVYU7+igDFVWrvI6/
bDfws5GcjT/IP9GxG6UJ55b8pu1+jzujaN
                YhZpr52mJfQHGRtM8FN0IAcCDDRQ00tXCHlOa6GP1s7RVIUNgCOzR/tbCCpJolGace
+j0F1uf26LN4453RR6o=",
                  "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/
SimpleNotificationService-f3ecfb7224c7233fe7b
                b5f59f96de52f.pem",
                  "UnsubscribeURL" : "https://sns.us-west-2.amazonaws.com/?
Action=Unsubscribe&SubscriptionArn=arn:aws:s
                ns:us-west-2:803981987763:ps-cmdlet-topic:f8ff77c6-e719-4d70-8e5c-
a54d41feb754"
                }
Attribute     : {}
```

CloudWatch from the AWS Tools for Windows PowerShell

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

This example assumes that you have set default credentials and a default region for your PowerShell session. Therefore, credentials and a region are not included in the invocation of the cmdlets.

Publish a Custom Metric to Your CloudWatch Dashboard

The following PowerShell code initializes an CloudWatch MetricDatum object and posts it to the service. You can see the result of this operation by navigating to the CloudWatch console.

```
$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). A string value is shown previously.

See Also

- Using the AWS Tools for Windows PowerShell (p. 40)
- AmazonCloudWatchClient.PutMetricData (.NET SDK Reference)
- MetricDatum (Service API Reference)
- Amazon CloudWatch Console

Document History

This topic describes significant changes to the documentation for the AWS Tools for Windows PowerShell.

Last documentation update: December 1, 2015

We also update the documentation periodically in response to customer feedback. To send us feedback, use the feedback link—"Tell us about it," next to "Did this page help you?"—located near the beginning of each page.

For additional information about changes and updates to the Tools for Windows PowerShell, see the release notes.

AWS Tools for Windows PowerShell 3.1.31.0

Release Date: 2015-12-01

Summary of Changes

- Added information to the Getting Started section about new cmdlets that use Security Assertion Markup Language (SAML) to support configuring federated identity for users.

AWS Tools for Windows PowerShell 2.3.19

Release Date: 2015-02-05

Summary of Changes

- Added information to the Cmdlets Discovery and Aliases section about the new `Get-AWSCmdletName` cmdlet that can help users more easily find their desired AWS cmdlets.

AWS Tools for Windows PowerShell 1.1.1.0

Release Date: 2013-05-15

Summary of Changes

- 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

AWS Tools for Windows PowerShell 1.0.1.0

Release Date: 2012-12-21

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 **Amazon Web ServicesWindows PowerShell for AWS** 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 **Amazon Web ServicesWindows PowerShell for AWS** 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 with an execution policy that disallows script execution.

AWS Tools for Windows PowerShell 1.0.0.0

Release Date: 2012-12-06

Initial release