
AWS Command Line Interface

User Guide



AWS Command Line Interface: User Guide

Copyright © 2019 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

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

Table of Contents

What Is the AWS CLI?	1
Using the Examples	2
About Amazon Web Services	3
Installing the AWS CLI	4
AWS CLI version 2	4
AWS CLI version 1	4
Migrating from AWS CLI version 1 to version 2	4
Installing the AWS CLI version 2	4
Installing on Linux or Unix	5
Installing on macOS	9
Installing on Windows	13
Installing the AWS CLI version 1	15
Installing the AWS CLI Using the Bundled Installer	15
Installing the AWS CLI Using pip	15
Installing the AWS CLI in a Virtual Environment	17
Steps to Take after Installation	17
Detailed Instructions for Each Environment	18
Linux	18
macOS	23
Windows	26
Virtualenv	29
Bundled Installer	30
Using the AWS CLI version 1 with Python 2.6 or Python 3.3	32
Migrating from V1 to V2	34
No automatic retrieval of webpages for parameters	34
All date/time values in ISO 8601 format	34
No hidden aliases	35
Consistent Amazon S3 keys and paths	36
No support for [plugins]	36
Configuring the AWS CLI	37
Quickly Configuring the AWS CLI	37
Access Key and Secret Access Key	37
Region	38
Output Format	38
Creating Multiple Profiles	39
Configuration Settings and Precedence	39
Configuration and Credential File Settings	40
Where Are Configuration Settings Stored?	40
Supported config File Settings	41
Named Profiles	50
Using Profiles with the AWS CLI	50
Configuring the AWS CLI to use AWS Single Sign-On	51
Configuring a Named Profile to Use AWS SSO	51
Using an AWS SSO Enabled Named Profile	54
Environment Variables	56
Command Line Options	58
Sourcing Credentials with an External Process	60
Getting Credentials from EC2 Instance Metadata	61
Using an HTTP Proxy	62
Authenticating to a Proxy	63
Using a Proxy on Amazon EC2 Instances	63
Using an IAM Role in the AWS CLI	63
Configuring and Using a Role	64
Using MFA	66

Cross-Account Roles and External ID	67
Specifying a Role Session Name for Easier Auditing	67
Assume Role with Web Identity	68
Clearing Cached Credentials	68
Command Completion	69
Identify Your Shell	69
Locate the AWS Completer	69
Add the Completer's Folder to Your Path	70
Enable Command Completion	71
Test Command Completion	71
Using the AWS CLI	72
Getting Help	72
AWS CLI Documentation	75
API Documentation	75
Command Structure	76
Specifying Parameter Values	76
.....	77
Using JSON for Parameters	78
Using Quotation Marks with Strings	80
Loading Parameters from a File	80
Generate the CLI Skeleton	82
Controlling Command Output	86
How to Select the Output Format	86
JSON Output Format	87
Text Output Format	87
Table Output Format	89
How to Filter the Output with the --query Option	90
Shorthand Syntax	95
Structure Parameters	95
Shorthand Syntax	96
Pagination	97
Return Codes	98
Using the AWS CLI with AWS Services	99
DynamoDB	99
Amazon EC2	101
Amazon EC2 Key Pairs	101
Amazon EC2 Security Groups	103
EC2 Instances	107
Glacier	113
Create an Amazon S3 Glacier Vault	113
Prepare a File for Uploading	113
Initiate a Multipart Upload and Upload Files	114
Complete the Upload	115
IAM	116
Creating IAM Users and Groups	117
Attaching an IAM Managed Policy to an IAM User	118
Setting an Initial Password for an IAM User	119
Create an Access Key for an IAM User	119
Amazon S3	120
High-Level (s3) Commands	120
API Level (s3api) Commands	124
Amazon SNS	126
Create a Topic	126
Subscribe to a Topic	126
Publish to a Topic	127
Unsubscribe from a Topic	127
Delete a Topic	127

Amazon SWF	128
List of Amazon SWF Commands	128
Working with Amazon SWF Domains	131
Troubleshooting Errors	133
Document History	140

What Is the AWS Command Line Interface?

Important

On January 10th, 2020, AWS CLI version 1.17 and later will no longer support Python 2.6 or Python 3.3. After this date, the installer for the AWS CLI will require Python 2.7, Python 3.4, or a later version to successfully install the AWS CLI. For more information, see [Using the AWS CLI version 1 with Python 2.6 or Python 3.3 \(p. 32\)](#) in this guide, and the [deprecation announcement in this blog post](#).

The AWS Command Line Interface (AWS CLI) is an open source tool that enables you to interact with AWS services using commands in your command-line shell.

The AWS CLI is available in two versions:

- **Version 1.x** – The generally available version of the AWS CLI that is suitable for use in production environments.
- **Version 2.x** – A preview version of the AWS CLI that is intended for testing and evaluation. This version does include some "breaking" changes that might require you to change your scripts so that they continue to operate as you expect. For a list of new features and breaking changes in version 2, see [Migrating from AWS CLI version 1 to version 2 \(p. 34\)](#).

Information in this guide applies to both versions unless we specifically state that it applies to only one version or the other.

With minimal configuration, the AWS CLI enables you to start running commands that implement functionality equivalent to that provided by the browser-based AWS Management Console from the command prompt in your favorite terminal program:

- **Linux shells** – Use common shell programs such as `bash`, `zsh`, and `tcsh` to run commands in Linux or macOS.
- **Windows command line** – On Windows, run commands at the Windows command prompt or in PowerShell.
- **Remotely** – Run commands on Amazon Elastic Compute Cloud (Amazon EC2) instances through a remote terminal program such as PuTTY or SSH, or with AWS Systems Manager.

All IaaS (infrastructure as a service) AWS administration, management, and access functions in the AWS Management Console are available in the AWS API and CLI. New AWS IaaS features and services provide full AWS Management Console functionality through the API and CLI at launch or within 180 days of launch.

The AWS CLI provides direct access to the public APIs of AWS services. You can explore a service's capabilities with the AWS CLI, and develop shell scripts to manage your resources. Or, you can take what you learn to develop programs in other languages by using the AWS SDKs.

In addition to the low-level, API-equivalent commands, several AWS services provide customizations for the AWS CLI. Customizations can include higher-level commands that simplify using a service with a complex API. For example, the `aws s3` commands provide a familiar syntax for managing files in Amazon Simple Storage Service (Amazon S3).

Example Upload a file to Amazon S3

`aws s3 cp` provides a shell-like copy command, and automatically performs a multipart upload to transfer large files quickly and resiliently.

```
$ aws s3 cp myvideo.mp4 s3://mybucket/
```

Performing the same task with the low-level commands (available under `aws s3api`) would take a lot more effort.

Depending on your use case, you might want to choose one of the AWS SDKs or the AWS Tools for PowerShell:

- [AWS Tools for PowerShell](#)
- [AWS SDK for Java](#)
- [AWS SDK for .NET](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for Ruby](#)
- [AWS SDK for Python \(Boto\)](#)
- [AWS SDK for PHP](#)
- [AWS SDK for Go](#)
- [AWS Mobile SDK for iOS](#)
- [AWS Mobile SDK for Android](#)

You can view—and fork—the source code for the AWS CLI on GitHub in the [aws-cli repository](#). Join the community of users on GitHub to provide feedback, request features, and submit your own contributions!

Using the Examples

The examples in this guide are formatted using the following conventions:

- **Prompt** – The command prompt is typically displayed as a dollar sign followed by a space (`$`). For commands that are Windows specific, `C:\>` is used as the prompt. Do not include the prompt when you type commands.
- **Directory** – When commands must be executed from a specific directory, the directory name is shown before the prompt symbol.
- **User input** – Command text that you should enter at the command line is formatted as **user input**.
- **Replaceable text** – Variable text, including names of resources that you choose, or IDs generated by AWS services that you must include in commands, is formatted as *replaceable text*. In multiple-line commands or commands where specific keyboard input is required, keyboard commands can also be shown as replaceable text.
- **Output** – Output returned by AWS services is shown under user input, and is formatted as computer output.

For example, the following command includes user input, replaceable text, and output.

```
$ aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-west-2
```

Default output format [None]: **ENTER**

To use this example, enter **aws configure** at the command line, and then press **Enter**. The command is **aws configure**. This command is interactive, so the AWS CLI outputs lines of text, prompting you to enter additional information. Enter each of your access keys in turn, and then press **Enter**. Then, enter an AWS Region name in the format shown, press **Enter**, and then press **Enter** a final time to skip the output format setting. The final **Enter** command is shown as replaceable text because there is no user input for that line. Otherwise, it would be implied.

The following example shows a simple noninteractive command with output from the service in **JSON** format.

```
$ aws ec2 create-security-group --group-name my-sg --description "My security group"
{
  "GroupId": "sg-903004f8"
}
```

To use this example, enter the full text of the command (the highlighted text after the prompt), and then press **Enter**. The name of the security group, **my-sg**, is replaceable. You can use the group name as shown, but you probably want to use a more descriptive name.

Note

Arguments that must be replaced (such as **AWS Access Key ID**), and those that should be replaced (such as **group name**), are both shown as *replaceable text in italics*. If an argument must be replaced, it's noted in the text that describes the example.

The JSON document, including the curly braces, is output. If you configure your CLI to output in text or table format, the output will be formatted differently. **JSON** is the default output format.

About Amazon Web Services

Amazon Web Services (AWS) is a collection of digital infrastructure services that developers can leverage when developing their applications. The services include computing, storage, database, and application synchronization (messaging and queuing). AWS uses a pay-as-you-go service model. You are charged only for the services that you—or your applications—use. Also, to make AWS more approachable as a platform for prototyping and experimentation, AWS offers a free usage tier. On this tier, services are free below a certain level of usage. For more information about AWS costs and the Free Tier, see [Test-Driving AWS in the Free Usage Tier](#). To obtain an AWS account, open the [AWS home page](#) and then click **Sign Up**.

Installing the AWS CLI

The AWS Command Line Interface is available in two versions.

AWS CLI version 2

Preview Evaluation Software

AWS CLI version 2 is provided as a preview for testing and evaluation. At this time, we do not recommend using it in a production environment. For production environments, we recommend that you use the generally available version 1.x.

You can provide feedback for this developer preview version in the [AWS CLI version 2 GitHub repo](#). Be sure to attach the "V2" label to your issue.

AWS CLI version 2 is the most recent major version of the AWS CLI and supports all of the latest features. Some features introduced in version 2 are not backward compatible with version 1 and you must upgrade to access those features.

AWS CLI version 2 is available to install only as a bundled installer. Although you might find it in some package managers, these are not produced or managed by AWS and are therefore not official and not supported by AWS. We recommend that you install the AWS CLI from only the official AWS distribution points, as documented in this guide.

For information about how to install AWS CLI version 2, see [Installing the AWS CLI version 2 \(p. 4\)](#).

AWS CLI version 1

AWS CLI version 1 is the original AWS CLI, and we continue to support it. However, major new features that are introduced in AWS CLI version 2 might not be backported to AWS CLI version 1. To use those features, you must install AWS CLI version 2.

For information about how to install AWS CLI version 1, see [Installing the AWS CLI version 1 \(p. 15\)](#).

Migrating from AWS CLI version 1 to version 2

If you ran commands or scripts with AWS CLI version 1 and you are considering migrating to AWS CLI version 2, see [Migrating from AWS CLI version 1 to version 2 \(p. 34\)](#) for a description of the changes that you should know about.

Installing the AWS CLI version 2

Preview Evaluation Software

AWS CLI version 2 is provided as a preview for testing and evaluation. At this time, we do not recommend using it in a production environment. For production environments, we recommend that you use the generally available version 1.x.

You can provide feedback for this developer preview version in the [AWS CLI version 2 GitHub repo](#). Be sure to attach the "V2" label to your issue.

This topic provides links to information about how to install version 2 of the AWS Command Line Interface (AWS CLI) on the supported operating systems. For information about how to install AWS CLI version 1, see [Installing the AWS CLI version 1 \(p. 15\)](#).

Note

For AWS CLI version 2, it doesn't matter if you have Python installed and if you do, it doesn't matter which version. AWS CLI version 2 uses only the version of Python (and all other dependencies) that are included and installed into a local virtual environment that is isolated.

Topics

- [Installing the AWS CLI version 2 on Linux or Unix \(p. 5\)](#)
- [Installing AWS CLI version 2 on macOS \(p. 9\)](#)
- [Installing AWS CLI version 2 on Windows \(p. 13\)](#)

Installing the AWS CLI version 2 on Linux or Unix

Preview Evaluation Software

AWS CLI version 2 is provided as a preview for testing and evaluation. At this time, we do not recommend using it in a production environment. For production environments, we recommend that you use the generally available version 1.x.

You can provide feedback for this developer preview version in the [AWS CLI version 2 GitHub repo](#). Be sure to attach the "V2" label to your issue.

This section describes how to install, upgrade, and remove the AWS CLI version 2 on Linux or Unix.

Topics

- [Prerequisites for Linux \(p. 5\)](#)
- [Installing on Linux \(p. 5\)](#)
- [Upgrading on Linux \(p. 7\)](#)
- [Removing from Linux \(p. 7\)](#)
- [Verifying the Integrity and Authenticity of the Downloaded Files \(p. 8\)](#)

Prerequisites for Linux

The AWS CLI version 2 has no dependencies on other software packages. It has a self-contained copy of all dependencies included with the installer. You no longer need to have Python installed first.

We test and support the AWS CLI version 2 on recent distributions of CentOS, Fedora, Ubuntu, Amazon Linux 1, and Amazon Linux 2.

Installing on Linux

Follow these steps from the command line to install the AWS CLI on Linux.

We provide the steps in one easy to copy and paste group. See the descriptions of each line in the steps that follow.

Note

- You can verify that integrity and authenticity of the installation file after you download it. For more information, see [Verifying the Integrity and Authenticity of the Downloaded Files \(p. 8\)](#) before you extract the files from the package.

- If you don't have `unzip`, use your favorite package manager to install it or an equivalent.

```
curl "https://d1vvhvl2y92vvt.cloudfront.net/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

1. You can download the file using the `curl` command. The options on the following example command cause the downloaded file to be written to the current directory with the local name `awscliv2.zip`.

```
$ curl "https://d1vvhvl2y92vvt.cloudfront.net/awscli-exe-linux-x86_64.zip" -o
"awscliv2.zip"
```

In this example, the `-o` option specifies the file name that the downloaded package is written to. In the previous example, the file is written to `awscliv2.zip` in the current folder.

Alternatively, you can use your browser to download the installer from the following URL:

https://d1vvhvl2y92vvt.cloudfront.net/awscli-exe-linux-x86_64.zip

You can verify the integrity and authenticity of the installation file after you download it. For more information, see [Verifying the Integrity and Authenticity of the Downloaded Files \(p. 8\)](#) before you unzip the package.

2. Unzip the installer. The following example command unzips the package to the current folder. If your Linux distribution doesn't have a built-in `unzip` command, use your favorite package manager, or an equivalent, to install it.

```
$ unzip awscliv2.zip
```

This creates a folder named `aws` under the current folder.

3. Run the install program.

```
$ sudo ./aws/install
```

The installation command is a file named `install` found in the newly unzipped `aws` folder. By default, the files are all installed to `/usr/local/aws`, and a symlink is created in `/usr/local/bin`. The command includes `sudo` to grant write permissions to those folders. You can install without `sudo` if you specify folders that you already have write permissions to. You can use the following parameters with the `install` command to specify those folders:

- `--install-dir` or `-i`

This option specifies the folder to copy all of the files to. This example installs the files to a folder named `/usr/local/aws-cli`. You must have write permissions to `/usr/local` to create this folder.

The default value is `/usr/local/aws-cli`.

- `--bin-dir` or `-b`

This option specifies that the main `aws` program in the `install` folder is symlinked to the file `aws2` in the specified path. This example creates the symlink `/usr/local/bin/aws2`. You must have write permissions to the specified folder. Creating a symlink to a folder that is already in your path eliminates the need to add the `install` directory to the user's `$PATH` variable.

The default value is `/usr/local/bin`.

Note

The preview release of the AWS CLI version 2 names the symlink `aws2` to enable the AWS CLI version 1 and version 2 to coexist side by side. Future releases of the AWS CLI version 2 might change this command name.

4. Confirm the installation.

```
$ aws2 --version
aws-cli/2.0.0dev0 Python/3.7.3 Linux/4.14.133-113.105.amzn2.x86_64 botocore/2.0.0dev0
```

Upgrading on Linux

To upgrade your copy of the AWS CLI version 2 running on Linux, run the same [steps that you used to install](#) (p. 5) it, but this time include the `--update` or `-u` option on the `install` command line. If the installer finds an existing version of the AWS CLI version 2 in the target installation folder and the `--update` option isn't used, the install fails.

Find the symlink that the installer created.

```
$ which aws2
/usr/local/bin/aws2
```

Use that to find the folder that the symlink points to.

```
$ ls -l /usr/local/bin/aws2
lrwxrwxrwx 1 ec2-user ec2-user 49 Oct 22 09:49 /usr/local/bin/aws2 -> /usr/local/aws-cli/v2/current/bin/aws
```

Then use that information to construct the install command.

```
$ sudo ./aws/install --install-dir /usr/local/aws-cli --bin-dir /usr/local/bin --update
```

Removing from Linux

To uninstall the AWS CLI version 2 from Linux, run the following commands, substituting the paths you used to install.

Find the symlinks that you created in the `--bin-dir` folder.

```
$ which aws2
/usr/local/bin/aws2
```

Use that to find the `--install-dir` folder that the symlink points to.

```
$ ls -l /usr/local/bin/aws2
lrwxrwxrwx 1 ec2-user ec2-user 49 Oct 22 09:49 /usr/local/bin/aws2 -> /usr/local/aws-cli/v2/current/bin/aws
```

Now delete the two symlinks in the `--bin-dir` folder. If your user account has write permission to these folders, you don't need to use `sudo`.

```
$ sudo rm /usr/local/bin/aws2
```

```
$ sudo rm /usr/local/bin/aws2_completer
```

Finally, you can delete the `--install-dir` folder. Again, if your user account has write permission to this folder, you don't need to use `sudo`.

```
$ sudo rm -rf /usr/local/aws-cli
```

Verifying the Integrity and Authenticity of the Downloaded Files

The AWS CLI version 2 installer package .zip files are cryptographically signed using PGP signatures. You can use the following steps to verify the signatures by using the `GnuPG` tool. If there is any damage or alteration of the files, this verification fails and you should not proceed with installation.

The following example assumes you downloaded the installer package and saved it locally as `awscli.v2.zip`. If you named it something else, substitute that name in the following steps.

Steps 1, 2, and 3 are prerequisites, and you need to perform them only once. You should perform steps 4 and 5 every time you download a copy of the installer package.

To validate the files using the PGP signature

1. Download and install the `gpg` command using the package manager for your version of Linux. For more information about `GnuPG`, see the [GnuPG website](#).
2. Create a text file and paste in the following text.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBF2Cr7UBEADJZHcgusOJl7ENSyUmXh85z0TRV0xJorM2B/JL0kHOyigQluUG
ZMLhENAG0bYatdrKP+3H91lvK05OpXwnO/R7fB/FSTouki4ciIx5OuLlnJZIXSzx
PqG10mkxImLNBGwoi6LtoOLYxqHN2iQtzlwTVmq9733zd3XfcXrZ3+Lb1HAgEt5G
TfnxEKJ8soPLyWmwdH6HWCnjZ/aIQRBtIQ05uVeEoYxSh6wOai7ss/KveoSNBbYz
gbdzogI2Y8cgH2nbfgp3DSasaLZEdCSsIsK1u05CinE7k2qZ7KgKAUIcT/cR/grk
C6VwsnDU0UOCideXcQ8WeHutqvqgZH1JgKDbznoIzeQHJD238GEu+eKhRHcz8/jeG
94zkcgJOz3KbZGYMiTh277Fvj9zzvZsbMBCedV1BTg3TggvdX4bdkhf5ch+7NtWO
lrFj6UwAsGukBTA0xC0l/dnSmZhJ7Z1KmEWilro/gOrjtOxqRQutlIqG22TaqoPG
fYVN+en3Zwbt97kcgZDwqbuykNt64oZWc4XKCa3mprEGC3IbJTBFGglXmZ7l9yWg
EEUJY0lhb2XrSuPWml39beWdKM8kzr1OjnlOm6+lpTRCBfo0wa9F8YZRhHPakwKkX
XDeOGpWRj4ohOx0d2GWkyV5xyN14p2tQOCdO0Dmz80yUTGrpPVQUtOEhXQARAQAB
tCFBV1MgQ0xJFRlYW0gPGF3cy1jbGlAYW1hem9uLmNvbT6JAlQEEwEiAD4WIQT7
Xbd/1cEYUaURraimMQrMRnJHXAUcXYKvtQIbAwUJB4TOAAULCQGHAgYVCgkICwIE
FgIDAQIeAQIXgAAKCRcmMQrMRnJHXJIXEACHLUIkg8OuPUKgjE3jejvQSA1aWuAM
yzy6fdpdlRUz6M6nmsUhoExjVIvibEJpzK5mhuS41b0vJ2ZUPgCv4zs2nBd7BGJ
MxKiWGBREgVtdqZ0SzyYH4PYCJSE732x/Fw9hfnh1dMTXNcrQXzwOmmFNNegG00x
au+VnpcR5Kz3smiTrIwZbRudo1ijhCYPQ7t5CMP9kjC6bObvy1hSIg2xNbMAN/Do
ikebAl36uA6Y/Uczjj3GxZW4ZWeFirmidKbtqvUz2yOUFSzobjiBSqZZHCreC34B
hw9bFNpuWC/OSrXgohdsc6vK50pDgDV5kM2qo9tMQ/izsAwTh/d/Gzv8H4lV9e0
tEis+EpR497PaxKKh9tJfON6Q1YLRHof5xePZtOIls3gfvS5hXA3HJ9yIxb8TOH
QYmVr3aIUes20i6meI3fuV36VFupwfrTKaL7VXnsrK2fq5cRvyJLNzXucg0WAjPF
RrAGLzY7nP1xeg1a0aep+pdsgjqlPJom8OCWc1+6DWbg0jsC74WoesAggBItoDMB
rsal1y/q+bPzpsnWjzHV8+1/EtZmSc8ZUGSJOPkfc7hObnfkl18h+1QtKTjZme4d
H17gsBJr+opwJw/Zio2LMjQBOqlm3K1A4zFTh7wBC7He6KPQea1p2XAMgtvAttNe
YLZATHZKTJyiqA==
=vY0k
-----END PGP PUBLIC KEY BLOCK-----
```

Here are the details of the public key for reference.

Key ID:	A6310ACC4672
Type:	RSA

```
Size:                4096/4096
Created:             2019-09-18
Expires:             2023-09-17
User ID:             AWS CLI Team <aws-cli@amazon.com>
Key fingerprint:    FB5D B77F D5C1 18B8 0511 ADA8 A631 0ACC 4672 475C
```

3. Import the AWS CLI public key with the following command, substituting *public-key-file-name* with whatever you named the file in step 2.

```
$ gpg --import public-key-file-name
gpg: /home/username/.gnupg/trustdb.gpg: trustdb created
gpg: key A6310ACC4672475C: public key "AWS CLI Team <aws-cli@amazon.com>" imported
gpg: Total number processed: 1
gpg:             imported: 1
```

4. Download the AWS CLI signature file for the package you downloaded. It has the same path and name as the .zip file it corresponds to, but has the extension .sig.

```
$ curl -o awscliv2.sig https://d1vvhv12y92vvt.cloudfront.net/awscli-exe-linux-x86\_64.zip.sig
```

5. Verify the signature, passing both the .sig and .zip file names as parameters to the gpg command.

```
$ gpg --verify awscliv2.sig awscliv2.zip
```

The output should look similar to the following.

```
gpg: Signature made Mon Nov  4 19:00:01 2019 PST
gpg:             using RSA key FB5D B77F D5C1 18B8 0511 ADA8 A631 0ACC 4672 475C
gpg: Good signature from "AWS CLI Team <aws-cli@amazon.com>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:             There is no indication that the signature belongs to the owner.
Primary key fingerprint: FB5D B77F D5C1 18B8 0511 ADA8 A631 0ACC 4672 475C
```

Important

The warning in the output is expected and doesn't indicate a problem. It occurs because there isn't a chain of trust between your personal PGP key (if you have one) and the AWS CLI PGP key. For more information, see [Web of trust](#).

Installing AWS CLI version 2 on macOS

This section describes how to install, upgrade, and remove AWS CLI version 2 on macOS.

Preview Evaluation Software

AWS CLI version 2 is provided as a preview for testing and evaluation. At this time, we do not recommend using it in a production environment. For production environments, we recommend that you use the generally available version 1.x.

You can provide feedback for this developer preview version in the [AWS CLI version 2 GitHub repo](#). Be sure to attach the "V2" label to your issue.

Topics

- [Prerequisites for macOS \(p. 10\)](#)
- [Installing on macOS \(p. 10\)](#)
- [Upgrading on macOS \(p. 11\)](#)
- [Removing from macOS \(p. 11\)](#)
- [Verifying the Integrity and Authenticity of the Downloaded Files \(p. 12\)](#)

Prerequisites for macOS

AWS CLI version 2 has no dependencies on other software packages. The installer includes a self-contained copy of all dependencies.

Installing on macOS

Follow these steps from the command line to install AWS CLI on macOS.

We provide the steps in one easy to copy and paste group. See the descriptions of each line in the steps that follow.

Note

- You can verify that integrity and authenticity of the installation file after you download it. For more information, see [Verifying the Integrity and Authenticity of the Downloaded Files \(p. 12\)](#) before you extract the files from the package.
- If you don't have unzip, use your favorite package manager to install it or an equivalent.

```
curl "https://d1vvhvl2y92vvt.cloudfront.net/awscli-exe-macos.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

1. You can download the file using the `curl` command. The options on the following example command cause the downloaded file to be written to the current directory with the final name `awscliv2.zip`.

```
$ curl "https://d1vvhvl2y92vvt.cloudfront.net/awscli-exe-macos.zip" -o "awscliv2.zip"
```

In these examples, the `-o` option specifies the file that the downloaded package is written to. In the previous example, the file is written to `awscliv2.zip` in the current folder.

Alternatively, you can use your browser to download the installer from the following URL:

<https://d1vvhvl2y92vvt.cloudfront.net/awscli-exe-macos.zip>

You can verify the integrity and authenticity of the installation file after you download it. For more information, see [Verifying the Integrity and Authenticity of the Downloaded Files \(p. 12\)](#) before you unzip the package

2. Unzip the installer. The following example command unzips the package to the current folder.

```
$ unzip awscliv2.zip
```

This creates a folder named `aws` under the current folder.

3. Run the install program.

```
$ sudo ./aws/install
```

The installation command is a file named `install` found in the newly unzipped `aws` folder. By default, the files are all installed to `/usr/local/aws`, and a symlink is created in `/usr/local/bin`. The command includes `sudo` to grant write permissions to those folders. You can install without `sudo` if you specify folders that you already have write permissions to. You can use the following parameters with the `install` command to specify those folders:

- `--install-dir` or `-i`

This option specifies the folder to copy all of the files to. You must have write permissions to `/usr/local` to create this folder.

The default value is `/usr/local/aws-cli`.

- `--bin-dir` or `-b`

This option specifies that the main `aws` program in the install folder is symlinked to the file `aws2` in the specified path. You must have write permissions to the specified folder. Creating a symlink to a folder that is already in your path eliminates the need to add the install directory to the user's `$PATH` variable.

The default value is `/usr/local/bin`.

Note

The preview release of the AWS CLI version 2 names the symlink `aws2` to enable the AWS CLI version 1 and version 2 to coexist side by side. Future revisions to the AWS CLI version 2 might change this command name.

4. Confirm the installation.

```
$ aws2 --version
aws-cli/2.0.0dev0 Python/3.7.3 Linux/4.14.133-113.105.amzn2.x86_64 botocore/2.0.0dev0
```

Upgrading on macOS

To upgrade your copy of the AWS CLI version 2 running on macOS, run the same [steps that you used to install](#) (p. 5), but this time include the `--update` or `-u` option on the `install` command line. If the installer finds an existing version of the AWS CLI version 2 in the target installation folder and the `--update` option isn't used, the install fails.

Find the symlink that the installer created.

```
$ which aws2
/usr/local/bin/aws2
```

Use that to find the folder that the symlink points to.

```
$ ls -l /usr/local/bin/aws2
lrwxrwxrwx 1 ec2-user ec2-user 49 Oct 22 09:49 /usr/local/bin/aws2 -> /usr/local/aws-cli/v2/current/bin/aws
```

Then use that information to construct the install command.

```
$ ./aws/install --install-dir /usr/local/aws-cli --bin-dir /usr/local/bin --update
```

Removing from macOS

To uninstall the AWS CLI version 2 from macOS, run the following commands, substituting the paths you used to install.

Find the symlinks that you created in the `--bin-dir` folder.

```
$ which aws2
/usr/local/bin/aws2
```


Use that to find the `--install-dir` folder that the symlink points to.

```
$ ls -l /usr/local/bin/aws2
lrwxrwxrwx 1 ec2-user ec2-user 49 Oct 22 09:49 /usr/local/bin/aws2 -> /usr/local/aws-cli/
v2/current/bin/aws
```

Now delete the two symlinks in the `--bin-dir` folder. Use `sudo` only if your user account doesn't have write permissions to the relevant folders.

```
$ sudo rm /usr/local/bin/aws2
$ sudo rm /usr/local/bin/aws2_completer
```

Finally, you can delete the `--install-dir` folder:

```
$ sudo rm -rf /usr/local/aws-cli
```

Verifying the Integrity and Authenticity of the Downloaded Files

The AWS CLI version 2 installer package .zip files are cryptographically signed using PGP signatures. You can use the following steps to verify the signatures by using the `GnuPG` tool. If there is any kind of damage or alteration of the files, this verification fails and you should not proceed with installation.

The following example assumes you downloaded the installer package and saved it locally as `awscliv2.zip`. If you named it something else, substitute that name in the following steps.

Steps 1, 2, and 3 are prerequisites, and you need to perform them only once. You should perform steps 4 and 5 every time you download a copy of the installer package.

To validate the files using the PGP signature

1. To download and install the `GnuPG` and its `gpg` command on macOS, we recommend using [Homebrew](#). Install Homebrew using the instructions from their website. Then, run the following command from your macOS terminal.

```
$ brew install gnupg
```

For more information, see the [GnuPG website](#).

2. Create a text file and paste in the following text.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBF2Cr7UBEADJZHcgusOJl7ENSyumXh85z0TRV0xJorM2B/JL0kHOyigQluUG
ZMLhENaG0bYatdrKP+3H91lvK05OpXwnO/R7fB/FSTouki4ciIx5OuLlnJZIXSzx
PqG10mk8xImLNBGwoi6Lto0LYxqHN2iQtzlwTVmq9733zd3XfcXrZ3+Lb1HAgEt5G
TfNxEKJ8soPLyWmwdH6HWCnjZ/aIQRBtIQ05uVeEoYxSh6wOai7ss/KveoSNBbYz
gbdzogI2Y8cgH2nbfgp3DSasaLZEdCSsIsK1u05CinE7k2qZ7KgKAUIcT/cR/grk
C6VwsnDU0UOCideXcQ8WeHutqvgZH1JgKDbznoIzeQHJD238GEu+eKhRHcz8/jeG
94zkcgJ0z3KbZGYMiTh277Fvj9zzvZsbMBCEdV1BTg3TqgvdX4bdkhf5cH+7NtW0
lrFj6UwAsGukBTA0xC0l/dnSmZhJ7Z1KmEWilro/gOrjtOxqRQutlIqG22TaqoPG
fYVN+en3Zwb797kcgZDwqbuykNt64oZWc4XKCa3mprEGC3IbJTBfgglXmZ719yWg
EEUJYOl2XrSuPWml39beWdKM8kzr1Ojnl0m6+lpTRCBfo0wa9F8YZRhHPakwKkX
XDeOGpWRj4ohOx0d2GWkyV5xyN14p2tQOCdODmz80yUTgRpPVQUtOEhXQARAQAB
tCFBV1MgQ0xJIFRlYW0gPGF3cy1jbG1AYW1hem9uLmNvbT6JAlQEEwEIAAD4WIQT7
Xbd/1cEyUAURraimMQRMRnJHXAUCXYKvtQIbAwUJB4TOAAULCQGHAgYVCgkICWIE
FgIDAQIEAQIXGAACRCRmMQRMRnJHXXIXEACHLUIkg8OuPUkGje3jejvQSA1aWUAM
yzy6fdpdlRUz6M6nmsUhoExjVIvibEJpzK5mhuSZ41b0vJ2ZUPgCv4zs2nBd7BGJ
MxKiWgBREgVtdqZ0SzyYH4PYCJSE732x/Fw9hfnh1dMTXNcrQXzwOmmFNNegG0Ox
```

```
au+VnpcR5Kz3smiTrIwZbRudo1ijhCYPQ7t5CMP9kjC6bObvy1hSig2xNbMAN/Do
ikebAl36uA6Y/Uczjj3GxZW4ZWeFirmidKbtqvUz2yOUFszobjiBSqZZHCreC34B
hw9bFNpuWC/OSrXgohdsc6vK50pDGdV5kM2qo9tMQ/izsAwTh/d/GzZv8H4lV9eO
tEis+EpR497PaxKKh9tJfON6Q1YLRHof5xePZtO1lS3gfvS5hXA3HJ9yIxb8TOH
QYmVr3aIUes20i6meI3fuV36VFupwfrTKaL7VXnsrK2fq5cRvyJLNzXucg0WAjPF
RrAGLzY7nP1xeg1a0aeP+pdsgjq1PJom8OCWc1+6DWbg0jsC74WoesAggBItODMB
rsally/q+bPzpsnWjzHV8+1/EtZmSc8ZUGSJOPkfc7hObnfkl18h+1QtKTjZme4d
H17gsBJr+opwJw/Zio2LMjQB0qlm3K1A4zFT7wBC7He6KPQea1p2XAMgtvATtNe
YLZATHZKTJyiqA==
=vYOK
-----END PGP PUBLIC KEY BLOCK-----
```

Here are the details of the public key for reference.

```
Key ID:      A6310ACC4672475C
Type:        RSA
Size:        4096/4096
Created:     2019-09-18
Expires:     2023-09-17
User ID:     AWS CLI Team <aws-cli@amazon.com>
Key fingerprint:  FB5D B77F D5C1 18B8 0511 ADA8 A631 0ACC 4672
```

3. Import the AWS CLI public key with the following command, substituting *public-key-file-name* with whatever you named the file in step 2.

```
$ gpg --import public-key-file-name
gpg: /Users/username/.gnupg/trustdb.gpg: trustdb created
gpg: key A6310ACC4672475C: public key "AWS CLI Team <aws-cli@amazon.com>" imported
gpg: Total number processed: 1
gpg:      imported: 1
```

4. Download the AWS CLI signature file. It has the same path and name as the installer file it corresponds to, but has the extension `.sig`.

```
$ curl -o awscliv2.sig https://d1vvhv12y92vvt.cloudfront.net/awscli-exe-macos.zip.sig
```

5. Verify the signature, passing both the `.sig` and `.zip` file names as parameters to the `gpg` command.

```
$ gpg --verify awscliv2.sig awscliv2.zip
```

The output should look similar to the following.

```
gpg: Signature made Mon Nov  4 19:00:01 2019 PST
gpg:      using RSA key FB5D B77F D5C1 18B8 0511 ADA8 A631 0ACC 4672 475C
gpg: Good signature from "AWS CLI Team <aws-cli@amazon.com>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:      There is no indication that the signature belongs to the owner.
Primary key fingerprint: FB5D B77F D5C1 18B8 0511 ADA8 A631 0ACC 4672 475C
```

Important

The warning in the output is expected and doesn't indicate a problem. It occurs because there isn't a chain of trust between your personal PGP key (if you have one) and the AWS CLI PGP key. For more information, see [Web of trust](#).

Installing AWS CLI version 2 on Windows

This section describes how to install, upgrade, and remove AWS CLI version 2 on Windows.

Preview Evaluation Software

AWS CLI version 2 is provided as a preview for testing and evaluation. At this time, we do not recommend using it in a production environment. For production environments, we recommend that you use the generally available version 1.x.

You can provide feedback for this developer preview version in the [AWS CLI version 2 GitHub repo](#). Be sure to attach the "V2" label to your issue.

Topics

- [Prerequisites for Windows \(p. 14\)](#)
- [Installing on Windows \(p. 14\)](#)
- [Upgrading on Windows \(p. 14\)](#)
- [Removing from Windows \(p. 15\)](#)

Prerequisites for Windows

- The AWS CLI version 2 is supported on Windows XP or later.
- The AWS CLI version 2 supports only 64-bit versions of Windows.

Installing on Windows

For Windows users, the MSI installation package offers a familiar and convenient way to install the AWS CLI version 2 without installing any other prerequisites.

To install the AWS CLI version 2 using the MSI installer

1. [Download the AWS CLI MSI installer for Windows \(64-bit\)](#)
2. Run the downloaded MSI installer and follow the onscreen instructions. By default, the AWS CLI installs to `C:\Program Files\Amazon\AWSCLI2`.

Note

The preview release of AWS CLI version 2 names the program `aws2` to enable AWS CLI version 1 and version 2 to coexist side-by-side. Future releases of AWS CLI version 2 might change this command name.

3. To confirm the installation, use the `aws2 --version` command at a command prompt (open the **Start** menu and search for `cmd` to start a command prompt).

```
C:\> aws2 --version
aws-cli/2.0.0dev0 Python/3.7.3 Windows/10 botocore/2.0.0dev0
```

Don't include the prompt symbol (`C:\>`, shown above) when you type a command. These are included in program listings to differentiate commands that you type from output returned by the AWS CLI. The rest of this guide uses the generic prompt symbol, `$`, except in cases where a command is Windows-specific.

If Windows is unable to find the program, you might need to close and reopen the command prompt to refresh the path, or [add the installation directory to your PATH \(p. 28\)](#) environment variable manually.

Upgrading on Windows

AWS CLI is updated regularly. Check the [Releases page on GitHub](#) to see when the latest version was released.

To update to the latest version, download the latest version of the MSI installer and run it, as described previously. It automatically overwrites the previous version.

Removing from Windows

To uninstall the AWS CLI, open the **Control Panel**, and then choose **Programs and Features**. Select the entry named **AWS Command Line Interface**, and then choose **Uninstall** to launch the uninstaller. Confirm that you want to uninstall the AWS CLI when you're prompted.

You can also launch the **Programs and Features** program from the command line with the following command.

```
C:\> appwiz.cpl
```

Installing the AWS CLI version 1

This topic describes how to install version 1 of the AWS Command Line Interface (AWS CLI). For information about how to install version 2, see [Installing the AWS CLI version 2 \(p. 4\)](#).

You can install the AWS CLI version 1 using any of the following techniques:

- [Using a bundled installer \(p. 15\)](#)
- [Using pip \(p. 15\)](#)
- [Using a virtual environment \(p. 17\)](#)

Prerequisites

- Python 2 version 2.7+ or Python 3 version 3.4+

Important

On January 10th, 2020, AWS CLI version 1.17 and later will no longer support Python 2.6 or Python 3.3. After this date, the installer for the AWS CLI will require Python 2.7, Python 3.4, or a later version to successfully install the AWS CLI. For more information, see [Using the AWS CLI version 1 with Python 2.6 or Python 3.3 \(p. 32\)](#) in this guide, and the [deprecation announcement in this blog post](#).

- Windows, Linux or macOS

You can find the version number of the most recent CLI at: <https://github.com/aws/aws-cli/blob/master/CHANGELOG.rst>.

In this guide, the commands shown assume you have Python v3 installed and the `pip` commands shown use the `pip3` version.

Installing the AWS CLI Using the Bundled Installer

For offline or automated installations on Linux or macOS, we recommend that you try the [bundled installer \(p. 30\)](#). The bundled installer includes the AWS CLI, its dependencies, and a shell script that performs the installation for you.

On Windows, the bundled installer is in the form of an [MSI installer \(p. 26\)](#).

Installing the AWS CLI Using pip

The `pip` package manager for Python provides an easy way to install, upgrade, and remove Python packages and their dependencies.

Installing the current AWS CLI Version

The AWS CLI is updated frequently with support for new services and commands. To determine whether you have the latest version, see the [releases page on GitHub](#).

If you already have `pip` and a supported version of Python, you can install the AWS CLI by using the following command. If you have Python version 3 installed, we recommend that you use the `pip3` command.

```
$ pip3 install awscli --upgrade --user
```

The `--upgrade` option tells `pip3` to upgrade any requirements that are already installed. The `--user` option tells `pip3` to install the program to a subdirectory of your user directory to avoid modifying libraries used by your operating system.

Upgrading to the latest version of the AWS CLI

We recommend that you regularly check to see if there is a new version of the AWS CLI and upgrade to it when you can.

Use the `pip3 list -o` command to check which packages are "outdated".

```
$ aws --version
aws-cli/1.16.273 Python/3.7.3 Linux/4.14.133-113.105.amzn2.x86_64 botocore/1.13.0

$ pip3 list -o
Package      Version Latest Type
-----
awscli       1.16.170 1.16.198 wheel
botocore     1.12.160 1.12.188 wheel
```

Because the previous command shows that there is a newer version of the AWS CLI available, you can run `pip3 install --upgrade` to get the latest version.

```
$ pip3 install --upgrade --user awscli
Collecting awscli
  Downloading https://files.pythonhosted.org/packages/dc/70/
b32e9534c32fe9331801449e1f7eacba6a1992c2e4af9c82ac9116661d3b/awscli-1.16.198-py2.py3-none-
any.whl (1.7MB)
    |#####| 1.7MB 1.6MB/s
Collecting botocore==1.12.188 (from awscli)
  Using cached https://files.pythonhosted.org/packages/10/
cb/8dcfb3e035a419f228df7d3a0eea5d52b528bde7ca162f62f3096a930472/botocore-1.12.188-py2.py3-
none-any.whl
Requirement already satisfied, skipping upgrade: docutils>=0.10 in ./venv/lib/python3.7/
site-packages (from awscli) (0.14)
Requirement already satisfied, skipping upgrade: rsa<=3.5.0,>=3.1.2 in ./venv/lib/
python3.7/site-packages (from awscli) (3.4.2)
Requirement already satisfied, skipping upgrade: colorama<=0.3.9,>=0.2.5 in ./venv/lib/
python3.7/site-packages (from awscli) (0.3.9)
Requirement already satisfied, skipping upgrade: PyYAML<=5.1,>=3.10; python_version !=
"2.6" in ./venv/lib/python3.7/site-packages (from awscli) (3.13)
Requirement already satisfied, skipping upgrade: s3transfer<0.3.0,>=0.2.0 in ./venv/lib/
python3.7/site-packages (from awscli) (0.2.0)
Requirement already satisfied, skipping upgrade: jmespath<1.0.0,>=0.7.1 in ./venv/lib/
python3.7/site-packages (from botocore==1.12.188->awscli) (0.9.4)
Requirement already satisfied, skipping upgrade: urllib3<1.26,>=1.20; python_version >=
"3.4" in ./venv/lib/python3.7/site-packages (from botocore==1.12.188->awscli) (1.24.3)
Requirement already satisfied, skipping upgrade: python-dateutil<3.0.0,>=2.1;
python_version >= "2.7" in ./venv/lib/python3.7/site-packages (from botocore==1.12.188-
>awscli) (2.8.0)
```

```
Requirement already satisfied, skipping upgrade: pyasn1>=0.1.3 in ./venv/lib/python3.7/site-packages (from rsa<=3.5.0,>=3.1.2->awscli) (0.4.5)
Requirement already satisfied, skipping upgrade: six>=1.5 in ./venv/lib/python3.7/site-packages (from python-dateutil<3.0.0,>=2.1; python_version >= "2.7"->botocore==1.12.188->awscli) (1.12.0)
Installing collected packages: botocore, awscli
  Found existing installation: botocore 1.12.160
    Uninstalling botocore-1.12.160:
      Successfully uninstalled botocore-1.12.160
  Found existing installation: awscli 1.16.170
    Uninstalling awscli-1.16.170:
      Successfully uninstalled awscli-1.16.170
Successfully installed awscli-1.16.198 botocore-1.12.188
```

Installing the AWS CLI in a Virtual Environment

If you encounter issues when you attempt to install the AWS CLI with `pip3`, you can [install the AWS CLI in a virtual environment](#) (p. 29) to isolate the tool and its dependencies. Or you can use a different version of Python than you normally do.

Steps to Take after Installation

- [Setting the Path to Include the AWS CLI](#) (p. 17)
- [Configure the AWS CLI with Your Credentials](#) (p. 17)
- [Upgrading to the Latest Version of the AWS CLI](#) (p. 18)
- [Uninstalling the AWS CLI](#) (p. 18)

Setting the Path to Include the AWS CLI

After you install the AWS CLI, you might need to add the path to the executable file to your `PATH` variable. For platform-specific instructions, see the following topics:

- **Linux** – [Add the AWS CLI version 1 Executable to Your Command Line Path](#) (p. 21)
- **Windows** – [Add the AWS CLI version 1 Executable to Your Command Line Path](#) (p. 28)
- **macOS** – [Add the AWS CLI version 1 Executable to Your macOS Command Line Path](#) (p. 25)

Verify that the AWS CLI installed correctly by running `aws --version`.

```
$ aws --version
aws-cli/1.16.273 Python/3.7.3 Linux/4.14.133-113.105.amzn2.x86_64 botocore/1.13.0
```

Configure the AWS CLI with Your Credentials

Before you can run a CLI command, you must configure the AWS CLI with your credentials.

You store credential information locally by defining [profiles](#) (p. 50) in the [AWS CLI configuration files](#) (p. 40), which are stored by default in your user's home directory. For more information, see [Configuring the AWS CLI](#) (p. 37).

Note

If you are running in an Amazon EC2 instance, credentials can be automatically retrieved from the instance metadata. For more information, see [Getting Credentials from EC2 Instance Metadata](#) (p. 61).

Upgrading to the Latest Version of the AWS CLI

The AWS CLI is updated regularly to add support for new services and commands. To update to the latest version of the AWS CLI, run the installation command again. For details about the latest version of the AWS CLI, see the [AWS CLI release notes](#).

```
$ pip3 install awscli --upgrade --user
```

Uninstalling the AWS CLI

If you need to uninstall the AWS CLI, use `pip uninstall`.

```
$ pip3 uninstall awscli
```

If you don't have Python and `pip`, use the procedure for your environment.

Detailed Instructions for Each Environment

- [Install the AWS CLI version 1 on Linux \(p. 18\)](#)
- [Install the AWS CLI version 1 on macOS \(p. 23\)](#)
- [Install the AWS CLI version 1 on Windows \(p. 26\)](#)
- [Install the AWS CLI version 1 in a Virtual Environment \(p. 29\)](#)
- [Install the AWS CLI version 1 Using the Bundled Installer \(Linux or macOS\) \(p. 30\)](#)
- [Using the AWS CLI version 1 with Python 2.6 or Python 3.3 \(p. 32\)](#)

Install the AWS CLI version 1 on Linux

Important

On January 10th, 2020, AWS CLI version 1.17 and later will no longer support Python 2.6 or Python 3.3. After this date, the installer for the AWS CLI will require Python 2.7, Python 3.4, or a later version to successfully install the AWS CLI. For more information, see [Using the AWS CLI version 1 with Python 2.6 or Python 3.3 \(p. 32\)](#) in this guide, and the [deprecation announcement in this blog post](#).

You can install version 1 of the AWS Command Line Interface (AWS CLI) and its dependencies on most Linux distributions by using `pip`, a package manager for Python.

Although the `awscli` package is available in repositories for other package managers such as `apt` and `yum`, these are not produced or managed by AWS and are therefore not official and not supported by AWS. We recommend that you install the AWS CLI from only the official AWS distribution points, as documented in this guide.

If you already have `pip`, follow the instructions in the main [installation topic \(p. 4\)](#). Run `pip --version` to see if your version of Linux already includes Python and `pip`. We recommend that if you have Python version 3+ installed, you use the `pip3` command.

```
$ pip3 --version
```

If you don't already have `pip` installed, check which version of Python is installed.

```
$ python --version
```

or

```
$ python3 --version
```

If you don't already have Python 2 version 2.7+ or Python 3 version 3.4+, you must first [install Python](#) (p. 22). If you do have Python installed, proceed to installing `pip` and the AWS CLI.

Sections

- [Install pip](#) (p. 19)
- [Install the AWS CLI version 1 with pip](#) (p. 20)
- [Upgrading to the Latest Version of the AWS CLI version 1](#) (p. 20)
- [Add the AWS CLI version 1 Executable to Your Command Line Path](#) (p. 21)
- [Installing Python on Linux](#) (p. 22)
- [Install the AWS CLI version 1 on Amazon Linux](#) (p. 22)

Install pip

If you don't already have `pip` installed, you can install it by using the script that the *Python Packaging Authority* provides.

To install pip

1. Use the `curl` command to download the installation script. The following command uses the `-O` (uppercase "O") parameter to specify that the downloaded file is to be stored in the current folder using the same name it has on the remote host.

```
$ curl -O https://bootstrap.pypa.io/get-pip.py
```

2. Run the script with Python to download and install the latest version of `pip` and other required support packages.

```
$ python get-pip.py --user
```

Or use the following.

```
$ python3 get-pip.py --user
```

When you include the `--user` switch, the script installs `pip` to the path `~/.local/bin`.

3. Ensure the folder that contains `pip` is part of your `PATH` variable.
 - a. Find your shell's profile script in your user folder. If you're not sure which shell you have, run `echo $SHELL`.

```
$ ls -a ~  
.  .. .bash_logout .bash_profile .bashrc Desktop Documents Downloads
```

- **Bash** – `.bash_profile`, `.profile`, or `.bash_login`
- **Zsh** – `.zshrc`
- **Tcsh** – `.tcshrc`, `.cshrc` or `.login`

- b. Add an export command at the end of your profile script that's similar to the following example.

```
export PATH=~/local/bin:$PATH
```


This command inserts the path, `~/ .local/bin` in this example, at the front of the existing `PATH` variable.

- c. Reload the profile into your current session to put those changes into effect.

```
$ source ~/.bash_profile
```

4. Now you can test to verify that `pip` is installed correctly.

```
$ pip3 --version
pip 19.2.3 from ~/.local/lib/python3.7/site-packages (python 3.7)
```

Install the AWS CLI version 1 with pip

Use `pip` to install the AWS CLI.

```
$ pip3 install awscli --upgrade --user
```

When you use the `--user` switch, `pip` installs the AWS CLI to `~/ .local/bin`.

Verify that the AWS CLI installed correctly.

```
$ aws --version
aws-cli/1.16.273 Python/3.7.3 Linux/4.14.133-113.105.amzn2.x86_64 botocore/1.13.0
```

If you get an error, see [Troubleshooting AWS CLI Errors \(p. 133\)](#).

Upgrading to the Latest Version of the AWS CLI version 1

We recommend that you regularly check to see if there is a new version of the AWS CLI and upgrade to it when you can.

Use the `pip list -o` command to check which packages are "outdated".

```
$ aws --version
aws-cli/1.16.170 Python/3.7.3 Linux/4.14.123-111.109.amzn2.x86_64 botocore/1.12.160

$ pip3 list -o
Package      Version Latest Type
-----
awscli       1.16.170 1.16.198 wheel
botocore     1.12.160 1.12.188 wheel
```

Because the previous command shows that there is a newer version of the AWS CLI version 1 available, you can run `pip install --upgrade` to get the latest version.

```
$ pip3 install --upgrade --user awscli
Collecting awscli
  Downloading https://files.pythonhosted.org/packages/dc/70/
b32e9534c32fe9331801449e1f7eacba6a1992c2e4af9c82ac9116661d3b/awscli-1.16.198-py2.py3-none-
any.whl (1.7MB)
    |#####| 1.7MB 1.6MB/s
Collecting botocore==1.12.188 (from awscli)
  Using cached https://files.pythonhosted.org/packages/10/
cb/8dcfb3e035a419f228df7d3a0eea5d52b528bde7ca162f62f3096a930472/botocore-1.12.188-py2.py3-
none-any.whl
```

```
Requirement already satisfied, skipping upgrade: docutils>=0.10 in ./venv/lib/python3.7/
site-packages (from awscli) (0.14)
Requirement already satisfied, skipping upgrade: rsa<=3.5.0,>=3.1.2 in ./venv/lib/
python3.7/site-packages (from awscli) (3.4.2)
Requirement already satisfied, skipping upgrade: colorama<=0.3.9,>=0.2.5 in ./venv/lib/
python3.7/site-packages (from awscli) (0.3.9)
Requirement already satisfied, skipping upgrade: PyYAML<=5.1,>=3.10; python_version !=
"2.6" in ./venv/lib/python3.7/site-packages (from awscli) (3.13)
Requirement already satisfied, skipping upgrade: s3transfer<0.3.0,>=0.2.0 in ./venv/lib/
python3.7/site-packages (from awscli) (0.2.0)
Requirement already satisfied, skipping upgrade: jmespath<1.0.0,>=0.7.1 in ./venv/lib/
python3.7/site-packages (from botocore==1.12.188->awscli) (0.9.4)
Requirement already satisfied, skipping upgrade: urllib3<1.26,>=1.20; python_version >=
"3.4" in ./venv/lib/python3.7/site-packages (from botocore==1.12.188->awscli) (1.24.3)
Requirement already satisfied, skipping upgrade: python-dateutil<3.0.0,>=2.1;
python_version >= "2.7" in ./venv/lib/python3.7/site-packages (from botocore==1.12.188-
>awscli) (2.8.0)
Requirement already satisfied, skipping upgrade: pyasn1>=0.1.3 in ./venv/lib/python3.7/
site-packages (from rsa<=3.5.0,>=3.1.2->awscli) (0.4.5)
Requirement already satisfied, skipping upgrade: six>=1.5 in ./venv/lib/python3.7/site-
packages (from python-dateutil<3.0.0,>=2.1; python_version >= "2.7"->botocore==1.12.188-
>awscli) (1.12.0)
Installing collected packages: botocore, awscli
  Found existing installation: botocore 1.12.160
  Uninstalling botocore-1.12.160:
    Successfully uninstalled botocore-1.12.160
  Found existing installation: awscli 1.16.170
  Uninstalling awscli-1.16.170:
    Successfully uninstalled awscli-1.16.170
Successfully installed awscli-1.16.198 botocore-1.12.188
```

Add the AWS CLI version 1 Executable to Your Command Line Path

After installing with pip, you might need to add the aws executable to your operating system's PATH environment variable.

You can verify which folder pip installed the AWS CLI in by running the following command.

```
$ which aws
/home/username/.local/bin/aws
```

You can reference this as ~/.local/bin/ because /home/username corresponds to ~ in Linux.

If you omitted the --user switch and so didn't install in user mode, the executable might be in the bin folder of your Python installation. If you don't know where Python is installed, run this command.

```
$ which python
/usr/local/bin/python
```

The output might be the path to a symlink, not to the actual executable. Run `ls -al` to see where it points.

```
$ ls -al /usr/local/bin/python
/usr/local/bin/python -> ~/.local/Python/3.6/bin/python3.6
```

If this is the same folder you added to the path in step 3 in [Install pip \(p. 19\)](#), you're done. Otherwise, perform those same steps 3a–3c again, adding this folder to the path.

Installing Python on Linux

If your distribution didn't come with Python, or came with an earlier version, install Python before installing `pip` and the AWS CLI.

To install Python 3 on Linux

1. See if Python is already installed.

```
$ python --version
```

Or use the following.

```
$ python3 --version
```

Note

If your Linux distribution came with Python, you might need to install the Python developer package to get the headers and libraries required to compile extensions, and install the AWS CLI. Use your package manager to install the developer package (typically named `python-dev` or `python-devel`).

2. If Python 2 version 2.7+ or Python 3 version 3.4+ or later is not installed, install Python with your distribution's package manager. The command and package name varies:

- On Debian derivatives such as Ubuntu, use `apt`. Check the `apt` repository for the versions of Python available to you. Then, run a command similar to the following, substituting the correct package name.

```
$ sudo apt-get install python3
```

- On Red Hat and derivatives, use `yum`. Check the `yum` repository for the versions of Python available to you. Then, run a command similar to the following, substituting the correct package name.

```
$ sudo yum install python36
```

- On SUSE and derivatives, use `zypper`. Check the repository for the versions of Python available to you. Then, run a command similar to the following, substituting the correct package name.

```
$ sudo zypper install python3
```

See the documentation for your system's package manager and for [Python](#) for more information about where it is installed and how to use it.

3. Open a command prompt or shell and run the following command to verify that Python installed correctly.

```
$ python3 --version
Python 3.7.3
```

Install the AWS CLI version 1 on Amazon Linux

The AWS Command Line Interface version 1 (AWS CLI version 1) comes preinstalled on Amazon Linux and Amazon Linux 2. Check the currently installed version by using the following command.

```
$ aws --version
aws-cli/1.16.273 Python/3.7.3 Linux/4.14.133-113.105.amzn2.x86_64 botocore/1.13.0
```

Important

Using `sudo` to complete a command grants the command full access to your system. We recommend using that command only when no more secure option exists. For commands like `pip`, we recommend that you avoid using `sudo` by using a [Python virtual environment \(venv\)](#) or by specifying the `--user` option to install in the user's folders instead of the system's folders.

If you use the `yum` package manager, you can install the AWS CLI with the command `yum install aws-cli`. You can use the command `yum update` to get the latest version available in the `yum` repository.

Note

The `yum` repository is not owned or maintained by Amazon and might not contain the latest version. Instead, we recommend that you use `pip` to get the latest version.

Prerequisites

Verify that Python and `pip` are already installed. For more information, see [Install the AWS CLI version 1 on Linux \(p. 18\)](#).

To install or upgrade the AWS CLI version 1 on Amazon Linux (user)

1. Use `pip3 install` to install the latest version of the AWS CLI version 1. We recommend that if you have Python version 3+ installed that you use `pip3`. If you run the command from within a [Python virtual environment \(venv\)](#), then you don't need to use the `--user` option.

```
$ pip3 install --upgrade --user awscli
```

2. Add the install location to the beginning of your `PATH` variable.

```
$ export PATH=/home/ec2-user/.local/bin:$PATH
```

Add this command to the end of your profile's startup script (for example, `~/.bashrc`) to persist the change between command line sessions.

3. Verify that you're running new version with `aws --version`.

```
$ aws --version
aws-cli/1.16.273 Python/3.7.3 Linux/4.14.133-113.105.amzn2.x86_64 botocore/1.13.0
```

Install the AWS CLI version 1 on macOS

The recommended way to install version 1 of the AWS Command Line Interface (AWS CLI) on macOS is to use the bundled installer. The bundled installer includes all dependencies and you can use it offline.

Important

On January 10th, 2020, AWS CLI version 1.17 and later will no longer support Python 2.6 or Python 3.3. After this date, the installer for the AWS CLI will require Python 2.7, Python 3.4, or a later version to successfully install the AWS CLI. For more information, see [Using the AWS CLI version 1 with Python 2.6 or Python 3.3 \(p. 32\)](#) in this guide, and the [deprecation announcement in this blog post](#).

Important

The bundled installer doesn't support installing to paths that contain spaces.

Sections

- [Prerequisites \(p. 24\)](#)
- [Install the AWS CLI version 1 Using the Bundled Installer \(p. 24\)](#)
- [Install the AWS CLI version 1 on macOS Using pip \(p. 25\)](#)
- [Add the AWS CLI version 1 Executable to Your macOS Command Line Path \(p. 25\)](#)

Prerequisites

- Python 2 version 2.7+ or Python 3 version 3.4+

Check your Python installation.

```
$ python --version
```

If your computer doesn't already have Python installed, or if you want to install a different version of Python, follow the procedure in [Install the AWS CLI version 1 on Linux \(p. 18\)](#).

Install the AWS CLI version 1 Using the Bundled Installer

Follow these steps from the command line to install the AWS CLI version 1 using the bundled installer.

To install the AWS CLI version 1 using the bundled installer

1. Here are the steps described below in one easy to copy-and-paste group. See the descriptions of each line in the steps that follow.

```
curl "https://s3.amazonaws.com/aws-cli/awscli-bundle.zip" -o "awscli-bundle.zip"
unzip awscli-bundle.zip
sudo ./awscli-bundle/install -i /usr/local/aws -b /usr/local/bin/aws
```

Note

If you don't have unzip, use your favorite package manager or an equivalent to install it.

2. Run the install program. This command installs the AWS CLI to `/usr/local/aws` and creates the symlink `aws` in the `/usr/local/bin` directory. Using the `-b` option to create a symlink eliminates the need to specify the install directory in the user's `$PATH` variable. This should enable all users to call the AWS CLI by typing `aws` from any directory.

```
$ sudo ./awscli-bundle/install -i /usr/local/aws -b /usr/local/bin/aws
```

Note

By default, the install script runs under the system's default version of Python. If you have installed an alternative version of Python and want to use that to install the AWS CLI, run the install script and specify that version by including the absolute path to the Python application, as shown in the following example.

```
$ sudo /usr/local/bin/python3.7 awscli-bundle/install -i /usr/local/aws -b /usr/local/bin/aws
```

To see an explanation of the `-i` and `-b` options, use the `-h` option.

```
$ ./awscli-bundle/install -h
```

Install the AWS CLI version 1 on macOS Using pip

You can also use `pip` directly to install the AWS CLI. If you don't have `pip`, follow the instructions in the main [installation topic \(p. 4\)](#). Run `pip3 --version` to see if your version of macOS already includes Python and `pip3`.

```
$ pip3 --version
```

To install the AWS CLI on macOS

1. Download and install the latest version of Python from the [downloads page](#) of [Python.org](#).
2. Download and run the `pip3` installation script provided by the Python Packaging Authority.

```
$ curl -O https://bootstrap.pypa.io/get-pip.py  
$ python3 get-pip.py --user
```

3. Use your newly installed `pip3` to install the AWS CLI. We recommend that if you use Python version 3+, that you use the `pip3` command.

```
$ pip3 install awscli --upgrade --user
```

4. Verify that the AWS CLI is installed correctly.

```
$ aws --version  
AWS CLI 1.16.273 (Python 3.7.3)
```

If the program isn't found, [add it to your command line path \(p. 25\)](#).

To upgrade to the latest version, run the installation command again.

```
$ pip3 install awscli --upgrade --user
```

Add the AWS CLI version 1 Executable to Your macOS Command Line Path

After installing with `pip`, you might need to add the `aws` program to your operating system's `PATH` environment variable. The location of the program depends on where Python is installed.

Example AWS CLI install location - macOS with Python 3.6 and `pip` (user mode)

```
~/Library/Python/3.7/bin
```

Substitute the version of Python that you have for the version in the example above.

If you don't know where Python is installed, run `which python`.

```
$ which python  
/usr/local/bin/python
```

The output might be the path to a symlink, not the actual program. Run `ls -al` to see where it points.

```
$ ls -al /usr/local/bin/python  
~/Library/Python/3.7/bin/python3.6
```

`pip` installs programs in the same folder that contains the Python application. Add this folder to your `PATH` variable.

To modify your `PATH` variable (Linux or macOS)

1. Find your shell's profile script in your user folder. If you're not sure which shell you have, run `echo $SHELL`.

```
$ ls -a ~  
.  ..  .bash_logout  .bash_profile  .bashrc  Desktop  Documents  Downloads
```

- **Bash** – `.bash_profile`, `.profile`, or `.bash_login`
- **Zsh** – `.zshrc`
- **Tcsh** – `.tcshrc`, `.cshrc`, or `.login`

2. Add an export command to your profile script.

```
export PATH=~/local/bin:$PATH
```

This command adds a path, `~/local/bin` in this example, to the current `PATH` variable.

3. Load the updated profile into your current session.

```
$ source ~/.bash_profile
```

Install the AWS CLI version 1 on Windows

You can install version 1 of the AWS Command Line Interface (AWS CLI) on Windows by using a standalone installer or `pip`, which is a package manager for Python. If you already have `pip`, follow the instructions in the main [installation topic \(p. 4\)](#).

Important

On January 10th, 2020, AWS CLI version 1.17 and later will no longer support Python 2.6 or Python 3.3. After this date, the installer for the AWS CLI will require Python 2.7, Python 3.4, or a later version to successfully install the AWS CLI. For more information, see [Using the AWS CLI version 1 with Python 2.6 or Python 3.3 \(p. 32\)](#) in this guide, and the [deprecation announcement in this blog post](#).

Sections

- [Install the AWS CLI version 1 Using the MSI Installer \(p. 26\)](#)
- [Install the AWS CLI version 1 Using Python and pip on Windows \(p. 27\)](#)
- [Add the AWS CLI version 1 Executable to Your Command Line Path \(p. 28\)](#)

Install the AWS CLI version 1 Using the MSI Installer

The AWS CLI version 1 is supported on Windows XP or later. For Windows users, the MSI installation package offers a familiar and convenient way to install the AWS CLI version 1 without installing any other prerequisites.

When updates are released, you must repeat the installation process to get the latest version of the AWS CLI version 1.

To install the AWS CLI version 1 using the MSI installer

1. Download the appropriate MSI installer.

- [Download the AWS CLI MSI installer for Windows \(64-bit\)](#)
- [Download the AWS CLI MSI installer for Windows \(32-bit\)](#)
- [Download the AWS CLI setup file](#) (includes both the 32-bit and 64-bit MSI installers and will automatically install the correct version)

Note

The MSI installer for the AWS CLI version 1 doesn't work with Windows Server 2008 (version 6.0.6002). Use [pip \(p. 27\)](#) to install with this version of Windows Server.

2. Run the downloaded MSI installer or the setup file.
3. Follow the onscreen instructions.

By default, the AWS CLI version 1 installs to `C:\Program Files\Amazon\AWSCLI` (64-bit version) or `C:\Program Files (x86)\Amazon\AWSCLI` (32-bit version). To confirm the installation, use the `aws --version` command at a command prompt (open the **Start** menu and search for `cmd` to start a command prompt).

```
C:\> aws --version
aws-cli/1.16.273 Python/3.7.3 Windows/10 botocore/1.13.0
```

Don't include the prompt symbol (`C:\>`, shown above) when you type a command. These are included in program listings to differentiate commands that you type from output returned by the CLI. The rest of this guide uses the generic prompt symbol, `$`, except in cases where a command is Windows-specific.

If Windows is unable to find the program, you might need to close and reopen the command prompt to refresh the path, or [add the installation directory to your PATH \(p. 28\)](#) environment variable manually.

Updating an MSI Installation

The AWS CLI version 1 is updated regularly. Check the [Releases](#) page on GitHub to see when the latest version was released. To update to the latest version, download and run the MSI installer again, as described previously.

Uninstalling the AWS CLI version 1

To uninstall the AWS CLI version 1, open the **Control Panel**, and then choose **Programs and Features**. Select the entry named **AWS Command Line Interface**, and then choose **Uninstall** to launch the uninstaller. Confirm that you want to uninstall the AWS CLI when you're prompted.

You can also launch the **Programs and Features** program from the command line with the following command.

```
C:\> appwiz.cpl
```

Install the AWS CLI version 1 Using Python and `pip` on Windows

The Python Software Foundation provides installers for Windows that include `pip`.

To install Python and `pip` (Windows)

1. Download the Python Windows x86-64 installer from the [downloads page](#) of [Python.org](#).
2. Run the installer.
3. Choose **Add Python 3 to PATH**.
4. Choose **Install Now**.

The installer installs Python in your user folder and adds its program folders to your user path.

To install the AWS CLI version 1 with pip3 (Windows)

If you use Python version 3+, we recommend that you use the pip3 command.

1. Open the **Command Prompt** from the **Start** menu.
2. Use the following commands to verify that Python and pip are both installed correctly.

```
C:\> python --version
Python 3.7.1
C:\> pip3 --version
pip 19.2.3 from c:\program files\python37\lib\site-packages\pip (python 3.7)
```

3. Install the AWS CLI version 1 using pip.

```
C:\> pip3 install awscli
```

4. Verify that the AWS CLI version 1 is installed correctly.

```
C:\> aws --version
aws-cli/1.16.273 Python/3.7.3 Windows/10 botocore/1.13.0
```

To upgrade to the latest version, run the installation command again.

```
C:\> pip3 install --user --upgrade awscli
```

Add the AWS CLI version 1 Executable to Your Command Line Path

After installing the AWS CLI version 1 with pip, add the aws program to your operating system's `PATH` environment variable. With an MSI installation, this should happen automatically, but you might need to set it manually if the aws command doesn't run after you install it.

If this command returns a response, then you should be ready to run the tool. The `where` command, by default, shows where in the system `PATH` it found the specified program.

```
C:\> where aws
C:\Program Files\Amazon\AWSCLI\bin\aws.exe
```

You can find where the aws program is installed by running the following command.

```
C:\> where c:\ aws
C:\Program Files\Python37\Scripts\aws
```

If the `where` command returns the following error, it's not in the system `PATH` and you can't run it by simply typing its name.

```
C:\> where c:\ aws
INFO: Could not find files for the given pattern(s).
```

In that case, run the `where` command with the `/R` *path* parameter to tell it to search all folders, and then add the path manually. Use the command line or File Explorer to discover where it is installed on your computer.

```
C:\> where /R c:\ aws
c:\Program Files\Amazon\AWSCLI\bin\aws.exe
c:\Program Files\Amazon\AWSCLI\bincompat\aws.cmd
c:\Program Files\Amazon\AWSCLI\runtime\Scripts\aws
c:\Program Files\Amazon\AWSCLI\runtime\Scripts\aws.cmd
...
```

The paths that show up depend on your platform and which method you used to install the AWS CLI.

Typical paths include:

- **Python 3 and pip3** – C:\Program Files\Python37\Scripts\
- **Python 3 and pip3 --user option on earlier versions of Windows** – %USERPROFILE%\AppData\Local\Programs\Python\Python37\Scripts
- **Python 3 and pip3 --user option on Windows 10** – %USERPROFILE%\AppData\Roaming\Python\Python37\Scripts
- **MSI installer (64-bit)** – C:\Program Files\Amazon\AWSCLI\bin
- **MSI installer (32-bit)** – C:\Program Files (x86)\Amazon\AWSCLI\bin

Note

Folder names that include version numbers can vary. The examples above reflect the use of Python version 3.7. Replace as needed with the version number you are using.

To modify your PATH variable (Windows)

1. Press the Windows key and enter **environment variables**.
2. Choose **Edit environment variables for your account**.
3. Choose **PATH**, and then choose **Edit**.
4. Add the path to the **Variable value** field. For example: **C:\new\path**
5. Choose **OK** twice to apply the new settings.
6. Close any running command prompts and reopen the command prompt window.

Install the AWS CLI version 1 in a Virtual Environment

You can avoid requirement version conflicts with other pip packages by installing version 1 of the AWS Command Line Interface (AWS CLI) in a virtual environment.

Important

On January 10th, 2020, AWS CLI version 1.17 and later will no longer support Python 2.6 or Python 3.3. After this date, the installer for the AWS CLI will require Python 2.7, Python 3.4, or a later version to successfully install the AWS CLI. For more information, see [Using the AWS CLI version 1 with Python 2.6 or Python 3.3 \(p. 32\)](#) in this guide, and the [deprecation announcement in this blog post](#).

To install the AWS CLI version 1 in a virtual environment

1. Install virtualenv using pip.

```
$ pip install --user virtualenv
```

2. Create a virtual environment and name it.

```
$ virtualenv ~/cli-ve
```

Alternatively, you can use the `-p` option to specify a version of Python other than the default.

```
$ virtualenv -p /usr/bin/python37 ~/cli-ve
```

3. Activate your new virtual environment.

Linux or macOS

```
$ source ~/cli-ve/bin/activate
```

Windows

```
$ %USERPROFILE%\cli-ve\Scripts\activate
```

The prompt changes to show that your virtual environment is active.

```
(cli-ve)-$
```

4. Install the AWS CLI version 1 into your virtual environment.

```
(cli-ve)-$ pip install --upgrade awscli
```

5. Verify that the AWS CLI version 1 is installed correctly.

```
$ aws --version  
aws-cli/1.16.273 Python/3.7.3 Linux/4.14.133-113.105.amzn2.x86_64 botocore/1.13.0
```

You can use the `deactivate` command to exit the virtual environment. Whenever you start a new session, you must reactivate the environment.

To upgrade to the latest version, run the installation command again.

```
(cli-ve)-$ pip install --upgrade awscli
```

Install the AWS CLI version 1 Using the Bundled Installer (Linux or macOS)

On Linux or macOS, you can use the bundled installer to install version 1 of the AWS Command Line Interface (AWS CLI). The bundled installer includes all dependencies and can be used offline.

The bundled installer doesn't support installing to paths that contain spaces.

Important

On January 10th, 2020, AWS CLI version 1.17 and later will no longer support Python 2.6 or Python 3.3. After this date, the installer for the AWS CLI will require Python 2.7, Python 3.4, or a later version to successfully install the AWS CLI. For more information, see [Using the AWS CLI version 1 with Python 2.6 or Python 3.3 \(p. 32\)](#) in this guide, and the [deprecation announcement in this blog post](#).

Sections

- [Prerequisites \(p. 31\)](#)
- [Install the AWS CLI version 1 Using the Bundled Installer \(p. 31\)](#)

- [Install the AWS CLI version 1 without Sudo \(Linux or macOS\) \(p. 32\)](#)
- [Uninstall the AWS CLI version 1 \(p. 32\)](#)

Prerequisites

- Linux or macOS
- Python 2 version 2.7+ or Python 3 version 3.4+

Check your Python installation.

```
$ python --version
```

If your computer doesn't already have Python installed, or you would like to install a different version of Python, follow the procedure in [Install the AWS CLI version 1 on Linux \(p. 18\)](#).

Install the AWS CLI version 1 Using the Bundled Installer

The following steps enable you to install the AWS CLI version 1 from the command line on any build of Linux or macOS.

To download it directly (without using `curl`), use this link:

- <https://s3.amazonaws.com/aws-cli/awscli-bundle.zip>

The following is a summary of the installation commands explained below that you can cut and paste to run as a single set of commands.

```
curl "https://s3.amazonaws.com/aws-cli/awscli-bundle.zip" -o "awscli-bundle.zip"
unzip awscli-bundle.zip
sudo ./awscli-bundle/install -i /usr/local/aws -b /usr/local/bin/aws
```

Follow these steps from the command line to install the AWS CLI version 1 using the bundled installer.

To install the AWS CLI version 1 using the bundled installer

1. Download the AWS CLI version 1 bundled installer using the following command.

```
$ curl "https://s3.amazonaws.com/aws-cli/awscli-bundle.zip" -o "awscli-bundle.zip"
```

2. Extract the files from the package.

```
$ unzip awscli-bundle.zip
```

Note

If you don't have `unzip`, use your Linux distribution's built-in package manager to install it.

3. Run the install program.

```
$ sudo ./awscli-bundle/install -i /usr/local/aws -b /usr/local/bin/aws
```

Note

By default, the install script runs under the system default version of Python. If you have installed an alternative version of Python and want to use that to install the AWS CLI, run the install script with that version by absolute path to the Python executable, as follows.

```
$ sudo /usr/local/bin/python3.7 awscli-bundle/install -i /usr/local/aws -b /usr/local/bin/aws
```

The installer installs the AWS CLI at `/usr/local/aws` and creates the symlink `aws` at the `/usr/local/bin` directory. Using the `-b` option to create a symlink eliminates the need to specify the install directory in the user's `$PATH` variable. This should enable all users to call the AWS CLI by typing `aws` from any directory.

To see an explanation of the `-i` and `-b` options, use the `-h` option.

```
$ ./awscli-bundle/install -h
```

Install the AWS CLI version 1 without Sudo (Linux or macOS)

If you don't have `sudo` permissions or want to install the AWS CLI only for the current user, you can use a modified version of the previous commands. The first two commands are the same. The last command uses the `-b` parameter to specify the folder where the installer places the `aws` symlink file. You must have write permissions to the specified folder.

```
$ curl "https://s3.amazonaws.com/aws-cli/awscli-bundle.zip" -o "awscli-bundle.zip"
$ unzip awscli-bundle.zip
$ ./awscli-bundle/install -b ~/bin/aws
```

This installs the AWS CLI to the default location (`~/local/lib/aws`) and creates a symbolic link (symlink) at `~/bin/aws`. Make sure that `~/bin` is in your `PATH` environment variable for the symlink to work.

```
$ echo $PATH | grep ~/bin      // See if $PATH contains ~/bin (output will be empty if it
                              // doesn't)
$ export PATH=~/bin:$PATH     // Add ~/bin to $PATH if necessary
```

Tip

To ensure that your `$PATH` settings are retained between sessions, add the `export` line to your shell profile (`~/.profile`, `~/.bash_profile`, and so on).

Uninstall the AWS CLI version 1

The bundled installer doesn't put anything outside of the installation directory except the optional symlink, so uninstalling is as simple as deleting those two items.

```
$ sudo rm -rf /usr/local/aws
$ sudo rm /usr/local/bin/aws
```

Using the AWS CLI version 1 with Python 2.6 or Python 3.3

Important

On January 10th, 2020, AWS CLI version 1.17 and later will no longer support Python 2.6 or Python 3.3. After this date, the installer for the AWS CLI will require Python 2.7, Python 3.4, or a later version to successfully install the AWS CLI. For more information, see [Using the](#)

[AWS CLI version 1 with Python 2.6 or Python 3.3 \(p. 32\)](#) in this guide, and the [deprecation announcement in this blog post](#).

AWS CLI version 1 requires that you have a version of Python installed. This Python installation can be any supported version of Python. Because Python 2.6 and Python 3.3 are no longer supported and are no longer receiving security updates, we are deprecating support for Python 2.6 and Python 3.3 with the AWS CLI version 1. We strongly recommend that you upgrade to a later version of Python.

Starting on January 10th, 2020, AWS CLI releases version 1.17 and later will work only with later versions of Python.

To continue using Python 2.6 or Python 3.3 with the AWS CLI version 1, you must "pin" your current installation to an older version of the AWS CLI version 1, as described in the following sections. Pinning your current version prevents it from being updated to a later version.

Note

Using an earlier version of the AWS CLI version 1 prevents you from accessing new services or features that were added to the AWS CLI after the date that your earlier version was initially released. We recommend that whenever possible, you upgrade your Python version to a supported version, and use a later version of the AWS CLI version 1.

Pip

You can force `pip` to download an AWS CLI version 1 version that is compatible with Python 2.6 or Python 3.3 by using a command that specifies `awscli<1.17`, similar to the following example.

```
$ pip3 install --upgrade --user awscli<1.17
```

If you install the AWS CLI version 1 using a [pip Requirements file](#), include a line similar to the following.

```
awscli<1.17
```

Bundled installer on Linux or macOS

You can pin your installation to a specific earlier version with the bundled installer. To do this, download and save a copy of the bundled installer that includes a version of the AWS CLI version 1 that is compatible with the version of Python that you want to use. You can use the following URL format to download the file, replacing `{VERSION}` with the version number that you want to use, as shown. Version numbers less than 1.17 support the earlier Python releases.

```
https://s3.amazonaws.com/aws-cli/awscli-bundle-{VERSION}.zip
```

For example, the following command downloads the AWS CLI version 1.16.188.

```
$ curl https://s3.amazonaws.com/aws-cli/awscli-bundle-1.16.188.zip -o awscli-bundle.zip
```

From here, you can continue to follow the installation instructions in [Install the AWS CLI version 1 Using the Bundled Installer \(Linux or macOS\) \(p. 30\)](#), starting with step 2.

MSI installer on Windows

The MSI installer version of the AWS CLI version 1 for Windows is not affected by this deprecation. This version of the AWS CLI version 1 installation includes and uses its own embedded copy of Python, independent of any other Python version you might have installed. If you're using an MSI installer-based AWS CLI, no changes are required.

Migrating from AWS CLI version 1 to version 2

This topic describes the changes in behavior between AWS CLI version 1 and AWS CLI version 2 that might require you to make changes to scripts or commands to get the same behavior in version 2 as you did in version 1.

AWS CLI version 2 no longer automatically retrieves `http://` or `https://` URLs for parameters

The AWS CLI version 2 no longer performs a GET operation when a parameter value begins with `http://` or `https://`, and then using the returned content as the value of the parameter. If you need to retrieve a URL and pass the contents read from that URL as the value of a parameter, we recommend that you use `curl` or a similar tool to download the contents of the URL to a local file. Then use the `file://` syntax to read the contents of that file and use it as the parameter's value.

For example, the following command no longer tries to retrieve the contents of the page found at `http://www.google.com` and pass those contents as the parameter. Instead, it passes the literal text string `https://google.com` as the parameter.

```
$ aws2 ssm put-parameter --value http://www.google.com --name prod.microservice1.db.secret --type String 2
```

If you really do want to retrieve and use the contents of a web URL as a parameter, you can do the following in version 2.

```
$ curl https://my.example.com/mypolicyfile.json -o mypolicyfile.json
$ aws iam put-role-policy --policy-document file:///mypolicyfile.json --role-name MyRole --policy-name MyReadOnlyPolicy
```

In the previous example, the `-o` parameter tells `curl` to save the file in the current folder with the same name as the source file. The second command retrieves the content of that downloaded file and passes the content as the value of `--policy-document`.

AWS CLI version 2 now returns all timestamp output values in ISO 8601 format

By default, AWS CLI version 2 returns all timestamp response values in the [ISO 8601 format](#). In AWS CLI version 1, commands returned timestamp values in whatever format was returned by the HTTP API response, which could vary from service to service.

ISO 8601 formatted timestamps look like the following examples. The first example shows the time in [Coordinated Universal Time \(UTC\)](#) by including a `Z` after the time. The date and the time are separated by a `T`.

```
2019-10-31T22:21:41Z
```

To specify a different time zone, instead of the `Z`, specify a `+` or `-` and the number of hours the desired time zone is ahead of or behind UTC, as a two-digit value. The following example shows the same time as the previous example but adjusted to Pacific Standard time, which is eight hours behind UTC.

```
2019-10-31T14:21:41-08
```

To see timestamps in the format returned by the HTTP API response, add the following line to your `.aws/config` profile.

```
cli_timestamp_format = wire
```

AWS CLI version 2 no longer supports "hidden" aliases

AWS CLI version 2 no longer supports the following hidden aliases that were supported in version 1.

In the following table, the first column displays the service, command, and parameter that work in all versions, including AWS CLI version 2. The second column displays the alias that no longer works in AWS CLI version 2

Working Service, Command, and Parameter	Obsolete Alias
cognito-identity create-identity-pool open-id-connect-provider-arns	open-id-connect-provider-ar-ns
storagegateway describe-tapes tape-arns	tape-ar-ns
storagegateway.describe-tape-archives.tape-arns	tape-ar-ns
storagegateway.describe-vtl-devices.vtl-device-arns	vtl-device-ar-ns
storagegateway.describe-cached-iscsi-volumes.volume-arns	volume-ar-ns
storagegateway.describe-stored-iscsi-volumes.volume-arns	volume-ar-ns
route53domains.view-billing.start-time	start
deploy.create-deployment-group.ec2-tag-set	ec-2-tag-set
deploy.list-application-revisions.s3-bucket	s-3-bucket
deploy.list-application-revisions.s3-key-prefix	s-3-key-prefix
deploy.update-deployment-group.ec2-tag-set	ec-2-tag-set
iam.enable-mfa-device.authentication-code1	authentication-code-1
iam.enable-mfa-device.authentication-code2	authentication-code-2
iam.resync-mfa-device.authentication-code1	authentication-code-1
iam.resync-mfa-device.authentication-code2	authentication-code-2
importexport.get-shipping-label.street1	street-1
importexport.get-shipping-label.street2	street-2
importexport.get-shipping-label.street3	street-3
lambda.publish-version.code-sha256	code-sha-256
lightsail.import-key-pair.public-key-base64	public-key-base-64
opsworks.register-volume.ec2-volume-id	ec-2-volume-id

AWS CLI version 2 uses Amazon S3 keys more consistently

For the Amazon S3 customization commands in the `s3` namespace, we improved the consistency of how paths are shown. In the AWS CLI version 2, paths are always displayed relative to the relevant key. The AWS CLI version 1 sometimes showed paths in absolute form and sometimes in relative form.

AWS CLI version 2 currently does not support the `[plugins]` section in the AWS `config` file

AWS CLI version 2 does not support the `[plugins]` section of the `~/.aws/config` file. Version 1 plugins do not work in version 2, and can cause failures. We recommend that you either remove the section or rename it.

Configuring the AWS CLI

This section explains how to configure the settings that the AWS Command Line Interface (AWS CLI) uses to interact with AWS. These include your security credentials, the default output format, and the default AWS Region.

Note

AWS requires that all incoming requests are cryptographically signed. The AWS CLI does this for you. The "signature" includes a date/time stamp. Therefore, you must ensure that your computer's date and time are set correctly. If you don't, and the date/time in the signature is too far off of the date/time recognized by the AWS service, AWS rejects the request.

Sections

- [Quickly Configuring the AWS CLI \(p. 37\)](#)
- [Creating Multiple Profiles \(p. 39\)](#)
- [Configuration Settings and Precedence \(p. 39\)](#)
- [Configuration and Credential File Settings \(p. 40\)](#)
- [Named Profiles \(p. 50\)](#)
- [Configuring the AWS CLI to use AWS Single Sign-On \(p. 51\)](#)
- [Environment Variables To Configure the AWS CLI \(p. 56\)](#)
- [Command Line Options \(p. 58\)](#)
- [Sourcing Credentials with an External Process \(p. 60\)](#)
- [Getting Credentials from EC2 Instance Metadata \(p. 61\)](#)
- [Using an HTTP Proxy \(p. 62\)](#)
- [Using an IAM Role in the AWS CLI \(p. 63\)](#)
- [Command Completion \(p. 69\)](#)

Quickly Configuring the AWS CLI

For general use, the `aws configure` command is the fastest way to set up your AWS CLI installation.

```
$ aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]: json
```

When you enter this command, the AWS CLI prompts you for four pieces of information (access key, secret access key, AWS Region, and output format). These are described in the following sections. The AWS CLI stores this information in a *profile* (a collection of settings) named `default`. The information in the `default` profile is used any time you run an AWS CLI command that doesn't explicitly specify a profile to use.

Access Key and Secret Access Key

The `AWS Access Key ID` and `AWS Secret Access Key` are your AWS credentials. They are associated with an AWS Identity and Access Management (IAM) user or role that determines what permissions you have. For a tutorial on how to create a user with the IAM service, see [Creating Your First IAM Admin User and Group](#) in the *IAM User Guide*.

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 from the AWS Management Console. As a best practice, do not use the AWS account root user access keys for any task where it's not required. Instead, [create a new administrator IAM user](#) with access keys for yourself.

The only time that you can view or download the secret access key is when you create the keys. You cannot recover them later. However, you can create new access keys at any time. You must also have permissions to perform the required IAM actions. For more information, see [Permissions Required to Access IAM Resources](#) in the *IAM User Guide*.

To create access keys for an IAM user

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Users**.
3. Choose the name of the user whose access keys you want to create, and then choose the **Security credentials** tab.
4. In the **Access keys** section, choose **Create access key**.
5. To view the new access key pair, choose **Show**. You will not have access to the secret access key again after this dialog box closes. Your credentials will look something like this:
 - 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. You will not have access to the secret access key again after this dialog box closes.

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

7. After you download the .csv file, choose **Close**. When you create an access key, the key pair is active by default, and you can use the pair right away.

Related topics

- [What Is IAM?](#) in the *IAM User Guide*
- [AWS Security Credentials](#) in *AWS General Reference*

Region

The `Default region name` identifies the AWS Region whose servers you want to send your requests to by default. This is typically the Region closest to you, but it can be any Region. For example, you can type `us-west-2` to use US West (Oregon). This is the Region that all later requests are sent to, unless you specify otherwise in an individual command.

Note

You must specify an AWS Region when using the AWS CLI, either explicitly or by setting a default Region. For a list of the available Regions, see [Regions and Endpoints](#). The Region designators used by the AWS CLI are the same names that you see in AWS Management Console URLs and service endpoints.

Output Format

The `Default output format` specifies how the results are formatted. The value can be any of the values in the following list. If you don't specify an output format, `json` is used as the default.

- **json**: The output is formatted as a [JSON](#) string.
- **text**: The output is formatted as multiple lines of tab-separated string values, which can be useful if you want to pass the output to a text processor, like `grep`, `sed`, or `awk`.
- **table**: The output is formatted as a table using the characters `+-` to form the cell borders. It typically presents the information in a "human-friendly" format that is much easier to read than the others, but not as programmatically useful.

Creating Multiple Profiles

If you use the command shown in the previous section, the result is a single profile named `default`. You can create additional configurations that you can refer to with a name by specifying the `--profile` option and assigning a name. The following example creates a profile named `produser`. You can specify credentials from a completely different account and Region than the other profiles.

```
$ aws configure --profile produser
AWS Access Key ID [None]: AKIAI44QH8DHBEXAMPLE
AWS Secret Access Key [None]: je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY
Default region name [None]: us-east-1
Default output format [None]: text
```

Then, when you run a command, you can omit the `--profile` option and use the credentials and settings stored in the default profile.

```
$ aws s3 ls
```

Or you can specify a `--profile` *profilename* and use the credentials and settings stored under that name.

```
$ aws s3 ls --profile produser
```

To update any of your settings, simply run `aws configure` again (with or without the `--profile` parameter, depending on which profile you want to update) and enter new values as appropriate. The next sections contain more information about the files that `aws configure` creates, additional settings, and named profiles.

Configuration Settings and Precedence

The AWS CLI uses a set of *credential providers* to look for AWS credentials. Each credential provider looks for credentials in a different place, such as the system or user environment variables, local AWS configuration files, or explicitly declared on the command line as a parameter. The AWS CLI looks for credentials and configuration settings by invoking the providers in the following order, stopping when it finds a set of credentials to use:

1. **Command line options (p. 58)** – You can specify `--region`, `--output`, and `--profile` as parameters on the command line.
2. **Environment variables (p. 56)** – You can store values in the environment variables: `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `AWS_SESSION_TOKEN`. If they are present, they are used.
3. **CLI credentials file (p. 40)** – This is one of the files that is updated when you run the command `aws configure`. The file is located at `~/.aws/credentials` on Linux or macOS, or at `C:\Users\USERNAME\.aws\credentials` on Windows. This file can contain the credential details for the default profile and any named profiles.

4. **CLI configuration file (p. 40)** – This is another file that is updated when you run the command `aws configure`. The file is located at `~/.aws/config` on Linux or macOS, or at `C:\Users\USERNAME\.aws\config` on Windows. This file contains the configuration settings for the default profile and any named profiles.
5. **Container credentials** – You can associate an IAM role with each of your Amazon Elastic Container Service (Amazon ECS) task definitions. Temporary credentials for that role are then available to that task's containers. For more information, see [IAM Roles for Tasks](#) in the *Amazon Elastic Container Service Developer Guide*.
6. **Instance profile credentials** – You can associate an IAM role with each of your Amazon Elastic Compute Cloud (Amazon EC2) instances. Temporary credentials for that role are then available to code running in the instance. The credentials are delivered through the Amazon EC2 metadata service. For more information, see [IAM Roles for Amazon EC2](#) in the *Amazon EC2 User Guide for Linux Instances* and [Using Instance Profiles](#) in the *IAM User Guide*.

Configuration and Credential File Settings

You can save your frequently used configuration settings and credentials in files that are maintained by the AWS CLI. The files are divided into sections that can be referenced by name. These are called "profiles". Unless you specify otherwise, the CLI uses the settings found in the profile named `default`. To use alternate settings, you can create and reference additional profiles. You can also override an individual setting by either setting one of the supported environment variables, or by using a command line parameter.

- [Where Are Configuration Settings Stored? \(p. 40\)](#)
- [Global Settings \(p. 42\)](#)
- [S3 Custom Command Settings \(p. 47\)](#)

Where Are Configuration Settings Stored?

The AWS CLI stores the credentials that you specify with `aws configure` in a local file named `credentials`, in a folder named `.aws` in your home directory. The other configuration options that you specify with `aws configure` are stored in a local file named `config`, also stored in the `.aws` folder in your home directory. Where you find your home directory location varies based on the operating system, but is referred to using the environment variables `%UserProfile%` in Windows and `$HOME` or `~` (tilde) in Unix-based systems.

For example, the following commands list the contents of the `.aws` folder.

Linux or macOS

```
$ ls ~/.aws
```

Windows

```
C:\> dir "%UserProfile%\aws"
```

The AWS CLI uses two files to store the sensitive credential information (in `~/.aws/credentials`) separated from the less sensitive configuration options (in `~/.aws/config`).

You can specify a non-default location for the `config` file by setting the `AWS_CONFIG_FILE` environment variable to another local path. See [Environment Variables To Configure the AWS CLI \(p. 56\)](#) for details.

Storing Credentials in the Config File

The AWS CLI can also read credentials from the `config` file. You can keep all of your profile settings in a single file. If there are ever credentials in both locations for a profile (say you used `aws configure` to update the profile's keys), the keys in the credentials file take precedence. These files are also used by the various language software development kits (SDKs). If you use one of the SDKs in addition to the AWS CLI, you might receive additional warnings if credentials aren't stored in their own file.

The files generated by the CLI for the profile configured in the previous section look like this.

`~/.aws/credentials`

```
[default]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxrFcYEXAMPLEKEY
```

`~/.aws/config`

```
[default]
region=us-west-2
output=json
```

Note

The preceding examples show the files with a single, default profile. For examples of the files with multiple named profiles, see [Named Profiles \(p. 50\)](#).

When you use a shared profile that specifies an IAM role, the AWS CLI calls the AWS STS `AssumeRole` operation to retrieve temporary credentials. These credentials are then stored (in `~/.aws/cli/cache`). Subsequent AWS CLI commands use the cached temporary credentials until they expire, and at that point the AWS CLI automatically refreshes the credentials.

Supported config File Settings

Topics

- [Global Settings \(p. 42\)](#)
- [S3 Custom Command Settings \(p. 47\)](#)

The following settings are supported in the `config` file. The values listed in the specified (or default) profile are used unless they are overridden by the presence of an environment variable with the same name, or a command line option with the same name.

You can configure these settings by editing the `config` file directly with a text editor, or by using the `aws configure set` command. Specify the profile that you want to modify with the `--profile` setting. For example, the following command sets the `region` setting in the profile named `integ`.

```
aws configure set region us-west-2 --profile integ
```

You can also retrieve the value for any setting by using the `get` subcommand.

```
$ aws configure get region --profile default
us-west-2
```

If the output is empty, the setting is not explicitly set and uses the default value.

Global Settings

api_versions

Some AWS services maintain multiple API versions to support backward compatibility. By default, CLI commands use the latest available API version. You can specify an API version to use for a profile by including the `api_versions` setting in the `config` file.

This is a "nested" setting that is followed by one or more indented lines that each identify one AWS service and the API version to use. See the documentation for each service to understand which API versions are available.

The following example shows how to specify an API version for two AWS services. These API versions are used only for commands that run under the profile that contains these settings.

```
api_versions =  
    ec2 = 2015-03-01  
    cloudfront = 2015-09-017
```

This setting does not have an environment variable or command line parameter equivalent.

aws_access_key_id (p. 37)

Specifies the AWS access key used as part of the credentials to authenticate the command request. Although this can be stored in the `config` file, we recommend that you store this in the `credentials` file.

Can be overridden by the `AWS_ACCESS_KEY_ID` environment variable. You can't specify the access key ID as a command line option.

```
aws_access_key_id = 123456789012
```

aws_secret_access_key (p. 37)

Specifies the AWS secret key used as part of the credentials to authenticate the command request. Although this can be stored in the `config` file, we recommend that you store this in the `credentials` file.

Can be overridden by the `AWS_SECRET_ACCESS_KEY` environment variable. You can't specify the secret access key as a command line option.

```
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

aws_session_token

Specifies an AWS session token. A session token is required only if you manually specify temporary security credentials. Although this can be stored in the `config` file, we recommend that you store this in the `credentials` file.

Can be overridden by the `AWS_SESSION_TOKEN` environment variable. You can't specify the session token as a command line option.

```
aws_session_token = AQoEXAMPLEH4aoAH0gNCAPyJxz4BlCFFxWNE1OPTgk5TthT  
+FvwqnKwRcOIfrRh3c/LTo6UDdyJwOOvEVPvLXCrrrUtdnniCEXAMPLE/  
IvU1dYUg2RVAJBanLiHb4IgRmpRV3zrkuWJOgQs8IZZaIv2BXIa2R4OlGk
```

ca_bundle

Specifies a CA certificate bundle (a file with the .pem extension) that is used to verify SSL certificates.

Can be overridden by the AWS_CA_BUNDLE environment variable or the `--ca-bundle` command line option.

```
ca_bundle = dev/apps/ca-certs/cabundle-2019mar05.pem
```

cli_follow_urlparam

This feature is available only with version 1 of the AWS CLI.

The following feature is available only if you use version 1 of the AWS CLI. It isn't available if you run version 2.

Specifies whether the CLI attempts to follow URL links in command line parameters that begin with `http://` or `https://`. When enabled, the retrieved content is used as the parameter value instead of the URL.

- **true:** This is the default value. If specified, any string parameters that begin with `http://` or `https://` are fetched and any downloaded content is used as the parameter value for the command.
- **false:** If specified, the CLI does not treat parameter string values that begin with `http://` or `https://` differently from other strings.

This entry does not have an equivalent environment variable or command line option.

```
cli_follow_urlparam = false
```

cli_timestamp_format

Specifies the format of timestamp values included in the output. You can specify either of the following values:

- **iso8601** – The default value for the AWS CLI version 2. If specified, the AWS CLI reformats all timestamps according to [ISO 8601](#).
- **wire** – The default value for the AWS CLI version 1. If specified, the AWS CLI displays all timestamp values exactly as received in the HTTP query response.

This entry does not have an equivalent environment variable or command line option.

```
cli_timestamp_format = iso8601
```

credential_process (p. 60)

Specifies an external command that the CLI runs to generate or retrieve authentication credentials to use for this command. The command must return the credentials in a specific format. For more information about how to use this setting, see [Sourcing Credentials with an External Process](#) (p. 60).

This entry does not have an equivalent environment variable or command line option.

```
credential_process = /opt/bin/awscreds-retriever --username susan
```

credential_source (p. 63)

Used within Amazon EC2 instances or EC2 containers to specify where the AWS CLI can find credentials to use to assume the role you specified with the `role_arn` parameter. You cannot specify both `source_profile` and `credential_source` in the same profile.

This parameter can have one of three values:

- **Environment** – Specifies that the AWS CLI is to retrieve source credentials from environment variables.
- **Ec2InstanceMetadata** – Specifies that the AWS CLI is to use the IAM role attached to the [EC2 instance profile](#) to get source credentials.
- **EcsContainer** – Specifies that the AWS CLI is to use the IAM role attached to the ECS container as source credentials.

```
credential_source = Ec2InstanceMetadata
```

duration_seconds

Specifies the maximum duration of the role session, in seconds. The value can range from 900 seconds (15 minutes) up to the maximum session duration setting for the role (which can be a maximum of 43200). This is an optional parameter and by default, the value is set to 3600 seconds.

external_id (p. 67)

Specifies a unique identifier that is used by third parties to assume a role in their customers' accounts. This maps to the `ExternalId` parameter in the `AssumeRole` operation. This parameter is needed only if the trust policy for the role specifies a value for `ExternalId`. For more information, see [How to use an External Gateway When Granting Access to Your AWS Resources to a Third Party](#) in the *IAM User Guide*.

mfa_serial (p. 66)

The identification number of an MFA device to use when assuming a role. This is mandatory only if the trust policy of the role being assumed includes a condition that requires MFA authentication. The value can be either a serial number for a hardware device (such as GAHT12345678) or an Amazon Resource Name (ARN) for a virtual MFA device (such as `arn:aws:iam::123456789012:mfa/user`).

output (p. 38)

Specifies the default output format for commands requested using this profile. You can specify any of the following values:

- **json** – This is the default value. The output is formatted as a [JSON](#) string.
- **text** – The output is formatted as multiple lines of tab-separated string values, which can be useful if you want to pass the output to a text processor, like `grep`, `sed`, or `awk`.
- **table** – The output is formatted as a table using the characters `+-` to form the cell borders. It typically presents the information in a "human-friendly" format that is much easier to read than the others, but not as programmatically useful.

Can be overridden by the `AWS_DEFAULT_OUTPUT` environment variable or the `--output` command line option.

```
output = table
```

parameter_validation

Specifies whether the AWS CLI client attempts to validate parameters before sending them to the AWS service endpoint.

- **true** – This is the default value. If specified, the CLI performs local validation of command line parameters.
- **false** – If specified, the CLI does not validate command line parameters before sending them to the AWS service endpoint.

This entry does not have an equivalent environment variable or command line option.

```
parameter_validation = false
```

[region \(p. 38\)](#)

Specifies the default AWS Region to send requests to for commands requested using this profile. You can specify any of the Region codes available for the chosen service as listed in [AWS Regions and Endpoints](#) in the *Amazon Web Services General Reference*.

Can be overridden by the `AWS_DEFAULT_REGION` environment variable or the `--region` command line option.

```
region = us-west-2
```

[role_arn \(p. 63\)](#)

Specifies the Amazon Resource Name (ARN) of an IAM role that you want to use to run the AWS CLI commands. You must also specify one of the following parameters to identify the credentials that have permission to assume this role:

- `source_profile`
- `credential_source`

```
role_arn = arn:aws:iam::123456789012:role/role-name
```

[role_session_name \(p. 67\)](#)

Specifies the name to attach to the role session. This value is provided to the `RoleSessionName` parameter when the AWS CLI calls the `AssumeRole` operation, and becomes part of the assumed role user ARN: `arn:aws:sts::123456789012:assumed-role/role_name/role_session_name`. This is an optional parameter. If you do not provide this value, a session name is generated automatically. This name appears in AWS CloudTrail logs for entries associated with this session.

```
role_session_name = maria_garcia_role
```

[source_profile \(p. 63\)](#)

Specifies a named profile with long-term credentials that the AWS CLI can use to assume a role that you specified with the `role_arn` parameter. You cannot specify both `source_profile` and `credential_source` in the same profile.

```
source_profile = production-profile
```

[sso_account_id \(p. 51\)](#)

Specifies the AWS account ID that contains the IAM role with the permission that you want to grant to the associated AWS SSO user.

This setting does not have an environment variable or command line option.

```
sso_account_id = 123456789012
```

This feature is available only with version 2 of the AWS CLI.

The following feature is available only if you use version 2 of the AWS CLI. It isn't available if you run version 1. For information about how to install the preview of version 2, see [Installing the AWS CLI version 2 \(p. 4\)](#).

sso_region (p. 51)

Specifies the AWS Region that contains the AWS SSO portal host. This is separate from, and can be a different Region than the default CLI region parameter.

This setting does not have an environment variable or command line option.

```
aws_sso_region = us-west-2
```

This feature is available only with version 2 of the AWS CLI.

The following feature is available only if you use version 2 of the AWS CLI. It isn't available if you run version 1. For information about how to install the preview of version 2, see [Installing the AWS CLI version 2 \(p. 4\)](#).

sso_role_name (p. 51)

Specifies the friendly name of the IAM role that defines the user's permissions when using this profile.

This setting does not have an environment variable or command line option.

```
sso_role_name = ReadAccess
```

This feature is available only with version 2 of the AWS CLI.

The following feature is available only if you use version 2 of the AWS CLI. It isn't available if you run version 1. For information about how to install the preview of version 2, see [Installing the AWS CLI version 2 \(p. 4\)](#).

sso_start_url (p. 51)

Specifies the URL that points to the organization's AWS SSO user portal. The AWS CLI uses this URL to establish a session with the AWS SSO service to authenticate its users.

This setting does not have an environment variable or command line option.

```
sso_start_url = https://my-sso-portal.awsapps.com/start
```

This feature is available only with version 2 of the AWS CLI.

The following feature is available only if you use version 2 of the AWS CLI. It isn't available if you run version 1. For information about how to install the preview of version 2, see [Installing the AWS CLI version 2 \(p. 4\)](#).

sts_regional_endpoints

Specifies the AWS service endpoint that the AWS CLI client uses to talk to the AWS Security Token Service (AWS STS). You can specify one of two values:

- **regional** – The AWS CLI uses the AWS STS endpoint that corresponds to the configured Region. For example, if the client is configured to use `us-west-2`, all calls to AWS STS are made to the regional endpoint `sts.us-west-2.amazonaws.com` instead of the global `sts.amazonaws.com` endpoint.
- **legacy** – Uses the global STS endpoint, `sts.amazonaws.com`, for the following Regions: `ap-northeast-1`, `ap-south-1`, `ap-southeast-1`, `ap-southeast-2`, `aws-global`, `ca-central-1`, `eu-central-1`, `eu-north-1`, `eu-west-1`, `eu-west-2`, `eu-west-3`, `sa-east-1`,

us-east-1, us-east-2, us-west-1, and us-west-2. All other Regions use their respective regional endpoint.

[web_identity_token_file](#) (p. 68)

Specifies the path to a file that contains an OAuth 2.0 access token or OpenID Connect ID token that is provided by an identity provider. The AWS CLI loads the contents of this file and passes it as the `WebIdentityToken` argument to the `AssumeRoleWithWebIdentity` operation.

`tcp_keepalive`

Specifies whether the AWS CLI client uses TCP keep-alive packets.

This entry does not have an equivalent environment variable or command line option.

```
tcp_keepalive = false
```

S3 Custom Command Settings

Amazon S3 supports several settings that configure how the AWS CLI performs Amazon S3 operations. Some apply to all S3 commands in both the `s3api` and `s3` namespaces. Others are specifically for the S3 "custom" commands that abstract common operations and do more than a one-to-one mapping to an API operation. The `aws s3` transfer commands `cp`, `sync`, `mv`, and `rm` have additional settings you can use to control S3 transfers.

All of these options can be configured by specifying the `s3` nested setting in your `config` file. Each setting is then indented on its own line.

Note

These settings are entirely optional. You should be able to successfully use the `aws s3` transfer commands without configuring any of these settings. These settings are provided to enable you to tune for performance or to account for the specific environment where you are running these `aws s3` commands.

The following settings apply to any S3 command in the `s3` or `s3api` namespaces.

`addressing_style`

Specifies which addressing style to use. This controls whether the bucket name is in the hostname or is part of the URL. Valid values are: `path`, `virtual`, and `auto`. The default value is `auto`.

There are two styles of constructing an S3 endpoint. The first is called `virtual` and includes the bucket name as part of the hostname. For example: `https://bucketname.s3.amazonaws.com`. Alternatively, with the `path` style, you treat the bucket name as if it is a path in the URI; for example, `https://s3.amazonaws.com/bucketname`. The default value in the CLI is to use `auto`, which attempts to use the `virtual` style where it can, but will fall back to `path` style when required. For example, if your bucket name is not DNS compatible, the bucket name cannot be part of the hostname and must be in the path. With `auto`, the CLI will detect this condition and automatically switch to `path` style for you. If you set the addressing style to `path`, you must then ensure that the AWS Region you configured in the AWS CLI matches the Region of your bucket.

`payload_signing_enabled`

Specifies whether to SHA256 sign sigv4 payloads. By default, this is disabled for streaming uploads (`UploadPart` and `PutObject`) when using HTTPS. By default, this is set to `false` for streaming uploads (`UploadPart` and `PutObject`), but only if a `ContentMD5` is present (it is generated by default) and the endpoint uses HTTPS.

If set to `true`, S3 requests receive additional content validation in the form of a SHA256 checksum which is calculated for you and included in the request signature. If set to `false`, the checksum

isn't calculated. Disabling this can be useful to reduce the performance overhead created by the checksum calculation.

`use_dualstack_endpoint`

Use the Amazon S3 dual IPv4 / IPv6 endpoint for all `s3` and `s3api` commands. The default value is `false`. This is mutually exclusive with the `use_accelerate_endpoint` setting.

If set to `true`, the AWS CLI directs all Amazon S3 requests to the dual IPv4 / IPv6 endpoint for the configured Region.

`use_accelerate_endpoint`

Use the Amazon S3 Accelerate endpoint for all `s3` and `s3api` commands. The default value is `false`. This is mutually exclusive with the `use_dualstack_endpoint` setting.

If set to `true`, the AWS CLI directs all Amazon S3 requests to the S3 Accelerate endpoint at `s3-accelerate.amazonaws.com`. To use this endpoint, you must enable your bucket to use S3 Accelerate. All requests are sent using the virtual style of bucket addressing: `my-bucket.s3-accelerate.amazonaws.com`. Any `ListBuckets`, `CreateBucket`, and `DeleteBucket` requests aren't sent to the S3 Accelerate endpoint as that endpoint doesn't support those operations. This behavior can also be set if the `--endpoint-url` parameter is set to `https://s3-accelerate.amazonaws.com` or `http://s3-accelerate.amazonaws.com` for any `s3` or `s3api` command.

The following settings apply only to commands in the `s3` namespace command set.

`max_bandwidth`

Specifies the maximum bandwidth that can be consumed for uploading and downloading data to and from Amazon S3. The default is no limit.

This limits the maximum bandwidth that the S3 commands can use to transfer data to and from Amazon S3. This value applies to only uploads and downloads; it doesn't apply to copies or deletes. The value is expressed as bytes per second. The value can be specified as:

- An integer. For example, `1048576` sets the maximum bandwidth usage to 1 megabyte per second.
- An integer followed by a rate suffix. You can specify rate suffixes using: `KB/s`, `MB/s`, or `GB/s`. For example, `300KB/s`, `10MB/s`.

In general, we recommend that you first try to lower bandwidth consumption by lowering `max_concurrent_requests`. If that doesn't adequately limit bandwidth consumption to the desired rate, you can use the `max_bandwidth` setting to further limit bandwidth consumption. This is because `max_concurrent_requests` controls how many threads are currently running. If you instead first lower `max_bandwidth` but leave a high `max_concurrent_requests` setting, it can result in threads having to wait unnecessarily. This can lead to excess resource consumption and connection timeouts.

`max_concurrent_requests`

Specifies the maximum number of concurrent requests. The default value is 10.

The `aws s3` transfer commands are multithreaded. At any given time, multiple Amazon S3 requests can be running. For example, when you use the command `aws s3 cp localdir s3://bucket/ --recursive` to upload files to an S3 bucket, the AWS CLI can upload the files `localdir/file1`, `localdir/file2`, and `localdir/file3` in parallel. The setting `max_concurrent_requests` specifies the maximum number of transfer operations that can run at the same time.

You might need to change this value for a few reasons:

- Decreasing this value – On some environments, the default of 10 concurrent requests can overwhelm a system. This can cause connection timeouts or slow the responsiveness of the system. Lowering this value makes the S3 transfer commands less resource intensive. The tradeoff

is that S3 transfers can take longer to complete. Lowering this value might be necessary if you use a tool to limit bandwidth.

- Increasing this value – In some scenarios, you might want the S3 transfers to complete as quickly as possible, using as much network bandwidth as necessary. In this scenario, the default number of concurrent requests might not be sufficient to use all of the available network bandwidth. Increasing this value can improve the time it takes to complete an S3 transfer.

`max_queue_size`

Specifies the maximum number of tasks in the task queue. The default value is 1000.

The AWS CLI internally uses a model where it queues up Amazon S3 tasks that are then executed by consumers whose numbers are limited by `max_concurrent_requests`. A task generally maps to a single S3 operation. For example, a task could be a `PutObjectTask`, or a `GetObjectTask`, or an `UploadPartTask`. The rate at which tasks are added to the queue can be much faster than the rate at which consumers finish the tasks. To avoid unbounded growth, the task queue size is capped to a specific size. This setting changes the value of that maximum number.

You generally don't need to change this setting. This setting also corresponds to the number of tasks that the CLI is aware of that need to be run. This means that by default the CLI can only see 1000 tasks ahead. Increasing this value means that the CLI can more quickly know the total number of tasks needed, assuming that the queuing rate is quicker than the rate of task completion. The tradeoff is that a larger `max_queue_size` requires more memory.

`multipart_chunksize`

Specifies the chunk size that the AWS CLI uses for multipart transfers of individual files. The default value is 8 MB, with a minimum of 5 MB.

When a file transfer exceeds the `multipart_threshold`, the CLI divides the file into chunks of this size. This value can be specified using the same syntax as `multipart_threshold`, either as the number of bytes as an integer, or by using a size and a suffix.

`multipart_threshold`

Specifies the size threshold the AWS CLI uses for multipart transfers of individual files. The default value is 8 MB.

When uploading, downloading, or copying a file, the S3 commands switch to multipart operations if the file exceeds this size. You can specify this value in one of two ways:

- The file size in bytes. For example, 1048576.
- The file size with a size suffix. You can use KB, MB, GB, or TB. For example: 10MB, 1GB.

Note

S3 can impose constraints on valid values that can be used for multipart operations. For more information, see the [S3 Multipart Upload documentation](#) in the *Amazon Simple Storage Service Developer Guide*.

These settings are all set under a top-level `s3` key in the `config` file, as shown in the following example for the development profile.

```
[profile development]
s3 =
  max_concurrent_requests = 20
  max_queue_size = 10000
  multipart_threshold = 64MB
  multipart_chunksize = 16MB
  max_bandwidth = 50MB/s
  use_accelerate_endpoint = true
  addressing_style = path
```

Named Profiles

The AWS CLI supports using any of multiple *named profiles* that are stored in the `config` and `credentials` files. You can configure additional profiles by using `aws configure` with the `--profile` option, or by adding entries to the `config` and `credentials` files.

The following example shows a `credentials` file with two profiles. The first is used when you run a CLI command with no profile. The second is used when you run a CLI command with the `--profile user1` parameter.

`~/.aws/credentials` (Linux & Mac) or `%USERPROFILE%\aws\credentials` (Windows)

```
[default]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

[user1]
aws_access_key_id=AKIAI44QH8DHBEXAMPLE
aws_secret_access_key=je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY
```

Each profile can specify different credentials—perhaps from different IAM users—and can also specify different AWS Regions and output formats.

`~/.aws/config` (Linux & Mac) or `%USERPROFILE%\aws\config` (Windows)

```
[default]
region=us-west-2
output=json

[profile user1]
region=us-east-1
output=text
```

Important

The `credentials` file uses a different naming format than the CLI `config` file for named profiles. Include the prefix word "profile" only when configuring a named profile in the `config` file. Do **not** use the word `profile` when creating an entry in the `credentials` file.

Using Profiles with the AWS CLI

To use a named profile, add the `--profile profile-name` option to your command. The following example lists all of your Amazon EC2 instances using the credentials and settings defined in the `user1` profile from the previous example files.

```
$ aws ec2 describe-instances --profile user1
```

To use a named profile for multiple commands, you can avoid specifying the profile in every command by setting the `AWS_PROFILE` environment variable at the command line.

Linux or macOS

```
$ export AWS_PROFILE=user1
```

Setting the environment variable changes the default profile until the end of your shell session, or until you set the variable to a different value. You can make environment variables persistent across future sessions by putting them in your shell's startup script. For more information, see [Environment Variables To Configure the AWS CLI](#) (p. 56).

Windows

```
C:\> setx AWS_PROFILE user1
```

Using `set` to set an environment variable changes the value used until the end of the current command prompt session, or until you set the variable to a different value.

Using `setx` to set an environment variable changes the value in all command shells that you create after running the command. It does **not** affect any command shell that is already running at the time you run the command. Close and restart the command shell to see the effects of the change.

Configuring the AWS CLI to use AWS Single Sign-On

This feature is available only with version 2 of the AWS CLI.

The following feature is available only if you use version 2 of the AWS CLI. It isn't available if you run version 1. For information about how to install the preview of version 2, see [Installing the AWS CLI version 2 \(p. 4\)](#).

If your organization uses AWS Single Sign-On (AWS SSO), your users can sign in to Active Directory or a built-in AWS SSO directory and get mapped to an AWS Identity and Access Management (IAM) role that enables you to run AWS CLI commands. For more information about AWS SSO, see the [AWS Single Sign-On User Guide](#).

This topic describes how to configure the AWS CLI to authenticate the user with AWS SSO to get short-term credentials to run AWS CLI commands. It includes the following sections:

- [Configuring a Named Profile to Use AWS SSO \(p. 51\)](#) - How to create and configure profiles that use AWS SSO for authentication and mapping to an IAM role for AWS permissions.
- [Using an AWS SSO Enabled Named Profile \(p. 54\)](#) - how to login to AWS SSO from the CLI and use the provided AWS temporary credentials to run AWS CLI commands.

Configuring a Named Profile to Use AWS SSO

You can configure one or more of your AWS CLI [named profiles \(p. 50\)](#) to use a role from AWS SSO. You can create and configure multiple profiles and configure each one to use a different AWS SSO user portal or SSO-defined role.

You can configure the profile in the following ways:

- [Automatically \(p. 51\)](#), using the command `aws2 configure sso`
- [Manually \(p. 53\)](#), by editing the `.aws/config` file that stores the named profiles.

Automatic Configuration

You can add an AWS SSO enabled profile to your AWS CLI by running the following command, providing your AWS SSO start URL and the AWS Region that hosts the AWS SSO directory.

```
$ aws2 configure sso
SSO start URL [None]: [None]: https://my-sso-portal.awsapps.com/start
SSO region [None]:us-east-1
```


The AWS CLI attempts to open your default browser and begin the login process for your AWS SSO account.

SSO authorization page has automatically been opened in your default browser.
Follow the instructions in the browser to complete this authorization request.

If the AWS CLI cannot open the browser, the following message appears with instructions on how to manually start the login process.

Using a browser, open the following URL:

<https://my-sso-portal.awsapps.com/verify>

and enter the following code:

QCFK-N451

AWS SSO uses the code to associate the AWS SSO session with your current AWS CLI session. The AWS SSO browser page prompts you to sign in with your AWS SSO account credentials. This enables the AWS CLI (through the permissions associated with your AWS SSO account) to retrieve and display the AWS accounts and roles that you are authorized to use with AWS SSO.

Next, the AWS CLI displays the AWS accounts available for you to use. If you are authorized to use only one account, the AWS CLI selects that account for you automatically and skips the prompt. The AWS accounts that are available for you to use are determined by your user configuration in AWS SSO.

There are 2 AWS accounts available to you.

```
> DeveloperAccount, developer-account-admin@example.com (123456789011)
   ProductionAccount, production-account-admin@example.com (123456789022)
```

Use the arrow keys to select the account you want to use with this profile. The ">" character on the left points to the current choice. Press ENTER to make your selection.

Next, the AWS CLI confirms your account choice, and displays the IAM roles that are available to you in the selected account. If the selected account lists only one role, the AWS CLI selects that role for you automatically and skips the prompt. The roles that are available for you to use are determined by your user configuration in AWS SSO.

Using the account ID **123456789011**

There are 2 roles available to you.

```
> ReadOnly
   FullAccess
```

As before, use the arrow keys to select the IAM role you want to use with this profile. The ">" character on the left points to the current choice. Press <ENTER> to make your selection.

The AWS CLI confirms your role selection.

Using the role name "ReadOnly"

Now you can finish the configuration of your profile, by specifying the [default output format \(p. 44\)](#), the [default AWS Region \(p. 45\)](#) to send commands to, and providing a [name for the profile \(p. 39\)](#) so you can reference this profile from among all those defined on the local computer. In the following example, the user enters a default Region, default output format, and the name of the profile. You can alternatively press <ENTER> to select any default values that are shown between the square brackets. The suggested profile name is the account ID number followed by an underscore followed by the role name.

```
CLI default client Region [None]: us-west-2<ENTER>
CLI default output format [None]: json<ENTER>
CLI profile name [123456789011_ReadOnly]: my-dev-profile<ENTER>
```

Note

If you specify `default` as the profile name, this profile becomes the one used whenever you run an AWS CLI command and do not specify a profile name.

A final message describes the completed profile configuration.

To use this profile, specify the profile name using `--profile`, as shown:

```
aws2 s3 ls --profile my-dev-profile
```

The previous example entries would result in a named profile in `~/.aws/config` that looks like the following example.

```
[profile my-dev-profile]
sso_start_url = https://my-sso-portal.awsapps.com/start
sso_region = us-east-1
sso_account_id = 123456789011
sso_role_name = readOnly
region = us-west-2
output = json
```

At this point, you have a profile that you can use to request temporary credentials. You must use the `aws sso login` command to actually request and retrieve the temporary credentials needed to run commands. For instructions, see [Using an AWS SSO Enabled Named Profile \(p. 54\)](#).

Note

You can also run an AWS CLI command using the specified profile. If you are not currently logged in to the AWS SSO portal, it starts the login process for you automatically, just as if you had manually ran the command `aws sso login` command.

Manual Configuration

To manually add AWS SSO support to a named profile, you must add the following keys and values to the profile definition in the file `~/.aws/config` (Linux or macOS) or `%USERPROFILE%/.aws/config` (Windows).

`sso_start_url`

The URL that points to the organization's AWS SSO user portal.

```
sso_start_url = https://my-sso-portal.awsapps.com/start
```

`sso_region`

The AWS Region that contains the AWS SSO portal host. This is separate from, and can be a different region than the default CLI `region` parameter.

```
sso_region = us-west-2
```

`sso_account_id`

The AWS account ID that contains the IAM role that you want to use with this profile.

```
sso_account_id = 123456789011
```

sso_role_name

The name of the IAM role that defines the user's permissions when using this profile.

```
sso_role_name = ReadAccess
```

The presence of these keys identify this profile as one that uses AWS SSO to authenticate the user.

You can also include any other keys and values that are valid in the `.aws/config` file, such as [region](#), [output](#), or [s3](#). However, you can't include any credential related values, such as [role_arn](#) (p. 45) or [aws_secret_access_key](#) (p. 42). If you do, the AWS CLI produces an error.

So a typical AWS SSO profile in `.aws/config` might look similar to the following example.

```
[profile my-dev-profile]
sso_start_url = https://my-sso-portal.awsapps.com/start
sso_region = us-east-1
sso_account_id = 123456789011
sso_role_name = readOnly
region = us-west-2
output = json
```

At this point, you have a profile that you can use to request temporary credentials. However, you can't yet run an AWS CLI service command. You must first use the `aws sso login` command to actually request and retrieve the temporary credentials needed to run commands. For instructions, see the next section, [Using an AWS SSO Enabled Named Profile](#) (p. 54).

Using an AWS SSO Enabled Named Profile

This section describes how to use the AWS SSO profile you created in the previous section.

Signing In and Getting Temporary Credentials

After you configure a named profile automatically or manually, you can invoke it to request temporary credentials from AWS. Before you can run an AWS CLI service command, you must retrieve and cache a set of temporary credentials. To get these temporary credentials, run the following command.

```
$ aws2 sso login --profile my-dev-profile
```

The AWS CLI opens your default browser and verifies your AWS SSO log in.

```
SSO authorization page has automatically been opened in your default browser.
Follow the instructions in the browser to complete this authorization request.
Successfully logged into Start URL: https://my-sso-portal.awsapps.com/start
```

If you are not currently signed in to your AWS SSO account, you must provide your AWS SSO user name and password.

If the AWS CLI can't open your browser, it prompts you to open it yourself and enter the specified code.

```
$ aws2 sso login --profile my-dev-profile
Using a browser, open the following URL:
```

```
https://my-sso-portal.awsapps.com/verify
```

and enter the following code:
QCFK-N451

The AWS CLI opens your default browser (or you manually open the browser of your choice) to the specified page, and enter the provided code. The webpage then prompts you for your AWS SSO credentials.

Your AWS SSO session credentials are cached and include an expiration timestamp. When the credentials expire, the AWS CLI requests you to sign in to AWS SSO again.

If your AWS SSO credentials are valid, the AWS CLI uses them to securely retrieve AWS temporary credentials for the IAM role specified in the profile.

```
Welcome, you have successfully signed-in to the AWS-CLI.
```

Running a Command with Your AWS SSO Enabled Profile

You can use these temporary credentials to invoke an AWS CLI command with the associated named profile. The following example shows that the command was run under an assumed role that is part of the specified account.

```
$ aws2 sts get-caller-identity --profile my-dev-profile
{
  "UserId": "AROA12345678901234567:test-user@example.com",
  "Account": "123456789011",
  "Arn": "arn:aws:sts::123456789011:assumed-role/AWSPeregrine_readOnly_12321abc454d123/test-user@example.com"
}
```

As long as you signed in to AWS SSO and those cached credentials are not expired, the AWS CLI automatically renews expired AWS temporary credentials when needed. However, if your AWS SSO credentials expire, you must explicitly renew them by logging in to your AWS SSO account again.

```
$ aws2 s3 ls --profile my-sso-profile
Your short-term credentials have expired. Please sign-in to renew your credentials
SSO authorization page has automatically been opened in your default browser.
Follow the instructions in the browser to complete this authorization request.
```

You can create multiple AWS SSO enabled named profiles that each point to a different AWS account or role. You can also use the **aws sso login** command on more than one profile at a time. If any of them share the same AWS SSO user account, you must log in to that AWS SSO user account only once and then they all share a single set of AWS SSO cached credentials.

```
# The following command retrieves temporary credentials for the AWS account and role
# specified in one named profile. If you are not yet signed in to AWS SSO or your
# cached credentials have expired, it opens your browser and prompts you for your
# AWS SSO user name and password. It then retrieves AWS temporary credentials for
# the IAM role associated with this profile.
$ aws2 sso login --profile my-first-sso-profile

# The next command retrieves a different set of temporary credentials for the AWS
# account and role specified in the second named profile. It does not overwrite or
# in any way compromise the first profile's credentials. If this profile specifies the
# same AWS SSO portal, then it uses the SSO credentials that you retrieved in the
# previous command. The AWS CLI then retrieves AWS temporary credentials for the
# IAM role associated with the second profile. You don't have to sign in to
# AWS SSO again.
```

```
$ aws2 sso login --profile my-second-sso-profile

# The following command lists the Amazon EC2 instances accessible to the role
# identified in the first profile.
$ aws2 ec2 describe-instances --profile my-first-sso-profile

# The following command lists the Amazon EC2 instances accessible to the role
# identified in the second profile.
$ aws2 ec2 describe-instances --profile my-second-sso-profile
```

Signing Out of Your AWS SSO Sessions

When you are done using your AWS SSO enabled profiles, you can choose to do nothing and let the AWS temporary credentials and your AWS SSO credentials expire. However, you can also choose to run the following command to immediately delete all cached credentials in the SSO credential cache folder and all AWS temporary credentials that were based on the AWS SSO credentials. This makes those credentials unavailable to be used for any future command.

```
$ aws2 sso logout
Successfully signed out of all SSO profiles.
```

If you later want to run commands with one of your AWS SSO enabled profiles, you must again run the `aws sso login` command (see the previous section) and specify the profile to use.

Environment Variables To Configure the AWS CLI

Environment variables provide another way to specify configuration options and credentials, and can be useful for scripting or temporarily setting a named profile as the default.

Precedence of options

- If you specify an option by using one of the environment variables described in this topic, it overrides any value loaded from a profile in the configuration file.
- If you specify an option by using a parameter on the CLI command line, it overrides any value from either the corresponding environment variable or a profile in the configuration file.

Supported environment variables

The AWS CLI supports the following environment variables.

AWS_ACCESS_KEY_ID

Specifies an AWS access key associated with an IAM user or role.

If defined, this environment variable overrides the value for the profile setting `aws_access_key_id`. You can't specify the access key ID by using a command line option.

AWS_CA_BUNDLE

Specifies the path to a certificate bundle to use for HTTPS certificate validation.

If defined, this environment variable overrides the value for the profile setting `ca_bundle`. You can override this environment variable by using the `--ca-bundle` command line parameter.

AWS_CONFIG_FILE

Specifies the location of the file that the AWS CLI uses to store configuration profiles. The default path is `~/.aws/config`.

You can't specify this value in a named profile setting or by using a command line parameter.

[AWS_DEFAULT_OUTPUT \(p. 38\)](#)

Specifies the [output format \(p. 86\)](#) to use.

If defined, this environment variable overrides the value for the profile setting `output`. You can override this environment variable by using the `--output` command line parameter.

[AWS_DEFAULT_REGION \(p. 38\)](#)

Specifies the AWS Region to send the request to.

If defined, this environment variable overrides the value for the profile setting `region`. You can override this environment variable by using the `--region` command line parameter.

[AWS_PROFILE \(p. 50\)](#)

Specifies the name of the CLI profile with the credentials and options to use. This can be the name of a profile stored in a `credentials` or `config` file, or the value `default` to use the default profile.

If defined, this environment variable overrides the behavior of using the profile named `[default]` in the configuration file. You can override this environment variable by using the `--profile` command line parameter.

[AWS_ROLE_SESSION_NAME \(p. 67\)](#)

Specifies a name to associate with the role session. This value appears in CloudTrail logs for commands performed by the user of this profile.

If defined, this environment variable overrides the value for the profile setting `role_session_name`. You can't specify a role session name as a command line parameter.

[AWS_SECRET_ACCESS_KEY](#)

Specifies the secret key associated with the access key. This is essentially the "password" for the access key.

If defined, this environment variable overrides the value for the profile setting `aws_secret_access_key`. You can't specify the access key ID as a command line option.

[AWS_SESSION_TOKEN](#)

Specifies the session token value that is required if you are using temporary security credentials that you retrieved directly from AWS STS operations. For more information, see the [Output section of the `assume-role` command](#) in the *AWS CLI Command Reference*.

If defined, this environment variable overrides the value for the profile setting `aws_session_token`. You can't specify the session token as a command line option.

[AWS_SHARED_CREDENTIALS_FILE](#)

Specifies the location of the file that the AWS CLI uses to store access keys. The default path is `~/.aws/credentials`.

You can't specify this value in a named profile setting or by using a command line parameter.

Note

You can't specify AWS Single Sign-On (AWS SSO) authentication by using environment variables. Instead, you must use a named profile in the shared configuration file `.aws/config`. For more information, see [Configuring the AWS CLI to use AWS Single Sign-On \(p. 51\)](#).

The following example shows how you could configure environment variables for the default user. These values would override any values found in a named profile, or instance metadata. Once set, you can

override these values by specifying a parameter on the CLI command line, or by changing or removing the environment variable. For more information about precedence and how the AWS CLI determines which credentials to use, see [Configuration Settings and Precedence](#) (p. 39).

Linux or macOS

```
$ export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
$ export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY
$ export AWS_DEFAULT_REGION=us-west-2
```

Setting the environment variable changes the value used until the end of your shell session, or until you set the variable to a different value. You can make the variables persistent across future sessions by setting them in your shell's startup script.

Windows Command Prompt

```
C:\> setx AWS_ACCESS_KEY_ID AKIAIOSFODNN7EXAMPLE
C:\> setx AWS_SECRET_ACCESS_KEY wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY
C:\> setx AWS_DEFAULT_REGION us-west-2
```

Using `set` to set an environment variable changes the value used until the end of the current command prompt session, or until you set the variable to a different value. Using `setx` to set an environment variable changes the value used in both the current command prompt session and all command prompt sessions that you create after running the command. It does *not* affect other command shells that are already running at the time you run the command.

PowerShell

```
PS C:\> $Env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
PS C:\> $Env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY"
PS C:\> $Env:AWS_DEFAULT_REGION="us-west-2"
```

If you set an environment variable at the PowerShell prompt as shown in the previous examples, it saves the value for only the duration of the current session. To make the environment variable setting persistent across all PowerShell and Command Prompt sessions, store it by using the **System** application in **Control Panel**. Alternatively, you can set the variable for all future PowerShell sessions by adding it to your PowerShell profile. See the [PowerShell documentation](#) for more information about storing environment variables or persisting them across sessions.

Command Line Options

You can use the following command line options to override the default configuration settings for a single command. You can't use command line options to directly specify credentials, although you can specify which profile to use.

--profile *<string>*

Specifies the [named profile](#) (p. 50) to use for this command. To set up additional named profiles, you can use the `aws configure` command with the `--profile` option.

```
$ aws configure --profile <profilename>
```

--region *<string>*

Specifies which AWS Region to send this command's AWS request to. For a list of all of the Regions that you can specify, see [AWS Regions and Endpoints](#) in the *Amazon Web Services General Reference*.

--output *<string>*

Specifies the output format to use for this command. You can specify any of the following values:

- **json**: The output is formatted as a [JSON](#) string.
- **text**: The output is formatted as multiple lines of tab-separated string values, which can be useful if you want to pass the output to a text processor, like `grep`, `sed`, or `awk`.
- **table**: The output is formatted as a table using the characters `+``|``-` to form the cell borders. It typically presents the information in a "human-friendly" format that is much easier to read than the others, but not as programmatically useful.

--endpoint-url *<string>*

Specifies the URL to send the request to. For most commands, the AWS CLI automatically determines the URL based on the selected service and the specified AWS Region. However, some commands require that you specify an account-specific URL. You can also configure some AWS services to [host an endpoint directly within your private VPC](#), which might then need to be specified.

For a list of the standard service endpoints available in each Region, see [AWS Regions and Endpoints](#) in the *Amazon Web Services General Reference*.

--debug

A Boolean switch that enables debug logging. This includes additional diagnostic information about the operation of the command that can be useful when troubleshooting why a command provides unexpected results.

--no-paginate

A Boolean switch that disables automatic pagination of the output.

--query *<string>*

Specifies a [JMESPath query](#) to use in filtering the response data. For more information, see [How to Filter the Output with the --query Option \(p. 90\)](#).

--version

A Boolean switch that displays the current version of the AWS CLI program that is running.

--color *<string>*

Specifies support for color output. Valid values are `on`, `off`, and `auto`. The default value is `auto`.

--no-sign-request

A Boolean switch that disables signing the HTTP requests to the AWS service endpoint. This prevents credentials from being loaded.

--ca-bundle *<string>*

Specifies the certificate authority (CA) certificate bundle to use when verifying SSL certificates.

--cli-read-timeout *<integer>*

Specifies the maximum socket read time in seconds. If the value is set to zero (0) the socket read waits indefinitely (is blocking) and doesn't timeout.

--cli-connect-timeout *<integer>*

Specifies the maximum socket connect time in seconds. If the value is set to zero (0), the socket connect waits indefinitely (is blocking) and doesn't timeout.

When you provide one or more of these options as command line parameters, they override the default configuration, any corresponding profile setting, or environment variable setting for that single command.

Each option that takes an argument requires a space or equals sign (=) separating the argument from the option name. If the argument value is a string that contains a space, you must use quotation marks around the argument.

Common uses for command line options include checking your resources in multiple AWS Regions, and changing the output format for legibility or ease of use when scripting. For example, if you're not sure which Region your instance is running in, you can run the **describe-instances** command against each Region until you find it, as follows.

```
$ aws ec2 describe-instances --output table --region us-east-1
-----
|DescribeInstances|
+-----+
$ aws ec2 describe-instances --output table --region us-west-1
-----
|DescribeInstances|
+-----+
$ aws ec2 describe-instances --output table --region us-west-2
-----
|                                     DescribeInstances                                     |
+-----+-----+-----+-----+-----+-----+
||                                     Reservations                                     ||
+-----+-----+-----+-----+-----+-----+
||   OwnerId           | 012345678901 |   ||
||   ReservationId     | r-abcdefgh |   ||
+-----+-----+-----+-----+-----+-----+
||                                     Instances                                     ||
+-----+-----+-----+-----+-----+-----+
||   AmiLaunchIndex    | 0         |   ||
||   Architecture      | x86_64    |   ||
...

```

The argument types (for example, string, Boolean) for each command line option are described in detail in [Specifying Parameter Values \(p. 76\)](#).

Sourcing Credentials with an External Process

Warning

This topic discusses sourcing credentials from an external process. This could be a security risk if the command to generate the credentials becomes accessible by non-approved processes or users. We recommend that you use the supported, secure alternatives provided by the AWS CLI and AWS to reduce the risk of compromising your credentials. Ensure that you secure the `config` file and any supporting files and tools to prevent disclosure.

Ensure that your custom credential tool does not write any secret information to `stderr` because the SDKs and CLI can capture and log such information, potentially exposing it to unauthorized users.

If you have a method to generate or look up credentials that isn't directly supported by the AWS CLI, you can configure the CLI to use it by configuring the `credential_process` setting in the `config` file.

For example, you might include an entry similar to the following in the `config` file.

```
[profile developer]
credential_process = /opt/bin/awscreds-custom --username helen

```

Syntax

To create this string in a way that is compatible with any operating system, follow these rules:

- If the path or file name contains a space, surround the complete path and file name with double-quotation marks (" "). The path and file name can consist of only the characters: A-Z a-z 0-9 - _ . space
- If a parameter name or a parameter value contains a space, surround that element with double-quotation marks (" "). Surround only the name or value, not the pair.
- Do not include any environment variables in the strings. For example, you can't include \$HOME or %USERPROFILE%.
- Do not specify the home folder as ~. You must specify the full path.

Example for Windows

```
credential_process = "C:\Path\To\credentials.cmd" parameterWithoutSpaces "parameter with spaces"
```

Example for Linux or macOS

```
credential_process = "/Users/Dave/path/to/credentials.sh" parameterWithoutSpaces "parameter with spaces"
```

Expected output from the Credentials program

The AWS CLI runs the command as specified in the profile and then reads data from `STDOUT`. The command you specify must generate JSON output on `STDOUT` that matches the following syntax.

```
{
  "Version": 1,
  "AccessKeyId": "an AWS access key",
  "SecretAccessKey": "your AWS secret access key",
  "SessionToken": "the AWS session token for temporary credentials",
  "Expiration": "ISO8601 timestamp when the credentials expire"
}
```

Note

As of this writing, the `Version` key must be set to 1. This might increment over time as the structure evolves.

The `Expiration` key is an [ISO8601](#) formatted timestamp. If the `Expiration` key is not present in the tool's output, the CLI assumes that the credentials are long-term credentials that do not refresh. Otherwise the credentials are considered temporary credentials and are refreshed automatically by rerunning the `credential_process` command before they expire.

Note

The AWS CLI does *not* cache external process credentials the way it does assume-role credentials. If caching is required, you must implement it in the external process.

The external process can return a non-zero return code to indicate that an error occurred while retrieving the credentials.

Getting Credentials from EC2 Instance Metadata

When you run the AWS CLI from within an Amazon Elastic Compute Cloud (Amazon EC2) instance, you can simplify providing credentials to your commands. Each Amazon EC2 instance contains metadata that the AWS CLI can directly query for temporary credentials. To provide these, create an AWS Identity and Access Management (IAM) role that has access to the resources needed, and attach that role to the Amazon EC2 instance when you launch it.

Launch the instance and check to see if the AWS CLI is already installed (it comes preinstalled on Amazon Linux). If necessary, install the AWS CLI. You must still configure a default AWS Region to avoid having to specify it in every command.

In a named profile, to specify that you want to use the credentials available in the hosting Amazon EC2 instance profile, use the following line in the configuration file.

```
credential_source = Ec2InstanceMetadata
```

The following example shows how to assume the `marketingadminrole` role by referencing it in an Amazon EC2 instance profile.

```
[profile marketingadmin]
role_arn = arn:aws:iam::123456789012:role/marketingadminrole
credential_source = Ec2InstanceMetadata
```

To set the Region and default output format by running `aws configure` without specifying credentials, press **Enter** twice to skip the first two prompts.

```
$ aws configure
AWS Access Key ID [None]: ENTER
AWS Secret Access Key [None]: ENTER
Default region name [None]: us-west-2
Default output format [None]: json
```

When an IAM role is attached to the instance, the AWS CLI automatically and securely retrieves the credentials from the instance metadata. For more information, see [Granting Applications That Run on Amazon EC2 Instances Access to AWS Resources](#) in the *IAM User Guide*.

Using an HTTP Proxy

To access AWS through proxy servers, you can configure the `HTTP_PROXY` and `HTTPS_PROXY` environment variables with either the DNS domain names or IP addresses and port numbers that your proxy servers use.

Note

The following examples show the environment variable name in all uppercase letters. However, if you specify a variable twice—once with uppercase letters and once with lowercase letters—the one with lowercase letters wins. We recommend that you define each variable only once to avoid confusion and unexpected behavior.

The following examples show how you can use either the explicit IP address of your proxy or a DNS name that resolves to the IP address of your proxy. Either can be followed by a colon and the port number to which queries should be sent.

Linux or macOS

```
$ export HTTP_PROXY=http://10.15.20.25:1234
$ export HTTP_PROXY=http://proxy.example.com:1234
$ export HTTPS_PROXY=http://10.15.20.25:5678
$ export HTTPS_PROXY=http://proxy.example.com:5678
```

Windows

```
C:\> setx HTTP_PROXY http://10.15.20.25:1234
C:\> setx HTTP_PROXY=http://proxy.example.com:1234
```

```
C:\> setx HTTPS_PROXY=http://10.15.20.25:5678
C:\> setx HTTPS_PROXY=http://proxy.example.com:5678
```

Authenticating to a Proxy

The AWS CLI supports HTTP Basic authentication. Specify the user name and password in the proxy URL, as follows.

Linux or macOS

```
$ export HTTP_PROXY=http://username:password@proxy.example.com:1234
$ export HTTPS_PROXY=http://username:password@proxy.example.com:5678
```

Windows

```
C:\> setx HTTP_PROXY http://username:password@proxy.example.com:1234
C:\> setx HTTPS_PROXY=http://username:password@proxy.example.com:5678
```

Note

The AWS CLI doesn't support NTLM proxies. If you use an NTLM or Kerberos protocol proxy, you might be able to connect through an authentication proxy like [Cntlm](#).

Using a Proxy on Amazon EC2 Instances

If you configure a proxy on an Amazon EC2 instance launched with an attached IAM role, ensure that you exempt the address used to access the [instance metadata](#). To do this, set the `NO_PROXY` environment variable to the IP address of the instance metadata service, 169.254.169.254. This address does not vary.

Linux or macOS

```
$ export NO_PROXY=169.254.169.254
```

Windows

```
C:\> setx NO_PROXY 169.254.169.254
```

Using an IAM Role in the AWS CLI

An [AWS Identity and Access Management \(IAM\) role](#) is an authorization tool that lets an IAM user gain additional (or different) permissions, or get permissions to perform actions in a different AWS account.

You can configure the AWS Command Line Interface (AWS CLI) to use an IAM role by defining a profile for the role in the `~/.aws/credentials` file.

The following example shows a role profile named `marketingadmin`. If you run commands with `--profile marketingadmin` (or specify it with the [AWS_PROFILE environment variable \(p. 56\)](#)), the CLI uses the credentials defined in the profile `user1` to assume the role with the Amazon Resource Name (ARN) `arn:aws:iam::123456789012:role/marketingadminrole`. You can run any operations that are allowed by the permissions assigned to that role.

```
[marketingadmin]
role_arn = arn:aws:iam::123456789012:role/marketingadminrole
source_profile = user1
```

You can then specify a `source_profile` that points to a separate named profile that contains IAM user credentials with permission to use the role. In the previous example, the `marketingadmin` profile uses the credentials in the `user1` profile. When you specify that an AWS CLI command is to use the profile `marketingadmin`, the CLI automatically looks up the credentials for the linked `user1` profile and uses them to request temporary credentials for the specified IAM role. The CLI uses the [sts:AssumeRole](#) operation in the background to accomplish this. Those temporary credentials are then used to run the requested CLI command. The specified role must have attached IAM permission policies that allow the requested CLI command to run.

To run a CLI command from within an Amazon Elastic Compute Cloud (Amazon EC2) instance or an Amazon Elastic Container Service (Amazon ECS) container, you can use an IAM role attached to the instance profile or the container. If you specify no profile or set no environment variables, that role is used directly. This enables you to avoid storing long-lived access keys on your instances. You can also use those instance or container roles only to get credentials for another role. To do this, you use `credential_source` (instead of `source_profile`) to specify how to find the credentials. The `credential_source` attribute supports the following values:

- `Environment` – Retrieves the source credentials from environment variables.
- `Ec2InstanceMetadata` – Uses the IAM role attached to the Amazon EC2 instance profile.
- `EcsContainer` – Uses the IAM role attached to the Amazon ECS container.

The following example shows the same `marketingadminrole` role used by referencing an Amazon EC2 instance profile.

```
[profile marketingadmin]
role_arn = arn:aws:iam::123456789012:role/marketingadminrole
credential_source = Ec2InstanceMetadata
```

When you invoke a role, you have additional options that you can require, such as the use of multi-factor authentication and an External ID (used by third-party companies to access their clients' resources). You can also specify unique role session names that can be more easily audited in AWS CloudTrail logs.

Sections

- [Configuring and Using a Role](#) (p. 64)
- [Using Multi-Factor Authentication](#) (p. 66)
- [Cross-Account Roles and External ID](#) (p. 67)
- [Specifying a Role Session Name for Easier Auditing](#) (p. 67)
- [Assume Role with Web Identity](#) (p. 68)
- [Clearing Cached Credentials](#) (p. 68)

Configuring and Using a Role

When you run commands using a profile that specifies an IAM role, the AWS CLI uses the source profile's credentials to call AWS Security Token Service (AWS STS) and request temporary credentials for the specified role. The user in the source profile must have permission to call `sts:assume-role` for the role in the specified profile. The role must have a trust relationship that allows the user in the source profile to use the role. The process of retrieving and then using temporary credentials for a role is often referred to as *assuming the role*.

You can create a role in IAM with the permissions that you want users to assume by following the procedure under [Creating a Role to Delegate Permissions to an IAM User](#) in the *AWS Identity and Access Management User Guide*. If the role and the source profile's IAM user are in the same account, you can enter your own account ID when configuring the role's trust relationship.

After creating the role, modify the trust relationship to allow the IAM user (or the users in the AWS account) to assume it.

The following example shows a trust policy that you could attach to a role. This policy allows the role to be assumed by any IAM user in the account 123456789012, *if* the administrator of that account explicitly grants the `sts:assumerole` permission to the user.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:root"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

The trust policy doesn't actually grant permissions. The administrator of the account must delegate the permission to assume the role to individual users by attaching a policy with the appropriate permissions. The following example shows a policy that you can attach to an IAM user that allows the user to assume only the `marketingadminrole` role. For more information about granting a user access to assume a role, see [Granting a User Permission to Switch Roles](#) in the *IAM User Guide*.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "arn:aws:iam::123456789012:role/marketingadminrole"
    }
  ]
}
```

The IAM user doesn't need to have additional permissions to run the CLI commands using the role profile. Instead, the permissions to run the command come from those attached to the *role*. You attach permission policies to the role to specify which actions can be performed against which AWS resources. For more information about attaching permissions to a role (which works identically to an IAM user), see [Changing Permissions for an IAM User](#) in the *IAM User Guide*.

Now that you have the role profile, role permissions, role trust relationship, and user permissions correctly configured, you can use the role at the command line by invoking the `--profile` option. For example, the following calls the Amazon S3 `ls` command using the permissions attached to the `marketingadmin` role as defined by the example at the beginning of this topic.

```
$ aws s3 ls --profile marketingadmin
```

To use the role for several calls, you can set the `AWS_DEFAULT_PROFILE` environment variable for the current session from the command line. While that environment variable is defined, you don't have to specify the `--profile` option on each command.

Linux or macOS

```
$ export AWS_PROFILE=marketingadmin
```

Windows

```
C:\> setx AWS_PROFILE marketingadmin
```

For more information about configuring IAM users and roles, see [Users and Groups](#) and [Roles](#) in the *IAM User Guide*.

Using Multi-Factor Authentication

For additional security, you can require that users provide a one-time key generated from a multi-factor authentication (MFA) device, a U2F device, or mobile app when they attempt to make a call using the role profile.

First, you can choose to modify the trust relationship on the IAM role to require MFA. This prevents anyone from using the role without first authenticating by using MFA. For an example, see the Condition line in the following example. This policy allows the IAM user named anika to assume the role the policy is attached to, but only if they authenticate by using MFA.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::123456789012:user/anika" },
      "Action": "sts:AssumeRole",
      "Condition": { "Bool": { "aws:multifactorAuthPresent": true } }
    }
  ]
}
```

Next, add a line to the role profile that specifies the ARN of the user's MFA device. The following sample config file entries show two role profiles that both use the access keys for the IAM user anika to request temporary credentials for the role cli-role. The user anika has permissions to assume the role, granted by the role's trust policy.

```
[profile role-without-mfa]
region = us-west-2
role_arn= arn:aws:iam::128716708097:role/cli-role
source_profile=cli-user

[profile role-with-mfa]
region = us-west-2
role_arn= arn:aws:iam::128716708097:role/cli-role
source_profile = cli-user
mfa_serial = arn:aws:iam::128716708097:mfa/cli-user

[profile anika]
region = us-west-2
output = json
```

The mfa_serial setting can take an ARN, as shown, or the serial number of a hardware MFA token.

The first profile, role-without-mfa, doesn't require MFA. However, because the previous example trust policy attached to the role requires MFA, any attempt to run a command with this profile fails.

```
$ aws iam list-users --profile role-without-mfa
```

```
An error occurred (AccessDenied) when calling the AssumeRole operation: Access denied
```

The second profile entry, `role-with-mfa`, identifies an MFA device to use. When the user attempts to run a CLI command with this profile, the CLI prompts the user to enter the one-time password (OTP) that the MFA device provides. If the MFA authentication succeeds, the command performs the requested operation. The OTP is not displayed on the screen.

```
$ aws iam list-users --profile role-with-mfa
Enter MFA code for arn:aws:iam::123456789012:mfa/cli-user:
{
  "Users": [
    {
      ...
    }
  ]
}
```

Cross-Account Roles and External ID

You can enable IAM users to use roles that belong to different accounts by configuring the role as a cross-account role. During role creation, set the role type to **Another AWS account**, as described in [Creating a Role to Delegate Permissions to an IAM user](#). Optionally, select **Require MFA**. **Require MFA** configures the appropriate condition in the trust relationship, as described in [Using Multi-Factor Authentication](#) (p. 66).

If you use an [external ID](#) to provide additional control over who can use a role across accounts, you must also add the `external_id` parameter to the role profile. You typically use this only when the other account is controlled by someone outside your company or organization.

```
[profile crossaccountrole]
role_arn = arn:aws:iam::234567890123:role/SomeRole
source_profile = default
mfa_serial = arn:aws:iam::123456789012:mfa/saanvi
external_id = 123456
```

Specifying a Role Session Name for Easier Auditing

When many individuals share a role, auditing becomes more of a challenge. You want to associate each operation invoked with the individual who invoked the action. However, when the individual uses a role, the assumption of the role by the individual is a separate action from the invoking of an operation, and you must manually correlate the two.

You can simplify this by specifying unique role session names when users assume a role. You do this by adding a `role_session_name` parameter to each named profile in the `config` file that specifies a role. The `role_session_name` value is passed to the `AssumeRole` operation and becomes part of the ARN for the role session. It is also included in the AWS CloudTrail logs for all logged operations.

For example, you could create a role-based profile as follows.

```
[profile namedsessionrole]
role_arn = arn:aws:iam::234567890123:role/SomeRole
source_profile = default
role_session_name = Session_Maria_Garcia
```

This results in the role session having the following ARN.

```
arn:aws:iam::234567890123:assumed-role/SomeRole/Session_Maria_Garcia
```

Also, all AWS CloudTrail logs include the role session name in the information captured for each operation.

Assume Role with Web Identity

You can configure a profile to indicate that the AWS CLI should assume a role using [web identity federation and Open ID Connect \(OIDC\)](#). When you specify this in a profile, the AWS CLI automatically makes the corresponding AWS STS `AssumeRoleWithWebIdentity` call for you.

Note

When you specify a profile that uses an IAM role, the AWS CLI makes the appropriate calls to retrieve temporary credentials. These credentials are stored in `~/.aws/cli/cache`. Subsequent AWS CLI commands that specify the same profile use the cached temporary credentials until they expire. At that point, the AWS CLI automatically refreshes the credentials.

To retrieve and use temporary credentials using web identity federation, you can specify the following configuration values in a shared profile.

[role_arn](#) (p. 63)

Specifies the ARN of the role to assume.

`web_identity_token_file`

Specifies the path to a file which contains an OAuth 2.0 access token or OpenID Connect ID token that is provided by the identity provider. The AWS CLI loads this file and passes its content as the `WebIdentityToken` argument of the `AssumeRoleWithWebIdentity` operation.

[role_session_name](#) (p. 67)

Specifies an optional name applied to this assume-role session.

The following is an example of the minimal amount of configuration needed to configure an assume role with web identity profile.

```
# In ~/.aws/config

[profile web-identity]
role_arn=arn:aws:iam:123456789012:role/RoleNameToAssume
web_identity_token_file=/path/to/a/token
```

You can also provide this configuration by using [environment variables](#) (p. 56).

`AWS_ROLE_ARN`

The ARN of the role to assume.

`AWS_WEB_IDENTITY_TOKEN_FILE`

The path to the web identity token file.

`AWS_ROLE_SESSION_NAME`

The name applied to this assume-role session.

Note

These environment variables currently apply only to the assume role with web identity provider. They don't apply to the general assume role provider configuration.

Clearing Cached Credentials

When you use a role, the AWS CLI caches the temporary credentials locally until they expire. The next time you try to use them, the AWS CLI attempts to renew them on your behalf.

If your role's temporary credentials are [revoked](#), they are not renewed automatically, and attempts to use them fail. However, you can delete the cache to force the AWS CLI to retrieve new credentials.

Linux or macOS

```
$ rm -r ~/.aws/cli/cache
```

Windows

```
C:\> del /s /q %UserProfile%\aws\cli\cache
```

Command Completion

On Unix-like systems, the AWS CLI includes a command-completion feature that enables you to use the **Tab** key to complete a partially typed command. On most systems, this feature isn't automatically installed, so you need to configure it manually.

To configure command completion, you must have two pieces of information: the name of the shell you're using and the location of the `aws_completer` script.

Amazon Linux

Command completion is automatically configured and enabled by default on Amazon EC2 instances that run Amazon Linux.

Sections

- [Identify Your Shell \(p. 69\)](#)
- [Locate the AWS Completer \(p. 69\)](#)
- [Add the Completer's Folder to Your Path \(p. 70\)](#)
- [Enable Command Completion \(p. 71\)](#)
- [Test Command Completion \(p. 71\)](#)

Identify Your Shell

If you're not sure which shell you're using, you can use one of the following commands to identify it.

echo \$SHELL – Show the shell's program file name. This usually matches the name of the in-use shell, unless you launched a different shell after logging in.

```
$ echo $SHELL
/bin/bash
```

ps – Show the processes running for the current user. One of them is the shell.

```
$ ps
  PID TTY          TIME CMD
 2148 pts/1    00:00:00 bash
 8756 pts/1    00:00:00 ps
```

Locate the AWS Completer

The location of the AWS completer can vary depending on the installation method used.

Package Manager – Programs such as `pip`, `yum`, `brew`, and `apt-get` typically install the AWS completer (or a symlink to it) to a standard path location. In this case, the `which` command can locate the completer for you.

If you used `pip` without the `--user` command, you might see the following path.

```
$ which aws_completer
/usr/local/aws/bin/aws_completer
```

If you used the `--user` parameter on the `pip` install command, you can typically find the completer in the `local/bin` folder under your `$HOME` folder.

```
$ which aws_completer
/home/username/.local/bin/aws_completer
```

Bundled Installer – If you used the bundled installer per the instructions in the previous section, the AWS completer is located in the `bin` subfolder of the installation directory.

```
$ ls /usr/local/aws/bin
activate
activate.csh
activate.fish
activate_this.py
aws
aws.cmd
aws_completer
...
```

If all else fails, you can use `find` to search your entire file system for the AWS completer.

```
$ find / -name aws_completer
/usr/local/aws/bin/aws_completer
```

Add the Completer's Folder to Your Path

For the AWS completer to work successfully, you must first add it to your computer's path.

1. Find your shell's profile script in your user folder. If you're not sure which shell you have, run `echo $SHELL`.

```
$ ls -a ~
.  ..  .bash_logout  .bash_profile  .bashrc  Desktop  Documents  Downloads
```

- **Bash**– `.bash_profile`, `.profile`, or `.bash_login`
 - **Zsh**– `.zshrc`
 - **Tcsh**– `.tcshrc`, `.cshrc`, or `.login`
2. Add an export command at the end of your profile script that's similar to the following example. Replace `/usr/local/aws/bin` with the folder that you discovered in the previous section.

```
export PATH=/usr/local/aws/bin:$PATH
```

3. Reload the profile into the current session to put those changes into effect. Replace `.bash_profile` with the name of the shell script you discovered in the first section.

```
$ source ~/.bash_profile
```

Enable Command Completion

Run a command to enable command completion. The command that you use depends on the shell that you're using. You can add the command to your shell's RC file to run it each time you open a new shell. In each command, replace the path `/usr/local/aws/bin` with the one found on your system in the previous section.

- **bash** – Use the built-in command `complete`.

```
$ complete -C '/usr/local/aws/bin/aws_completer' aws
```

Add the command to `~/.bashrc` to run it each time you open a new shell. Your `~/.bash_profile` should source `~/.bashrc` to ensure that the command is also run in login shells.

- **tcsh** – Complete for `tcsh` takes a word type and pattern to define the completion behavior.

```
> complete aws 'p/*`aws_completer`/'
```

Add the command to `~/.tschrc` to run it each time you open a new shell.

- **zsh** – source `bin/aws_zsh_completer.sh`.

```
% source /usr/local/aws/bin/aws_zsh_completer.sh
```

The AWS CLI uses bash compatibility autocompletion (`bashcompinit`) for `zsh` support. For more details, see the top of `aws_zsh_completer.sh`.

Add the command to `~/.zshrc` to run it each time you open a new shell.

Test Command Completion

After enabling command completion, enter a partial command and press **Tab** to see the available commands.

```
$ aws sTAB
s3          ses          sqs          sts          swf
s3api       sns          storagegateway support
```

Using the AWS CLI

This section introduces you to many of the common features and options available in the AWS Command Line Interface (AWS CLI).

Note

By default, the AWS CLI sends requests to AWS services by using HTTPS on TCP port 443. To use the AWS CLI successfully, you must be able to make outbound connections on TCP port 443.

Topics

- [Getting Help with the AWS CLI \(p. 72\)](#)
- [Command Structure in the AWS CLI \(p. 76\)](#)
- [Specifying Parameter Values for the AWS CLI \(p. 76\)](#)
- [Generate the AWS CLI Skeleton and Input Parameters from a JSON Input File \(p. 82\)](#)
- [Controlling Command Output from the AWS CLI \(p. 86\)](#)
- [Using Shorthand Syntax with the AWS CLI \(p. 95\)](#)
- [Using AWS CLI Pagination Options \(p. 97\)](#)
- [Understanding Return Codes from the AWS CLI \(p. 98\)](#)

Getting Help with the AWS CLI

You can get help with any command when using the AWS Command Line Interface (AWS CLI). To do so, simply type `help` at the end of a command name.

For example, the following command displays help for the general AWS CLI options and the available top-level commands.

```
$ aws help
```

The following command displays the available Amazon Elastic Compute Cloud (Amazon EC2) specific commands.

```
$ aws ec2 help
```

The following example displays detailed help for the Amazon EC2 `DescribeInstances` operation. The help includes descriptions of its input parameters, available filters, and what is included as output. It also includes examples showing how to type common variations of the command.

```
$ aws ec2 describe-instances help
```

The help for each command is divided into six sections:

Name

The name of the command.

```
NAME
    describe-instances -
```

Description

A description of the API operation that the command invokes.

DESCRIPTION

Describes one or more of your instances.

If you specify one or more instance IDs, Amazon EC2 returns information for those instances. If you do not specify instance IDs, Amazon EC2 returns information for all relevant instances. If you specify an instance ID that is not valid, an error is returned. If you specify an instance that you do not own, it is not included in the returned results.

...

Synopsis

The basic syntax for using the command and its options. If an option is shown in square brackets, it's optional, has a default value, or has an alternative option that you can use.

SYNOPSIS

```
describe-instances
[--dry-run | --no-dry-run]
[--instance-ids <value>]
[--filters <value>]
[--cli-input-json <value>]
[--starting-token <value>]
[--page-size <value>]
[--max-items <value>]
[--generate-cli-skeleton]
```

For example, `describe-instances` has a default behavior that describes *all* instances in the current account and AWS Region. You can optionally specify a list of `instance-ids` to describe one or more instances; `dry-run` is an optional Boolean flag that doesn't take a value. To use a Boolean flag, specify either shown value, in this case `--dry-run` or `--no-dry-run`. Likewise, `--generate-cli-skeleton` doesn't take a value. If there are conditions on an option's use, they are described in the **OPTIONS** section, or shown in the examples.

Options

A description of each of the options shown in the synopsis.

OPTIONS

```
--dry-run | --no-dry-run (boolean)
  Checks whether you have the required permissions for the action,
  without actually making the request, and provides an error response.
  If you have the required permissions, the error response is DryRun-
  Operation . Otherwise, it is UnauthorizedOperation .

--instance-ids (list)
  One or more instance IDs.

  Default: Describes all your instances.

...
```

Examples

Examples showing the usage of the command and its options. If no example is available for a command or use case that you need, request one using the feedback link on this page, or in the AWS CLI command reference on the help page for the command.

EXAMPLES

To describe an Amazon EC2 instance

Command:

```
aws ec2 describe-instances --instance-ids i-5203422c

To describe all instances with the instance type m1.small

Command:

aws ec2 describe-instances --filters "Name=instance-type,Values=m1.small"

To describe all instances with an Owner tag

Command:

aws ec2 describe-instances --filters "Name=tag-key,Values=Owner"
...
```

Output

Descriptions of each of the fields and data types included in the response from AWS.

For `describe-instances`, the output is a list of reservation objects, each of which contains several fields and objects that contain information about the instances associated with it. This information comes from the [API documentation for the reservation data type](#) used by Amazon EC2.

```
OUTPUT
Reservations -> (list)
  One or more reservations.

  (structure)
    Describes a reservation.

    ReservationId -> (string)
      The ID of the reservation.

    OwnerId -> (string)
      The ID of the AWS account that owns the reservation.

    RequesterId -> (string)
      The ID of the requester that launched the instances on your
      behalf (for example, AWS Management Console or Auto Scaling).

    Groups -> (list)
      One or more security groups.

      (structure)
        Describes a security group.

        GroupName -> (string)
          The name of the security group.

        GroupId -> (string)
          The ID of the security group.

    Instances -> (list)
      One or more instances.

      (structure)
        Describes an instance.

        InstanceId -> (string)
          The ID of the instance.

        ImageId -> (string)
          The ID of the AMI used to launch the instance.
```

```
State -> (structure)
    The current state of the instance.

Code -> (integer)
    The low byte represents the state. The high byte
    is an opaque internal value and should be ignored.

...
```

When the AWS CLI renders the output into JSON, it becomes an array of reservation objects, similar to the following example.

```
{
  "Reservations": [
    {
      "OwnerId": "012345678901",
      "ReservationId": "r-4c58f8a0",
      "Groups": [],
      "RequesterId": "012345678901",
      "Instances": [
        {
          "Monitoring": {
            "State": "disabled"
          },
          "PublicDnsName": "ec2-52-74-16-12.us-west-2.compute.amazonaws.com",
          "State": {
            "Code": 16,
            "Name": "running"
          },
        },
        ...
      ]
    },
    ...
  ]
}
```

Each reservation object contains fields describing the reservation and an array of instance objects, each with its own fields (for example, `PublicDnsName`) and objects (for example, `State`) that describe it.

Windows users

You can *pipe* (|) the output of the help command to the `more` command to view the help file one page at a time. Press the space bar or **PgDn** to view more of the document, and **q** to quit.

```
C:\> aws ec2 describe-instances help | more
```

AWS CLI Documentation

The [AWS CLI Command Reference](#) also contains the help content for all AWS CLI commands. The descriptions are presented for easy navigation and viewing on mobile, tablet, or desktop screens.

Note

The help files contain links that cannot be viewed or navigated to from the command line. You can view and interact with these links by using the online [AWS CLI Command Reference](#).

API Documentation

All commands in the AWS CLI correspond to requests made to an AWS service's public API. Each service with a public API has an API reference that can be found on the service's home page on the [AWS Documentation website](#). The content for an API reference varies based on how the API is constructed and which protocol is used. Typically, an API reference contains detailed information about the operations supported by the API, the data sent to and from the service, and any error conditions that the service can report.

API Documentation Sections

- **Actions** – Detailed information on each operation and its parameters (including constraints on length or content, and default values). It lists the errors that can occur for this operation. Each operation corresponds to a subcommand in the AWS CLI.
- **Data Types** – Detailed information about structures that a command might require as a parameter, or return in response to a request.
- **Common Parameters** – Detailed information about the parameters that are shared by all of action for the service.
- **Common Errors** – Detailed information about errors that can be returned by any of the service's operations.

The name and availability of each section can vary, depending on the service.

Service-specific CLIs

Some services have a separate CLI that dates from before a single AWS CLI was created to work with all services. These service-specific CLIs have separate documentation that is linked from the service's documentation page. Documentation for service-specific CLIs does not apply to the AWS CLI.

Command Structure in the AWS CLI

The AWS Command Line Interface (AWS CLI) uses a multipart structure on the command line that must be specified in this order:

1. The base call to the `aws` program.
2. The top-level *command*, which typically corresponds to an AWS service supported by the AWS CLI.
3. The *subcommand* that specifies which operation to perform.
4. General CLI options or parameters required by the operation. You can specify these in any order as long as they follow the first three parts. If an exclusive parameter is specified multiple times, only the *last value* applies.

```
$ aws <command> <subcommand> [options and parameters]
```

Parameters can take various types of input values, such as numbers, strings, lists, maps, and JSON structures. What is supported is dependent upon the command and subcommand you specify.

Specifying Parameter Values for the AWS CLI

Many parameters used in the AWS Command Line Interface (AWS CLI) are simple string or numeric values, such as the key-pair name `my-key-pair` in the following example.

```
$ aws ec2 create-key-pair --key-name my-key-pair
```

You can surround strings without any space characters with quotation marks or not. However, you must use quotation marks around strings that include one or more space characters. Use single quotation marks (`'`) in Linux, macOS, Unix, or PowerShell. Use double quotation marks (`"`) in the Windows command prompt, as shown in the following examples.

PowerShell, Linux or macOS

```
$ aws ec2 create-key-pair --key-name 'my key pair'
```

Windows command prompt

```
C:\> aws ec2 create-key-pair --key-name "my key pair"
```

Optionally, you can optionally separate the parameter name from the value with an equals sign (=) instead of a space. This is typically necessary only if the value of the parameter starts with a hyphen.

```
$ aws ec2 delete-key-pair --key-name=mykey
```

Topics

- [\(p. 77\)](#)
- [Using JSON for Parameters \(p. 78\)](#)
- [Using Quotation Marks with Strings \(p. 80\)](#)
- [Loading Parameters from a File \(p. 80\)](#)

This section describes some of the common parameter types and the typical required format. If you are having trouble formatting a parameter for a specific command, check the help by entering **help** after the command name, as shown.

```
$ aws ec2 describe-spot-price-history help
```

The help for each subcommand describes its function, options, output, and examples. The options section includes the name and description of each option with the option's parameter type in parentheses.

String – String parameters can contain alphanumeric characters, symbols, and white space from the [ASCII](#) character set. Strings that contain white space must be surrounded by quotation marks. We recommend that you don't use symbols or white space other than the standard space character because it can cause unexpected results.

Some string parameters can accept binary data from a file. See [Binary Files \(p. 82\)](#) for an example.

Timestamp – Timestamps are formatted according to the [ISO 8601](#) standard. These are sometimes referred to as "DateTime" or "Date" parameters.

```
$ aws ec2 describe-spot-price-history --start-time 2014-10-13T19:00:00Z
```

Acceptable formats include:

- **YYYY-MM-DDThh:mm:ss.sssTZD (UTC)**, for example, 2014-10-01T20:30:00.000Z
- **YYYY-MM-DDThh:mm:ss.sssTZD (with offset)**, for example, 2014-10-01T12:30:00.000-08:00
- **YYYY-MM-DD**, for example, 2014-10-01
- Unix time in seconds, for example, 1412195400. This is sometimes referred to as [Unix Epoch time](#) and represents the number of seconds since midnight, January 1, 1970 UTC.

List – One or more strings separated by spaces. If any of the string items contain a space, you must put quotation marks around that item.

```
$ aws ec2 describe-spot-price-history --instance-types m1.xlarge m1.medium
```

Boolean – Binary flag that turns an option on or off. For example, `ec2 describe-spot-price-history` has a Boolean `--dry-run` parameter that, when specified, validates the query with the service without actually running the query.

```
$ aws ec2 describe-spot-price-history --dry-run
```

The output indicates whether the command was well formed. This command also includes a `--no-dry-run` version of the parameter that you can use to explicitly indicate that the command should be run normally. Including it isn't necessary because this is the default behavior.

Integer – An unsigned, whole number.

```
$ aws ec2 describe-spot-price-history --max-items 5
```

Blob – "Binary large object". Blob parameters take a path to a local file that contains the binary data. The path should not contain any protocol identifier, such as `http://` or `file://`. The specified path is interpreted as being relative to the current working directory.

For example, the `--body` parameter for `aws s3api put-object` is a blob.

```
$ aws s3api put-object --bucket my-bucket --key testimage.png --body /tmp/image.png
```

Map – A set of key-value pairs specified in JSON or by using the CLI's [shorthand syntax \(p. 95\)](#). The following JSON example reads an item from an Amazon DynamoDB table named *my-table* with a map parameter, `--key`. The parameter specifies the primary key named *id* with a number value of *1* in a nested JSON structure.

```
$ aws dynamodb get-item --table-name my-table --key '{"id": {"N": "1"}}'

{
  "Item": {
    "name": {
      "S": "John"
    },
    "id": {
      "N": "1"
    }
  }
}
```

Using JSON for Parameters

JavaScript Object Notation (JSON) is useful for specifying complex command line parameters. For example, the following command uses shorthand notation to list all Amazon EC2 instances that have an instance type of `m1.small` or `m1.medium` that are also in the `us-west-2c` Availability Zone.

```
$ aws ec2 describe-instances --filters "Name=instance-type,Values=t2.micro,m1.medium"
"Name=availability-zone,Values=us-west-2c"
```

Instead, you can also specify the equivalent list of filters as a JSON array. Square brackets are used to create an array of JSON objects separated by commas. Each object is a comma-separated list of key-value pairs (in this example, `"Name"` and `"Values"` are both keys).

The value to the right of the `"Values"` key is itself an array. This is required, even if the array contains only one value string.

```
[
  {
    "Name": "instance-type",
    "Values": ["t2.micro", "m1.medium"]
  },
  {
    "Name": "availability-zone",
    "Values": ["us-west-2c"]
  }
]
```

The outermost brackets, however, are required only if more than one filter is specified. A single filter version of the previous command, formatted in JSON, looks like this.

```
$ aws ec2 describe-instances --filters '{"Name": "instance-type", "Values": ["t2.micro", "m1.medium"]}'
```

For some operations, you *must* format the data as JSON. For example, to pass parameters to the `--block-device-mappings` parameter in the `ec2 run-instances` command, you need to format the block device information as JSON.

Because every value is surrounded by double-quotes, JSON might also be required when values that are part of the parameter contain embedded or leading spaces.

This example shows the JSON to specify a single 20 GiB Amazon Elastic Block Store (Amazon EBS) device to be mapped at `/dev/sdb` on the launching instance.

```
{
  "DeviceName": "/dev/sdb",
  "Ebs": {
    "VolumeSize": 20,
    "DeleteOnTermination": false,
    "VolumeType": "standard"
  }
}
```

To attach multiple devices, list the objects in an array, as shown in the next example.

```
[
  {
    "DeviceName": "/dev/sdb",
    "Ebs": {
      "VolumeSize": 20,
      "DeleteOnTermination": false,
      "VolumeType": "standard"
    }
  },
  {
    "DeviceName": "/dev/sdc",
    "Ebs": {
      "VolumeSize": 10,
      "DeleteOnTermination": true,
      "VolumeType": "standard"
    }
  }
]
```

You can enter the JSON directly on the command line (see [Using Quotation Marks with Strings \(p. 80\)](#)), or save it to a file that is referenced from the command line (see [Loading Parameters from a File \(p. 80\)](#)).

When passing in large blocks of data, you might find it easier to first save the JSON to a file and then reference it from the command line. JSON data in a file is easier to read, edit, and share with others. This technique is described in a later section.

For more information about JSON, see [JSON.org](https://www.json.org/), [Wikipedia's JSON entry](https://en.wikipedia.org/wiki/JSON), and [RFC4627 - The application/json Media Type for JSON](https://tools.ietf.org/html/rfc4627).

Using Quotation Marks with Strings

The way you enter JSON-formatted parameters on the command line differs depending on your operating system.

Linux, macOS, or Unix

Use single quotation marks (' ') to enclose the JSON data structure, as in the following example.

```
$ aws ec2 run-instances --image-id ami-12345678 --  
block-device-mappings '[{"DeviceName":"/dev/sdb", "Ebs":  
{"VolumeSize":20, "DeleteOnTermination":false, "VolumeType":"standard"}}]'
```

PowerShell

PowerShell requires single quotation marks (' ') to enclose the JSON data structure, and a backslash (\) to escape each double quotation mark (") within the JSON structure, as in the following example.

```
PS C:\> aws ec2 run-instances --image-id ami-12345678 --block-device-  
mappings '[{"DeviceName\":\"/dev/sdb\", \"Ebs\":{\"VolumeSize\":20,  
\"DeleteOnTermination\":false, \"VolumeType\":\"standard\"}}]'
```

Windows Command Prompt

The Windows command prompt requires double quotation marks (" ") to enclose the JSON data structure. You must then escape (precede with a backslash [\] character) each double quotation mark (") within the JSON data structure itself, as in the following example.

```
C:\> aws ec2 run-instances --image-id ami-12345678 --block-device-  
mappings "[{\"DeviceName\":\"/dev/sdb\", \"Ebs\":{\"VolumeSize\":20,  
\"DeleteOnTermination\":false, \"VolumeType\":\"standard\"}}]"]
```

Only the outermost double quotation marks are not escaped.

If the value of a parameter is itself a JSON document, escape the quotation marks on the embedded JSON document. For example, the attribute parameter for `aws sqs create-queue` can take a `RedrivePolicy` key. The `--attributes` parameter takes a JSON document, which in turn contains `RedrivePolicy`, which also takes a JSON document as its value. The inner JSON embedded in the outer JSON must be escaped.

```
$ aws sqs create-queue --queue-name my-queue --  
attributes '{ "RedrivePolicy": "{\"deadLetterTargetArn\":\"arn:aws:sqs:us-  
west-2:0123456789012:deadletter\", \"maxReceiveCount\":5}" }'
```

Loading Parameters from a File

Some parameters expect file names as arguments, from which the AWS CLI loads the data. Other parameters enable you to specify the parameter value as either text typed on the command line or read

from a file. Whether a file is required or optional, you must encode the file correctly so that the AWS CLI can understand it. The file's encoding must match the reading system's default locale. You can be determine this by using the Python `locale.getpreferredencoding()` method.

Note

By default, Windows PowerShell outputs text as UTF-16, which conflicts with the UTF-8 encoding used by many Linux systems. We recommend that you use `-Encoding ascii` with your PowerShell `Out-File` commands to ensure the AWS CLI can read the resulting file.

Sometimes it's convenient to load a parameter value from a file instead of trying to type it all as a command line parameter value, such as when the parameter is a complex JSON string. To specify a file that contains the value, specify a file URL in the following format.

```
file://complete/path/to/file
```

The first two slash '/' characters are part of the specification. If the required path begins with a '/', the result is three slash characters: `file:///folder/file`.

The URL provides the path to the file that contains the actual parameter content.

Note

This behavior is disabled automatically for parameters that already expect a URL, such as parameter that identifies a AWS CloudFormation template URL. You can also disable this behavior by adding the following line to your CLI configuration file.

```
cli_follow_urlparam = false
```

The file paths in the following examples are interpreted to be relative to the current working directory.

Linux or macOS

```
// Read from a file in the current directory
$ aws ec2 describe-instances --filters file://filter.json

// Read from a file in /tmp
$ aws ec2 describe-instances --filters file:///tmp/filter.json
```

Windows

```
// Read from a file in C:\temp
C:\> aws ec2 describe-instances --filters file://C:\temp\filter.json
```

The `file://` prefix option supports Unix-style expansions, including `~/`, `./`, and `../`. On Windows, the `~/` expression expands to your user directory, stored in the `%USERPROFILE%` environment variable. For example, on Windows 10 you would typically have a user directory under `C:\Users\User Name\`.

You must still escape JSON documents that are embedded as the value of another JSON document.

```
$ aws sqs create-queue --queue-name my-queue --attributes file://attributes.json
```

attributes.json

```
{
  "RedrivePolicy": "{ \"deadLetterTargetArn\": \"arn:aws:sqs:us-west-2:0123456789012:deadletter\", \"maxReceiveCount\": \"5\" }"
}
```

Binary Files

For commands that take binary data as a parameter, specify that the data is binary content by using the `fileb://` prefix. Commands that accept binary data include:

- **aws ec2 run-instances** – `--user-data` parameter.
- **aws s3api put-object** – `--sse-customer-key` parameter.
- **aws kms decrypt** – `--ciphertext-blob` parameter.

The following example generates a binary 256-bit AES key using a Linux command line tool, and then provides it to Amazon S3 to encrypt an uploaded file server-side.

```
$ dd if=/dev/urandom bs=1 count=32 > sse.key
32+0 records in
32+0 records out
32 bytes (32 B) copied, 0.000164441 s, 195 kB/s
$ aws s3api put-object --bucket my-bucket --key test.txt --body test.txt --sse-customer-key
  fileb://sse.key --sse-customer-algorithm AES256
{
  "SSECustomerKeyMD5": "iVg8oWa8sy714+FjtesrJg==",
  "SSECustomerAlgorithm": "AES256",
  "ETag": "\"a6118e84b76cf98bf04bbe14b6045c6c\""
}
```

Remote Files

The AWS CLI also supports loading parameters from a file hosted on the internet with an `http://` or `https://` URL. The following example references a file stored in an Amazon S3 bucket. This allows you to access parameter files from any computer, but it does require that the container is publicly accessible.

```
$ aws ec2 run-instances --image-id ami-12345678 --block-device-mappings http://my-
  bucket.s3.amazonaws.com/filename.json
```

The preceding example assumes that the file `filename.json` contains the following JSON data.

```
[
  {
    "DeviceName": "/dev/sdb",
    "Ebs": {
      "VolumeSize": 20,
      "DeleteOnTermination": false,
      "VolumeType": "standard"
    }
  }
]
```

For another example referencing a file containing more complex JSON-formatted parameters, see [Attaching an IAM Managed Policy to an IAM User \(p. 118\)](#).

Generate the AWS CLI Skeleton and Input Parameters from a JSON Input File

Most of the AWS Command Line Interface (AWS CLI) commands support the ability to accept all of the parameter input from a file using the `--cli-input-json` parameter.

Those same commands helpfully provide the `--generate-cli-skeleton` to generate a file with all of the parameters that you can edit and fill in. Then you can run the command with the `--cli-input-json` parameter and point to the filled-in file.

Important

There are several AWS CLI commands that don't map directly to individual AWS API operations, such as the [aws s3 commands](#). Such commands don't support either the `--generate-cli-skeleton` or `--cli-input-json` parameters that are discussed in this topic. If you don't know whether a specific command supports these parameters, run the following command, replacing the *service* and *command* names with the ones you're interested in.

```
$ aws service command help
```

The output includes a *Synopsis* section that shows the parameters that the specified command supports.

The `--generate-cli-skeleton` parameter causes the command not to run, but instead to generate and display a parameter template that you can customize and use as input on a later command. The generated template includes all of the parameters that the command supports.

Because the AWS CLI is essentially a "wrapper" around the service's API, the skeleton file expects you to reference all parameters by their underlying API parameter name. This is likely different from the AWS CLI parameter name. For example, an AWS CLI parameter named `user-name` might map to the AWS service's API parameter named `UserName` (notice the altered capitalization and missing dash). We recommend that you use the `--generate-cli-skeleton` option to generate the template with the "correct" parameter names to avoid errors. You can also reference the API Reference Guide for the service to see the expected parameter names. You can delete any parameters from the template that are not required and for which you don't want to supply a value.

For example, if you run the following command, it generates the parameter template for the Amazon Elastic Compute Cloud (Amazon EC2) command **run-instances**.

```
$ aws ec2 run-instances --generate-cli-skeleton
{
  "DryRun": true,
  "ImageId": "",
  "MinCount": 0,
  "MaxCount": 0,
  "KeyName": "",
  "SecurityGroups": [
    ""
  ],
  "SecurityGroupIds": [
    ""
  ],
  "UserData": "",
  "InstanceType": "",
  "Placement": {
    "AvailabilityZone": "",
    "GroupName": "",
    "Tenancy": ""
  },
  "KernelId": "",
  "RamdiskId": "",
  "BlockDeviceMappings": [
    {
      "VirtualName": "",
      "DeviceName": "",
      "Ebs": {
        "SnapshotId": "",
        "VolumeSize": 0,
```



```

        "DeleteOnTermination": true,
        "VolumeType": "",
        "Iops": 0,
        "Encrypted": true
    },
    "NoDevice": ""
}
],
"Monitoring": {
    "Enabled": true
},
"SubnetId": "",
"DisableApiTermination": true,
"InstanceInitiatedShutdownBehavior": "",
"PrivateIpAddress": "",
"ClientToken": "",
"AdditionalInfo": "",
"NetworkInterfaces": [
    {
        "NetworkInterfaceId": "",
        "DeviceIndex": 0,
        "SubnetId": "",
        "Description": "",
        "PrivateIpAddress": "",
        "Groups": [
            ""
        ],
        "DeleteOnTermination": true,
        "PrivateIpAddresses": [
            {
                "PrivateIpAddress": "",
                "Primary": true
            }
        ],
        "SecondaryPrivateIpAddressCount": 0,
        "AssociatePublicIpAddress": true
    }
],
"IamInstanceProfile": {
    "Arn": "",
    "Name": ""
},
"EbsOptimized": true
}

```

To generate and use a parameter skeleton file

1. Run the command with the `--generate-cli-skeleton` parameter and direct the output to a file to save it.

```
$ aws ec2 run-instances --generate-cli-skeleton > ec2runinst.json
```

2. Open the parameter skeleton file in your text editor and remove any of the parameters that you don't need. For example, you might strip it down to the following. Be sure that the file is still valid JSON after you remove the elements you don't need.

```

{
    "DryRun": true,
    "ImageId": "",
    "KeyName": "",
    "SecurityGroups": [
        ""
    ],

```

```
"InstanceType": "",
"Monitoring": {
  "Enabled": true
}
}
```

In this example, we leave the `DryRun` parameter set to `true` to use the Amazon EC2 dry run feature. This feature lets you safely test the command without actually creating or modifying any resources.

3. Fill in the remaining values with values appropriate for your scenario. In this example, we provide the instance type, key name, security group and identifier of the Amazon Machine Image (AMI) to use. This example assumes the default AWS Region. The AMI `ami-dfc39aef` is a 64-bit Amazon Linux image hosted in the `us-west-2` Region. If you use a different Region, you must [find the correct AMI ID to use](#).

```
{
  "DryRun": true,
  "ImageId": "ami-dfc39aef",
  "KeyName": "mykey",
  "SecurityGroups": [
    "my-sg"
  ],
  "InstanceType": "t2.micro",
  "Monitoring": {
    "Enabled": true
  }
}
```

4. Run the command with the completed parameters by passing the JSON file to the `--cli-input-json` parameter using the `file://` prefix. The AWS CLI interprets the path to be relative to your current working directory, so in the following example that displays only the file name with no path, it looks for the file directly in the current working directory.

```
$ aws ec2 run-instances --cli-input-json file://ec2runinst.json
A client error (DryRunOperation) occurred when calling the RunInstances operation:
Request would have succeeded, but DryRun flag is set.
```

The dry run error indicates that the JSON is formed correctly and the parameter values are valid. If any other issues are reported in the output, fix them and repeat the previous step until the "Request would have succeeded" message is displayed.

5. Now you can set the `DryRun` parameter to `false` to disable dry run.

```
{
  "DryRun": false,
  "ImageId": "ami-dfc39aef",
  "KeyName": "mykey",
  "SecurityGroups": [
    "my-sg"
  ],
  "InstanceType": "t2.micro",
  "Monitoring": {
    "Enabled": true
  }
}
```

6. Run the command, and `run-instances` actually launches an EC2 instance and displays the details generated by the successful launch.

```
$ aws ec2 run-instances --cli-input-json file://ec2runinst.json
{
```

```
"OwnerId": "123456789012",  
"ReservationId": "r-d94a2b1",  
"Groups": [],  
"Instances": [  
...  
]
```

Controlling Command Output from the AWS CLI

This topic describes the different ways to control the output from the AWS Command Line Interface (AWS CLI).

Topics

- [How to Select the Output Format \(p. 86\)](#)
- [JSON Output Format \(p. 87\)](#)
- [Text Output Format \(p. 87\)](#)
- [Table Output Format \(p. 89\)](#)
- [How to Filter the Output with the --query Option \(p. 90\)](#)

How to Select the Output Format

The AWS CLI supports three output formats:

- JSON (`json`)
- Tab-delimited text (`text`)
- ASCII-formatted table (`table`)

As explained in the [configuration \(p. 37\)](#) topic, you can specify the output format in three ways:

- Using the `output` option in a named profile in the `config` file. The following example sets the default output format to `text`.

```
[default]  
output=text
```

- Using the `AWS_DEFAULT_OUTPUT` environment variable. The following output sets the format to `table` for the commands in this command line session until the variable is changed or the session ends. Using this environment variable overrides any value set in the `config` file.

```
$ export AWS_DEFAULT_OUTPUT="table"
```

- Using the `--output` option on the command line. The following example sets the output of only this one command to `json`. Using this option on the command overrides any currently set environment variable or the value in the `config` file.

```
$ aws swf list-domains --registration-status REGISTERED --output json
```

[AWS CLI precedence rules \(p. 39\)](#) apply. For example, using the `AWS_DEFAULT_OUTPUT` environment variable overrides any value set in the `config` file, and a value passed to an AWS CLI command with `--output` overrides any value set in the environment variable or in the `config` file.

The `json` option is best for handling the output programmatically via various languages or `jq` (a command line JSON processor).

The `table` format is easy to read.

The `text` format works well with traditional Unix text processing tools, such as `sed`, `grep`, and `awk`, and in PowerShell scripts.

You can customize and filter the results in any format by using the `--query` parameter. For more information, see [How to Filter the Output with the `--query` Option](#) (p. 90).

JSON Output Format

JSON is the default output format of the AWS CLI. Most languages can easily decode JSON strings using built-in functions or with publicly available libraries. You can combine JSON output with the `--query` option (p. 90) in powerful ways to filter and format the AWS CLI JSON-formatted output.

For more advanced filtering that you might not be able to do with `--query`, you can consider `jq`, a command line JSON processor. You can download it and find the official tutorial at <http://stedolan.github.io/jq/>.

Text Output Format

The `text` format organizes the AWS CLI output into tab-delimited lines. It works well with traditional Unix text tools such as `grep`, `sed`, and `awk`, and the text processing performed by PowerShell.

The text output format follows the basic structure shown below. The columns are sorted alphabetically by the corresponding key names of the underlying JSON object.

```
IDENTIFIER sorted-column1 sorted-column2
IDENTIFIER2 sorted-column1 sorted-column2
```

The following is an example of a text output.

```
$ aws ec2 describe-volumes --output text
VOLUMES us-west-2a 2013-09-17T00:55:03.000Z 30 snap-f23ec1c8 in-use
vol-e11a5288 standard
ATTACHMENTS 2013-09-17T00:55:03.000Z True /dev/sda1 i-a071c394
attached vol-e11a5288
VOLUMES us-west-2a 2013-09-18T20:26:15.000Z 8 snap-708e8348 in-use
vol-2e410a47 standard
ATTACHMENTS 2013-09-18T20:26:16.000Z True /dev/sda1 i-4b41a37c
attached vol-2e410a47
```

Important

We strongly recommend that if you specify `text` output, you also always use the `--query` option to ensure consistent behavior.

This is because the text format alphabetically orders output columns by the key name of the underlying JSON object, and similar resources might not have the same key names. For example, the JSON representation of a Linux-based EC2 instance might have elements that are not present in the JSON representation of a Windows-based instance, or vice versa. Also, resources might have key-value elements added or removed in future updates, altering the column ordering. This is where `--query` augments the functionality of the `text` output to provide you with complete control over the output format.

In the following example, the command specifies which elements to display and *defines the ordering* of the columns with the list notation `[key1, key2, ...]`. This gives you full

confidence that the correct key values are always displayed in the expected column. Finally, notice how the AWS CLI outputs `None` as values for keys that don't exist.

```
$ aws ec2 describe-volumes --query 'Volumes[*].[VolumeId,
Attachments[0].InstanceId, AvailabilityZone, Size, FakeKey]' --output text
vol-e11a5288    i-a071c394    us-west-2a    30    None
vol-2e410a47    i-4b41a37c    us-west-2a    8    None
```

The following example show how you can use `grep` and `awk` with the text output from the `aws ec2 describe-instances` command. The first command displays the Availability Zone, current state, and the instance ID of each instance in text output. The second command processes that output to display only the instance IDs of all running instances in the `us-west-2a` Availability Zone.

```
$ aws ec2 describe-instances --query 'Reservations[*].Instances[*].
[Placement.AvailabilityZone, State.Name, InstanceId]' --output text
us-west-2a    running i-4b41a37c
us-west-2a    stopped i-a071c394
us-west-2b    stopped i-97a217a0
us-west-2a    running i-3045b007
us-west-2a    running i-6fc67758

$ aws ec2 describe-instances --query 'Reservations[*].Instances[*].
[Placement.AvailabilityZone, State.Name, InstanceId]' --output text | grep us-west-2a |
grep running | awk '{print $3}'
i-4b41a37c
i-3045b007
i-6fc67758
```

The following example goes a step further and shows not only how to filter the output, but how to use that output to automate changing instance types for each stopped instance.

```
$ aws ec2 describe-instances --query 'Reservations[*].Instances[*].[State.Name,
InstanceId]' --output text |
> grep stopped |
> awk '{print $2}' |
> while read line;
> do aws ec2 modify-instance-attribute --instance-id $line --instance-type '{"Value":
"m1.medium"}';
> done
```

The text output can also be useful in PowerShell. Because the columns in text output are tab-delimited, you can easily split the output into an array by using PowerShell's ``t` delimiter. The following command displays the value of the third column (`InstanceId`) if the first column (`AvailabilityZone`) matches the string `us-west-2a`.

```
PS C:\>aws ec2 describe-instances --query 'Reservations[*].Instances[*].
[Placement.AvailabilityZone, State.Name, InstanceId]' --output text |
%{if ($_.split("`t")[0] -match "us-west-2a") { $_.split("`t")[2]; } }
i-4b41a37c
i-a071c394
i-3045b007
i-6fc67758
```

Notice that although the previous example does show how to use the `--query` parameter to parse the underlying JSON objects and pull out the desired column, PowerShell has its own ability to handle JSON that you can use if cross-platform compatibility isn't a concern. Instead of handling the output as text, as most command shells require, PowerShell lets you use the `ConvertFrom-JSON` cmdlet to produce a hierarchically structured object. You can then directly access the member you want from that object.

```
(aws ec2 describe-instances --output json | ConvertFrom-
Json).Reservations.Instances.InstanceId
```

Tip

If you output text, and filter the output to a single field using the `--query` parameter, the output is a single line of tab-separated values. To get each value onto a separate line, you can put the output field in brackets, as shown in the following examples.

Tab separated, single-line output:

```
$ aws iam list-groups-for-user --user-name susan --output text --query
"Groups[.GroupName]"
HRDepartment      Developers      SpreadsheetUsers  LocalAdmins
```

Each value on its own line by putting `[GroupName]` in brackets:

```
$ aws iam list-groups-for-user --user-name susan --output text --query
"Groups[.GroupName]"
HRDepartment
Developers
SpreadsheetUsers
LocalAdmins
```

Table Output Format

The table format produces human-readable representations of complex AWS CLI output in a tabular form.

```
$ aws ec2 describe-volumes --output table
```

DescribeVolumes						
Volumes						
AvailabilityZone	CreateTime	Size	SnapshotId	State		
VolumeId	VolumeType					
us-west-2a	2013-09-17T00:55:03.000Z	30	snap-f23ec1c8	in-use	vol-e11a5288 standard	
Attachments						
AttachTime	DeleteOnTermination	Device	InstanceId	State	VolumeId	
2013-09-17T00:55:03.000Z	True	/dev/sda1	i-a071c394	attached	vol-e11a5288	
Volumes						

```

|+-----+-----+-----+-----+-----+
|+-----+-----+|
|| AvailabilityZone | CreateTime | Size | SnapshotId | State |
| VolumeId | VolumeType ||
|+-----+-----+-----+-----+-----+
|+-----+-----+|
|| us-west-2a | 2013-09-18T20:26:15.000Z | 8 | snap-708e8348 | in-use |
| vol-2e410a47 | standard ||
|+-----+-----+-----+-----+-----+
|+-----+-----+|
||| Attachments
|||
|+-----+-----+-----+-----+-----+
|+-----+-----+|
||| AttachTime | DeleteOnTermination | Device | InstanceId |
| State | VolumeId |||
|+-----+-----+-----+-----+-----+
|+-----+-----+|
||| 2013-09-18T20:26:16.000Z | True | /dev/sda1 | i-4b41a37c |
| attached | vol-2e410a47 |||
|+-----+-----+-----+-----+-----+
|+-----+-----+|

```

You can combine the `--query` option with the `table` format to display a set of elements preselected from the raw output. Notice the output differences between dictionary and list notations: column names are alphabetically ordered in the first example, and unnamed columns are ordered as defined by the user in the second example. For more information about the `--query` option, see [How to Filter the Output with the --query Option \(p. 90\)](#).

```

$ aws ec2 describe-volumes --query 'Volumes[*].
{ID:VolumeId,InstanceId:Attachments[0].InstanceId,AZ:AvailabilityZone,Size:Size}' --output
table
-----
| DescribeVolumes |
|+-----+-----+-----+-----+
| AZ | ID | InstanceId | Size |
|+-----+-----+-----+-----+
| us-west-2a | vol-e11a5288 | i-a071c394 | 30 |
| us-west-2a | vol-2e410a47 | i-4b41a37c | 8 |
|+-----+-----+-----+-----+

$ aws ec2 describe-volumes --query 'Volumes[*].
[VolumeId,Attachments[0].InstanceId,AvailabilityZone,Size]' --output table
-----
| DescribeVolumes |
|+-----+-----+-----+-----+
| vol-e11a5288 | i-a071c394 | us-west-2a | 30 |
| vol-2e410a47 | i-4b41a37c | us-west-2a | 8 |
|+-----+-----+-----+-----+

```

How to Filter the Output with the --query Option

The AWS CLI provides built-in JSON-based output filtering capabilities with the `--query` option. The `--query` parameter accepts strings that are compliant with the [JMESPath specification](#).

Important

The output type you specify (`json`, `text`, or `table`) impacts how the `--query` option operates.

- If you specify `--output text`, the output is paginated *before* the `--query` filter is applied and the AWS CLI runs the query once on *each page* of the output. This can result in unexpected extra output, especially if your filter specifies an array element using something like `[0]`, because the output then includes the first matching element on each page. To work

around the extra output that can be produced if you use `--output text`, you can specify `--no-paginate`. This causes the filter to apply only to the complete set of results. But it does remove any pagination, so it might result in long output. You can also use other command line tools such as `head` or `tail` to additionally filter the output to only the values you want.

- If you specify `--output json`, the output is completely processed and converted into a JSON structure before the `--query` filter is applied. The AWS CLI runs the query only once against the entire output.

To demonstrate how `--query` works, we first start with the following default JSON output. This describes two Amazon Elastic Block Store (Amazon EBS) volumes attached to separate Amazon EC2 instances.

```
$ aws ec2 describe-volumes
{
  "Volumes": [
    {
      "AvailabilityZone": "us-west-2a",
      "Attachments": [
        {
          "AttachTime": "2013-09-17T00:55:03.000Z",
          "InstanceId": "i-a071c394",
          "VolumeId": "vol-e11a5288",
          "State": "attached",
          "DeleteOnTermination": true,
          "Device": "/dev/sda1"
        }
      ],
      "VolumeType": "standard",
      "VolumeId": "vol-e11a5288",
      "State": "in-use",
      "SnapshotId": "snap-f23ec1c8",
      "CreateTime": "2013-09-17T00:55:03.000Z",
      "Size": 30
    },
    {
      "AvailabilityZone": "us-west-2a",
      "Attachments": [
        {
          "AttachTime": "2013-09-18T20:26:16.000Z",
          "InstanceId": "i-4b41a37c",
          "VolumeId": "vol-2e410a47",
          "State": "attached",
          "DeleteOnTermination": true,
          "Device": "/dev/sda1"
        }
      ],
      "VolumeType": "standard",
      "VolumeId": "vol-2e410a47",
      "State": "in-use",
      "SnapshotId": "snap-708e8348",
      "CreateTime": "2013-09-18T20:26:15.000Z",
      "Size": 8
    }
  ]
}
```

You can choose to display only the first volume from the `volumes` list by using the following command that [indexes the first volume in the array](#).

```
$ aws ec2 describe-volumes --query 'Volumes[0]'
{
```



```
"AvailabilityZone": "us-west-2a",
"Attachments": [
  {
    "AttachTime": "2013-09-17T00:55:03.000Z",
    "InstanceId": "i-a071c394",
    "VolumeId": "vol-e11a5288",
    "State": "attached",
    "DeleteOnTermination": true,
    "Device": "/dev/sda1"
  }
],
"VolumeType": "standard",
"VolumeId": "vol-e11a5288",
"State": "in-use",
"SnapshotId": "snap-f23ec1c8",
"CreateTime": "2013-09-17T00:55:03.000Z",
"Size": 30
}
```

The next example uses the [wildcard notation](#) `[*]` to iterate over all of the volumes in the list, filtering out three elements from each volume: `VolumeId`, `AvailabilityZone`, and `Size`. The dictionary notation requires that you provide an alias for each JSON key, like this: `{Alias1:JSONKey1, Alias2:JSONKey2}`. A dictionary is inherently *unordered*, so the ordering of the keys/aliases within a structure might be inconsistent.

```
$ aws ec2 describe-volumes --query 'Volumes[*].{ID:VolumeId,AZ:AvailabilityZone,Size:Size}'
[
  {
    "AZ": "us-west-2a",
    "ID": "vol-e11a5288",
    "Size": 30
  },
  {
    "AZ": "us-west-2a",
    "ID": "vol-2e410a47",
    "Size": 8
  }
]
```

Using dictionary notation, you can also chain keys together, like `key1.key2[0].key3`, to filter elements deeply nested within the structure. The following example demonstrates this with the `Attachments[0].InstanceId` key, aliased to simply `InstanceId`.

```
$ aws ec2 describe-volumes --query 'Volumes[*].{ID:VolumeId,InstanceId:Attachments[0].InstanceId,AZ:AvailabilityZone,Size:Size}'
[
  {
    "InstanceId": "i-a071c394",
    "AZ": "us-west-2a",
    "ID": "vol-e11a5288",
    "Size": 30
  },
  {
    "InstanceId": "i-4b41a37c",
    "AZ": "us-west-2a",
    "ID": "vol-2e410a47",
    "Size": 8
  }
]
```

You can also filter multiple elements using list notation: `[key1, key2]`. This formats all filtered attributes into a single *ordered* list per object, regardless of type.

```
$ aws ec2 describe-volumes --query 'Volumes[*].[VolumeId, Attachments[0].InstanceId, AvailabilityZone, Size]'
[
  [
    "vol-e11a5288",
    "i-a071c394",
    "us-west-2a",
    30
  ],
  [
    "vol-2e410a47",
    "i-4b41a37c",
    "us-west-2a",
    8
  ]
]
```

To filter results by the value of a specific field, use the [JMESPath "?" operator](#). The following example query outputs only volumes in the us-west-2a Availability Zone.

```
$ aws ec2 describe-volumes \
  --query 'Volumes[?AvailabilityZone==`us-west-2a`]'
```

Note

When specifying a literal value such as "us-west-2" above in a JMESPath query expression, you must surround the value in backticks (` `) for it to be read properly.

Here are some additional examples that illustrate how you can get only the details you want from the output of your commands.

The following example lists Amazon EC2 volumes. The service produces a list of all attached volumes in the us-west-2a Availability Zone. The --query parameter further limits the output to only those volumes with a Size value that is larger than 50, and shows only the specified fields with user-defined names.

```
$ aws ec2 describe-volumes \
  --filters "Name=availability-zone,Values=us-west-2a" "Name=status,Values=attached" \
  --query 'Volumes[?Size > `50`].{Id:VolumeId,Size:Size,Type:VolumeType}'
[
  {
    "Id": "vol-0be9bb0bf12345678",
    "Size": 80,
    "Type": "gp2"
  }
]
```

The following example retrieves a list of images that meet several criteria. It then uses the --query parameter to sort the output by CreationDate, selecting only the most recent. Finally, it displays the ImageId of that one image.

```
$ aws ec2 describe-images \
  --owners amazon \
  --filters "Name=name,Values=amzn*gp2" "Name=virtualization-type,Values=hvm" "Name=root-device-type,Values=ebs" \
  --query "sort_by(Images, &CreationDate)[-1].ImageId" \
  --output text
ami-00ced3122871a4921
```

The following example uses the --query parameter to find a specific item in a list and then extracts information from that item. The example lists all of the Availability Zones associated with the

specified service endpoint. It extracts the item from the `ServiceDetails` list that has the specified `ServiceName`, then outputs the `AvailabilityZones` field from that selected item.

```
$ aws --region us-east-1 ec2 describe-vpc-endpoint-services \
  --query 'ServiceDetails[?ServiceName==`com.amazonaws.us-east-1.ecs`].AvailabilityZones'
[
  [
    "us-east-1a",
    "us-east-1b",
    "us-east-1c",
    "us-east-1d",
    "us-east-1e",
    "us-east-1f"
  ]
]
```

The `--query` parameter also enables you to count items in the output. The following example displays the number of available volumes that are more than 1000 IOPS by using `length` to count how many in a list.

```
$ aws ec2 describe-volumes \
  --filters "Name=status,Values=available" \
  --query 'length(Volumes[?Iops > `1000`])'
3
```

The following example shows how to list all of your snapshots that were created after a specified date, including only a few of the available fields in the output.

```
$ aws ec2 describe-snapshots --owner self \
  --output json \
  --query 'Snapshots[?StartTime>=`2018-02-07`].
{Id:SnapshotId,Vid:VolumeId,Size:VolumeSize}' \
[
  {
    "id": "snap-0effb42b7a1b2c3d4",
    "vid": "vol-0be9bb0bf12345678",
    "Size": 8
  }
]
```

The following example lists the five most recent Amazon Machine Images (AMIs) that you created, sorted from most recent to oldest.

```
$ aws ec2 describe-images \
  --owners self \
  --query 'reverse(sort_by(Images,&CreationDate))[ :5 ].{id:ImageId,date:CreationDate}'
[
  {
    "id": "ami-0a1b2c3d4e5f60001",
    "date": "2018-11-28T17:16:38.000Z"
  },
  {
    "id": "ami-0a1b2c3d4e5f60002",
    "date": "2018-09-15T13:51:22.000Z"
  },
  {
    "id": "ami-0a1b2c3d4e5f60003",
    "date": "2018-08-19T10:22:45.000Z"
  },
  {
    "id": "ami-0a1b2c3d4e5f60004",

```

```
    "date": "2018-05-03T12:04:02.000Z"
  },
  {
    "id": "ami-0a1b2c3d4e5f60005",
    "date": "2017-12-13T17:16:38.000Z"
  }
]
```

The following example shows only the `InstanceId` for any unhealthy instances in the specified Auto Scaling group.

```
$ aws autoscaling describe-auto-scaling-groups \
  --auto-scaling-group-name My-AutoScaling-Group-Name \
  --output text \
  --query 'AutoScalingGroups[*].Instances[?HealthStatus==`Unhealthy`].InstanceId'
```

Combined with the three output formats that are explained in more detail in the following sections, the `--query` option is a powerful tool you can use to customize the content and style of outputs.

For more examples and the full spec of JMESPath, the underlying JSON-processing library, see <http://jmespath.org/specification.html>.

Using Shorthand Syntax with the AWS CLI

The AWS Command Line Interface (AWS CLI) can accept many of its option parameters in JSON format. However, it can be tedious to enter large JSON lists or structures on the command line. To make this easier, the AWS CLI also supports a shorthand syntax that enables a simpler representation of your option parameters than using the full JSON format.

Structure Parameters

The shorthand syntax in the AWS CLI makes it easier for users to input parameters that are flat (non-nested structures). The format is a comma-separated list of key-value pairs.

Linux or macOS

```
--option key1=value1,key2=value2,key3=value3
```

PowerShell

```
--option "key1=value1,key2=value2,key3=value3"
```

These are both equivalent to the following example, formatted in JSON.

```
--option '{"key1":"value1","key2":"value2","key3":"value3"}'
```

There must be no white space between each comma-separated key-value pair. Here is an example of the Amazon DynamoDB `update-table` command with the `--provisioned-throughput` option specified in shorthand.

```
$ aws dynamodb update-table \
  --provisioned-throughput ReadCapacityUnits=15,WriteCapacityUnits=10 \
```

```
--table-name MyDDBTable
```

This is equivalent to the following example formatted in JSON.

```
$ aws dynamodb update-table \  
  --provisioned-throughput '{"ReadCapacityUnits':15,"WriteCapacityUnits":10}' \  
  --table-name MyDDBTable
```

Using Shorthand Syntax with the AWS Command Line Interface

You can specify Input parameters in a list form in two ways: JSON or shorthand. The AWS CLI shorthand syntax is designed to make it easier to pass in lists with number, string, or non-nested structures.

The basic format is shown here, where values in the list are separated by a single space.

```
--option value1 value2 value3
```

This is equivalent to the following example, formatted in JSON.

```
--option '[value1,value2,value3]'
```

As previously mentioned, you can specify a list of numbers, a list of strings, or a list of non-nested structures in shorthand. The following is an example of the `stop-instances` command for Amazon Elastic Compute Cloud (Amazon EC2), where the input parameter (list of strings) for the `--instance-ids` option is specified in shorthand.

```
$ aws ec2 stop-instances \  
  --instance-ids i-1486157a i-1286157c i-ec3a7e87
```

This is equivalent to the following example formatted in JSON.

```
$ aws ec2 stop-instances \  
  --instance-ids '["i-1486157a","i-1286157c","i-ec3a7e87"]'
```

The following example shows the Amazon EC2 `create-tags` command, which takes a list of non-nested structures for the `--tags` option. The `--resources` option specifies the ID of the instance to tag.

```
$ aws ec2 create-tags \  
  --resources i-1286157c \  
  --tags Key=My1stTag,Value=Value1 Key=My2ndTag,Value=Value2 Key=My3rdTag,Value=Value3
```

This is equivalent to the following example, formatted in JSON. The JSON parameter is written over multiple lines for readability.

```
$ aws ec2 create-tags \  
  --resources i-1286157c \  
  --tags '[  
    {"Key": "My1stTag", "Value": "Value1"},  
    {"Key": "My2ndTag", "Value": "Value2"},  
    {"Key": "My3rdTag", "Value": "Value3"}  
  ]'
```

J'

Using AWS CLI Pagination Options

For commands that can return a large list of items, the AWS Command Line Interface (AWS CLI) has three options to control the number of items included in the output when the AWS CLI calls a service's API to populate the list.

By default, the AWS CLI uses a page size of 1000 and retrieves all available items. For example, if you run `aws s3api list-objects` on an Amazon S3 bucket that contains 3,500 objects, the AWS CLI makes four calls to Amazon S3, handling the service-specific pagination logic for you in the background and returning all 3,500 objects in the final output.

If you see issues when running list commands on a large number of resources, the default page size of 1000 might be too high. This can cause calls to AWS services to exceed the maximum allowed time and generate a "timed out" error. You can use the `--page-size` option to specify that the AWS CLI request a smaller number of items from each call to the AWS service. The CLI still retrieves the full list, but performs a larger number of service API calls in the background and retrieves a smaller number of items with each call. This gives the individual calls a better chance of succeeding without a timeout. Changing the page size doesn't affect the output; it affects only the number of API calls that need to be made to generate the output.

```
$ aws s3api list-objects \
  --bucket my-bucket \
  --page-size 100
{
  "Contents": [
  ...
```

To include fewer items at a time in the AWS CLI output, use the `--max-items` option. The AWS CLI still handles pagination with the service as described previously, but prints out only the number of items at a time that you specify.

```
$ aws s3api list-objects \
  --bucket my-bucket \
  --max-items 100
{
  "NextToken": "eyJNYXJrZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxfQ==",
  "Contents": [
  ...
```

If the number of items output (`--max-items`) is fewer than the total number of items returned by the underlying API calls, the output includes a `NextToken` that you can pass to a subsequent command to retrieve the next set of items. The following example shows how to use the `NextToken` value returned by the previous example, and enables you to retrieve the second 100 items.

Note

The parameter `--starting-token` cannot be null or empty. If the previous command does not return a `NextToken` value, there are no more items to return and you do not need to call the command again.

```
$ aws s3api list-objects \
  --bucket my-bucket \
  --max-items 100 \
  --starting-token eyJNYXJrZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxfQ==
{
  "Contents": [
```

...

The specified AWS service might not return items in the same order each time you call. If you specify different values for `--page-size` and `--max-items`, you can get unexpected results with missing or duplicated items. To prevent this, use the same number for `--page-size` and `--max-items` to sync the AWS CLI pagination with the pagination of the underlying service. You can also retrieve the whole list and perform any necessary paging operations locally.

Understanding Return Codes from the AWS CLI

To determine the return code of an AWS Command Line Interface (AWS CLI) command, run one of the following commands immediately after running the CLI command.

Linux/Unix/Mac systems

```
$ echo $?
```

Windows PowerShell

```
PS> echo $lastexitcode
```

Windows Command Prompt

```
C:\> echo %errorlevel%
```

The following are the return code values that can be returned at the end of running an AWS Command Line Interface (AWS CLI) command.

Cc	Meaning
0	The command completed successfully. There were no errors generated by either the AWS CLI or by the AWS service to which the request was sent.
1	One or more Amazon S3 transfer operations failed. <i>Limited to S3 commands.</i>
2	The meaning of this return code depends on the command: <ul style="list-style-type: none"> <i>Applicable to all CLI commands</i> – the command entered on the command line couldn't be parsed. Parsing failures can be caused by, but aren't limited to, missing required subcommands or arguments, or using unknown commands or arguments. <i>Limited to S3 commands.</i> – One or more files marked for transfer were skipped during the transfer process. However, all other files marked for transfer were successfully transferred. Files that are skipped during the transfer process include: files that don't exist; files that are character special devices, block special device, FIFO queues, or sockets; and files that the user doesn't have read permissions to.
130	The command was interrupted by a SIGINT (Ctrl+C).
255	The command failed. There were errors generated by the AWS CLI or by the AWS service to which the request was sent.

For more details about a failure, run the command with the `--debug` switch. This produces a detailed report of the steps the AWS CLI uses to process the command, and what the result of each step was.

Using the AWS CLI to Work with AWS Services

This section provides examples that show how to use the AWS Command Line Interface (AWS CLI) to access various AWS services.

For a complete reference of all the available commands for each service, see the [AWS CLI Command Reference](#), or use the built-in command line help. For more information, see [Getting Help with the AWS CLI](#) (p. 72).

Topics

- [Using Amazon DynamoDB with the AWS CLI](#) (p. 99)
- [Using Amazon EC2 with the AWS CLI](#) (p. 101)
- [Using Amazon S3 Glacier with the AWS CLI](#) (p. 113)
- [Using AWS Identity and Access Management from the AWS CLI](#) (p. 116)
- [Using Amazon S3 with the AWS CLI](#) (p. 120)
- [Using Amazon SNS with the AWS CLI](#) (p. 126)
- [Using Amazon SWF with the AWS CLI](#) (p. 128)

Using Amazon DynamoDB with the AWS CLI

The AWS Command Line Interface (AWS CLI) provides support for all of the AWS database services, including Amazon DynamoDB. You can use the AWS CLI for ad hoc operations, such as creating a table. You can also use it to embed DynamoDB operations within utility scripts.

To list the AWS CLI commands for DynamoDB, use the following command.

```
$ aws dynamodb help
```

Before you run any commands, set your default credentials. For more information, see [Configuring the AWS CLI](#) (p. 37).

The command line format consists of an DynamoDB command name, followed by the parameters for that command. The AWS CLI supports the CLI [shorthand syntax](#) (p. 95) for the parameter values, and full JSON.

For example, the following command creates a table named `MusicCollection`.

Note

For readability, long commands in this section are broken into separate lines. The backslash (\) character is the line continuation character for the Linux command line, and lets you copy and paste (or enter) multiple lines at a Linux prompt. If you're using a shell that doesn't use the backslash for line continuation, replace the backslash with that shell's line continuation character. Or remove the backslashes and put the entire command on a single line.


```
$ aws dynamodb create-table \  
  --table-name MusicCollection \  
  --attribute-definitions AttributeName=Artist,AttributeType=S  
  AttributeName=SongTitle,AttributeType=S \  
  --key-schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \  
  --provisioned-throughput ReadCapacityUnits=1,WriteCapacityUnits=1
```

You can add new lines to the table with commands similar to those shown in the following example. These examples use a combination of shorthand syntax and JSON.

```
$ aws dynamodb put-item \  
  --table-name MusicCollection \  
  --item '{  
    "Artist": {"S": "No One You Know"},  
    "SongTitle": {"S": "Call Me Today"} ,  
    "AlbumTitle": {"S": "Somewhat Famous"}  
  }' \  
  --return-consumed-capacity TOTAL  
{  
  "ConsumedCapacity": {  
    "CapacityUnits": 1.0,  
    "TableName": "MusicCollection"  
  }  
}  
  
$ aws dynamodb put-item \  
  --table-name MusicCollection \  
  --item '{  
    "Artist": {"S": "Acme Band"},  
    "SongTitle": {"S": "Happy Day"} ,  
    "AlbumTitle": {"S": "Songs About Life"}  
  }' \  
  --return-consumed-capacity TOTAL  
{  
  "ConsumedCapacity": {  
    "CapacityUnits": 1.0,  
    "TableName": "MusicCollection"  
  }  
}
```

It can be difficult to compose valid JSON in a single-line command. To make this easier, the AWS CLI can read JSON files. For example, consider the following JSON snippet, which is stored in a file named `expression-attributes.json`.

```
{  
  ":v1": {"S": "No One You Know"},  
  ":v2": {"S": "Call Me Today"}  
}
```

You can use that file to issue a query request using the AWS CLI. In the following example, the content of the `expression-attributes.json` file is used as the value for the `--expression-attribute-values` parameter.

```
$ aws dynamodb query --table-name MusicCollection \  
  --key-condition-expression "Artist = :v1 AND SongTitle = :v2" \  
  --expression-attribute-values file://expression-attributes.json  
{  
  "Count": 1,  
  "Items": [  
    {  
      "Artist": {"S": "No One You Know"},  
      "SongTitle": {"S": "Call Me Today"},  
      "AlbumTitle": {"S": "Somewhat Famous"}  
    }  
  ]  
}
```

```
{
  "AlbumTitle": {
    "S": "Somewhat Famous"
  },
  "SongTitle": {
    "S": "Call Me Today"
  },
  "Artist": {
    "S": "No One You Know"
  }
},
"ScannedCount": 1,
"ConsumedCapacity": null
}
```

For more information about using the AWS CLI with DynamoDB, see [DynamoDB](#) in the *AWS CLI Command Reference*.

In addition to DynamoDB, you can use the AWS CLI with DynamoDB Local. DynamoDB Local is a small client-side database and server that mimics the DynamoDB service. DynamoDB Local enables you to write applications that use the DynamoDB API, without manipulating any tables or data in the DynamoDB web service. Instead, all of the API actions are rerouted to a local database. This lets you save on provisioned throughput, data storage, and data transfer fees.

For more information about DynamoDB Local and how to use it with the AWS CLI, see the following sections of the [Amazon DynamoDB Developer Guide](#):

- [DynamoDB Local](#)
- [Using the AWS CLI with DynamoDB Local](#)

Using Amazon EC2 with the AWS CLI

You can access the features of Amazon Elastic Compute Cloud (Amazon EC2) using the AWS Command Line Interface (AWS CLI). To list the AWS CLI commands for Amazon EC2, use the following command.

```
aws ec2 help
```

Before you run any commands, set your default credentials. For more information, see [Configuring the AWS CLI](#) (p. 37).

This topic shows examples of AWS CLI commands that perform common tasks for Amazon EC2.

Topics

- [Creating, Displaying, and Deleting Amazon EC2 Key Pairs](#) (p. 101)
- [Creating, Configuring, and Deleting Security Groups for Amazon EC2](#) (p. 103)
- [Launching, Listing, and Terminating Amazon EC2 Instances](#) (p. 107)

Creating, Displaying, and Deleting Amazon EC2 Key Pairs

You can use the AWS Command Line Interface (AWS CLI) to create, display, and delete your key pairs for Amazon Elastic Compute Cloud (Amazon EC2). You use key pairs to connect to an Amazon EC2 instance.

You must provide the key pair to Amazon EC2 when you create the instance, and then use that key pair to authenticate when you connect to the instance.

Note

The following examples assume that you have already [configured your default credentials](#) (p. 101).

Topics

- [Create a Key Pair](#) (p. 102)
- [Display Your Key Pair](#) (p. 103)
- [Delete Your Key Pair](#) (p. 103)

Create a Key Pair

To create a key pair, use the `create-key-pair` command with the `--query` option, and the `--output` text option to pipe your private key directly into a file.

```
$ aws ec2 create-key-pair --key-name MyKeyPair --query 'KeyMaterial' --output text  
> MyKeyPair.pem
```

For PowerShell, the `> file` redirection defaults to UTF-8 encoding, which cannot be used with some SSH clients. So, you must convert the output by piping it to the `out-file` command and explicitly set the encoding to `ascii`.

```
PS C:\>aws ec2 create-key-pair --key-name MyKeyPair --query 'KeyMaterial' --output text |  
out-file -encoding ascii -filepath MyKeyPair.pem
```

The resulting `MyKeyPair.pem` file looks similar to the following.

```
-----BEGIN RSA PRIVATE KEY-----  
EXAMPLEKEYKCAQEay7WZhaDsra1W3mRlQtvhwYORRX8gnxgDAfRt/gx42kWXst4rXE/b5CpSgie/  
vBoU7jLxx92pNHofnByP+Dc21eyyz6CvjTmWA0JwfWiW5/akH7iO5dSrvC7dQkW2duV5QuUdEOQW  
Z/aNxMniGQE6XAgfwlnXVBwrerrrQo+ZWQeqiUwWmkuEbLeJfLhMCvYURpUMSCloehm449ilx9X1F  
G50TCFeOzf18dqqCP6GzbPaIjiU19xX/azOR9V+tpUOzEL+wmXnZt3/nHPQ5xvD20JH67km6SuPW  
oPzev/D8V+x4+bHthfSjR9Y7DvQFjfBVwHXigBdtZcU2/wei8D/HYwIDAQABaoIBAGZ1kaEvnqrq  
/uler7vgIn5m7lN5Lk4hJLAIW6tUT/fzvtCHK0SkbQCQXuriHmQ2MqyJX/0kn2NfjLV/ufGxbLl  
mb5qwmGUnEpJaZD6QSSs3kICLwWUYUiGfc0uiSbmJoap/GTLU0W5Mfcv36PaBUNy5p53V6G7hXb2  
bahyWyJNfjLe4M86yd2YK3V2CmK+X/BosShnJ36+hjrXPPWmV3N9zEmCdJJA+K15DYmhm/tJWSD9  
81oGk9TopEp7CkIfatEATyyZiVqoRq6k64iuM9JkA3OzdXzMqexXVJ1TLZVEH0E7bh1Y9d801ozR  
oQs/FiZNAx2iijCWyv0lpjE73+kCgYEA9mZtyhkHkFDpwrSM1APaL8oNAbbjwEy7Z5Mqfql+lIp1  
YkriL0DbLXlvRAH+yHPRit2hHOjtUNZh4Axv+cpG09qbUI3+43eEy24B7G/Uh+GTfbjsXsOxQx/x  
p9otyVwc7hsQ5TA5PZb+mvkJ5OBEKzet9XcKwONBYELGhnEPe7cCgYEA06Vgov6YHleHui9kHuws  
ayav0elc5zkkjF9nfHFJRry21R1trw2Vdpn+9g481URrpzWVOEihvm+xtTmaZlSp/1lkq75XDwnU  
WA8gkn6O3QE3fq2yN98BURsAKdJfJ5RL1HvGQvTe10HLYYXpJnEkHv+Unl2ajLivWU+5pbBrKbUC  
gYBjbo+OZk0sCcpZ29sbzjYjpIddErySIyRX5gV2uNQwAjlDp9PfN295yQ+BxMBXiIycWVQiw0bH  
oMo7yykABY7Ozd5wQewBQ4AdS1WSX4nGDtsiFxiWi5sKuAAeOCbTosy1s8w8fxoJ5Tz1sdoxNeGs  
Arq6Wv/G16zQuAE9zK9vvwKBGF+09VI/1wJBirsDGz9whVWFPrTkJNvJZzYt69qezx1sjgFKshy  
WBhd4xHZtmCqpBP1Aymejr/TolbxyARMXmNIOWIANNXMGB4KGSylmzSVAoQ+fqR+cJ3d0dyP1lj  
jyb0Ed/NY8frlNDxAVHE8BSkdsx2f6LEyBKJSRr9snRAoGAMrTwXneXzvTskF/S5Fyu0iOegLda  
NWUH38v/nDCgEpIXD5Hn3qAEcjul1jmbwlvTwnY2jVhv7UGd8MjwUTNGItdb6nsYqM2asrnF3qS  
VRkAKKKYegJkpUfVTrW0YFjXkfcR/V+QFL5OndHAKJXjW7a4ejJLncTzmZSpYzwApc=  
-----END RSA PRIVATE KEY-----
```

Your private key isn't stored in AWS and can be retrieved **only** when it's created. You can't recover it later. Instead, if you lose the private key, you must create a new key pair.

If you're connecting to your instance from a Linux computer, we recommend that you use the following command to set the permissions of your private key file so that only you can read it.

```
$ chmod 400 MyKeyPair.pem
```

Display Your Key Pair

A "fingerprint" is generated from your key pair, and you can use it to verify that the private key that you have on your local machine matches the public key that's stored in AWS.

The fingerprint is an SHA1 hash taken from a DER-encoded copy of the private key. This value is captured when the key pair is created, and is stored in AWS with the public key. You can view the fingerprint in the Amazon EC2 console or by running the AWS CLI command `aws ec2 describe-key-pairs`.

The following example displays the fingerprint for `MyKeyPair`.

```
$ aws ec2 describe-key-pairs --key-name MyKeyPair
{
  "KeyPairs": [
    {
      "KeyName": "MyKeyPair",
      "KeyFingerprint": "1f:51:ae:28:bf:89:e9:d8:1f:25:5d:37:2d:7d:b8:ca:9f:f5:f1:6f"
    }
  ]
}
```

For more information about keys and fingerprints, see [Amazon EC2 Key Pairs](#) in the *Amazon EC2 User Guide for Linux Instances*.

Delete Your Key Pair

To delete a key pair, run the following command, substituting `MyKeyPair` with the name of the pair to delete.

```
$ aws ec2 delete-key-pair --key-name MyKeyPair
```

Creating, Configuring, and Deleting Security Groups for Amazon EC2

You can create a security group for your Amazon Elastic Compute Cloud (Amazon EC2) instances that essentially operates as a firewall, with rules that determine what network traffic can enter and leave.

You can create security groups to use in a virtual private cloud (VPC), or in the EC2-Classic shared flat network. For more information about the differences between EC2-Classic and EC2-VPC, see [Supported Platforms](#) in the *Amazon EC2 User Guide for Linux Instances*.

Use the AWS Command Line Interface (AWS CLI) to create a security group, add rules to existing security groups, and delete security groups.

Note

The following examples assume that you have already [configured your default credentials](#) (p. 101).

Topics

- [Create a Security Group](#) (p. 104)
- [Add Rules to Your Security Group](#) (p. 105)

- [Delete Your Security Group \(p. 107\)](#)

Create a Security Group

You can create security groups associated with VPCs or for EC2-Classical.

EC2-VPC

The following example shows how to create a security group for a specified VPC.

```
$ aws ec2 create-security-group --group-name my-sg --description "My security group" --vpc-id vpc-1a2b3c4d
{
  "GroupId": "sg-903004f8"
}
```

To view the initial information for a security group, run the [describe-security-groups](#) command. You can reference an EC2-VPC security group only by its `vpc-id`, not its name.

```
$ aws ec2 describe-security-groups --group-ids sg-903004f8
{
  "SecurityGroups": [
    {
      "IpPermissionsEgress": [
        {
          "IpProtocol": "-1",
          "IpRanges": [
            {
              "CidrIp": "0.0.0.0/0"
            }
          ],
          "UserIdGroupPairs": []
        }
      ],
      "Description": "My security group",
      "IpPermissions": [],
      "GroupName": "my-sg",
      "VpcId": "vpc-1a2b3c4d",
      "OwnerId": "123456789012",
      "GroupId": "sg-903004f8"
    }
  ]
}
```

EC2-Classical

The following example shows how to create a security group for EC2-Classical.

```
$ aws ec2 create-security-group --group-name my-sg --description "My security group"
{
  "GroupId": "sg-903004f8"
}
```

To view the initial information for `my-sg`, run the [describe-security-groups](#) command. For an EC2-Classical security group, you can reference it by its name.

```
$ aws ec2 describe-security-groups --group-names my-sg
```

```
{
  "SecurityGroups": [
    {
      "IpPermissionsEgress": [],
      "Description": "My security group",
      "IpPermissions": [],
      "GroupName": "my-sg",
      "OwnerId": "123456789012",
      "GroupId": "sg-903004f8"
    }
  ]
}
```

Add Rules to Your Security Group

When you run an Amazon EC2 instance, you must enable rules in the security group to allow incoming network traffic for your means of connecting to the image.

For example, if you're launching a Windows instance, you typically add a rule to allow inbound traffic on TCP port 3389 to support Remote Desktop Protocol (RDP). If you're launching a Linux instance, you typically add a rule to allow inbound traffic on TCP port 22 to support SSH connections.

Use the [authorize-security-group-ingress](#) command to add a rule to your security group. A required parameter of this command is the public IP address of your computer, or the network (in the form of an address range) that your computer is attached to, in [CIDR](#) notation.

Note

We provide the following service, <https://checkip.amazonaws.com/>, to enable you to determine your public IP address. To find other services that can help you identify your IP address, use your browser to search for "*what is my IP address*". If you connect through an ISP or from behind your firewall using a dynamic IP address (through a NAT gateway from a private network), your address can change periodically. In that case, you must find out the range of IP addresses used by client computers.

EC2-VPC

The following example shows how to add a rule for RDP (TCP port 3389) to an EC2-VPC security group with the ID `sg-903004f8`. This example assumes the client computer has an address somewhere in the CIDR range `203.0.113.0/24`.

You can start by confirming that your public address shows as included in the CIDR range `203.0.113.0/24`.

```
$ curl https://checkip.amazonaws.com
203.0.113.57
```

With that information confirmed, you can add the range to your security group by running the [authorize-security-group-ingress](#) command.

```
$ aws ec2 authorize-security-group-ingress --group-id sg-903004f8 --protocol tcp --port
3389 --cidr 203.0.113.0/24
```

The following command adds another rule to enable SSH to instances in the same security group.

```
$ aws ec2 authorize-security-group-ingress --group-id sg-903004f8 --protocol tcp --port 22
--cidr 203.0.113.0/24
```

To view the changes to the security group, run the [describe-security-groups](#) command.

```
$ aws ec2 describe-security-groups --group-ids sg-903004f8
{
  "SecurityGroups": [
    {
      "IpPermissionsEgress": [
        {
          "IpProtocol": "-1",
          "IpRanges": [
            {
              "CidrIp": "0.0.0.0/0"
            }
          ],
          "UserIdGroupPairs": []
        }
      ],
      "Description": "My security group"
      "IpPermissions": [
        {
          "ToPort": 22,
          "IpProtocol": "tcp",
          "IpRanges": [
            {
              "CidrIp": "203.0.113.0/24"
            }
          ],
          "UserIdGroupPairs": [],
          "FromPort": 22
        }
      ],
      "GroupName": "my-sg",
      "OwnerId": "123456789012",
      "GroupId": "sg-903004f8"
    }
  ]
}
```

EC2-Classic

The following command adds a rule for RDP to the EC2-Classic security group named *my-sg*.

```
$ aws ec2 authorize-security-group-ingress --group-name my-sg --protocol tcp --port 3389 --
cidr 203.0.113.0/24
```

The following command adds another rule for SSH to the same security group.

```
$ aws ec2 authorize-security-group-ingress --group-name my-sg --protocol tcp --port 22 --
cidr 203.0.113.0/24
```

To view the changes to your security group, run the [describe-security-groups](#) command.

```
$ aws ec2 describe-security-groups --group-names my-sg
{
  "SecurityGroups": [
    {
      "IpPermissionsEgress": [],
      "Description": "My security group"
      "IpPermissions": [
        {

```

```
        "ToPort": 22,
        "IpProtocol": "tcp",
        "IpRanges": [
            {
                "CidrIp": "203.0.113.0/24"
            }
        ]
        "UserIdGroupPairs": [],
        "FromPort": 22
    }
],
"GroupName": "my-sg",
"OwnerId": "123456789012",
"GroupId": "sg-903004f8"
}
]
```

Delete Your Security Group

To delete a security group, run the [delete-security-group](#) command.

Note

You can't delete a security group if it's currently attached to an environment.

EC2-VPC

The following command deletes an EC2-VPC security group.

```
$ aws ec2 delete-security-group --group-id sg-903004f8
```

EC2-Classic

The following command deletes the EC2-Classic security group named `my-sg`.

```
$ aws ec2 delete-security-group --group-name my-sg
```

Launching, Listing, and Terminating Amazon EC2 Instances

You can use the AWS Command Line Interface (AWS CLI) to launch, list, and terminate Amazon Elastic Compute Cloud (Amazon EC2) instances. You need a [key pair](#) (p. 101) and a [security group](#) (p. 103). You also need to select an Amazon Machine Image (AMI) and make a note of the AMI ID. For more information, see [Finding a Suitable AMI](#) in the *Amazon EC2 User Guide for Linux Instances*.

If you launch an instance that isn't within the AWS 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.

Note

The following examples assume that you have already [configured your default credentials](#) (p. 101).

Topics

- [Launch Your Instance](#) (p. 108)
- [Add a Block Device to Your Instance](#) (p. 111)

- [Add a Tag to Your Instance \(p. 111\)](#)
- [Connect to Your Instance \(p. 112\)](#)
- [List Your Instances \(p. 112\)](#)
- [Terminate Your Instance \(p. 112\)](#)

Launch Your Instance

To launch an Amazon EC2 instance using the AMI you selected, use the [run-instances](#) command. You can launch the instance into a virtual private cloud (VPC), or if your account supports it, into EC2-Classic.

Initially, your instance appears in the pending state, but changes to the running state after a few minutes.

EC2-VPC

The following example shows how to launch a `t2.micro` instance in the specified subnet of a VPC. Replace the *italicized* parameter values with your own.

```
$ aws ec2 run-instances --image-id ami-xxxxxxx --count 1 --instance-type t2.micro --key-name MyKeyPair --security-group-ids sg-903004f8 --subnet-id subnet-6e7f829e
{
  "OwnerId": "123456789012",
  "ReservationId": "r-5875ca20",
  "Groups": [
    {
      "GroupName": "my-sg",
      "GroupId": "sg-903004f8"
    }
  ],
  "Instances": [
    {
      "Monitoring": {
        "State": "disabled"
      },
      "PublicDnsName": null,
      "Platform": "windows",
      "State": {
        "Code": 0,
        "Name": "pending"
      },
      "EbsOptimized": false,
      "LaunchTime": "2013-07-19T02:42:39.000Z",
      "PrivateIpAddress": "10.0.1.114",
      "ProductCodes": [],
      "VpcId": "vpc-1a2b3c4d",
      "InstanceId": "i-5203422c",
      "ImageId": "ami-173d747e",
      "PrivateDnsName": ip-10-0-1-114.ec2.internal,
      "KeyName": "MyKeyPair",
      "SecurityGroups": [
        {
          "GroupName": "my-sg",
          "GroupId": "sg-903004f8"
        }
      ],
      "ClientToken": null,
      "SubnetId": "subnet-6e7f829e",
      "InstanceType": "t2.micro",
      "NetworkInterfaces": [
        {
```

```

        "Status": "in-use",
        "SourceDestCheck": true,
        "VpcId": "vpc-1a2b3c4d",
        "Description": "Primary network interface",
        "NetworkInterfaceId": "eni-a7edb1c9",
        "PrivateIpAddresses": [
            {
                "PrivateDnsName": "ip-10-0-1-114.ec2.internal",
                "Primary": true,
                "PrivateIpAddress": "10.0.1.114"
            }
        ],
        "PrivateDnsName": "ip-10-0-1-114.ec2.internal",
        "Attachment": {
            "Status": "attached",
            "DeviceIndex": 0,
            "DeleteOnTermination": true,
            "AttachmentId": "eni-attach-52193138",
            "AttachTime": "2013-07-19T02:42:39.000Z"
        },
        "Groups": [
            {
                "GroupName": "my-sg",
                "GroupId": "sg-903004f8"
            }
        ],
        "SubnetId": "subnet-6e7f829e",
        "OwnerId": "123456789012",
        "PrivateIpAddress": "10.0.1.114"
    }
],
"SourceDestCheck": true,
"Placement": {
    "Tenancy": "default",
    "GroupName": null,
    "AvailabilityZone": "us-west-2b"
},
"Hypervisor": "xen",
"BlockDeviceMappings": [
    {
        "DeviceName": "/dev/sda1",
        "Ebs": {
            "Status": "attached",
            "DeleteOnTermination": true,
            "VolumeId": "vol-877166c8",
            "AttachTime": "2013-07-19T02:42:39.000Z"
        }
    }
],
"Architecture": "x86_64",
"StateReason": {
    "Message": "pending",
    "Code": "pending"
},
"RootDeviceName": "/dev/sda1",
"VirtualizationType": "hvm",
"RootDeviceType": "ebs",
"Tags": [
    {
        "Value": "MyInstance",
        "Key": "Name"
    }
],
"AmiLaunchIndex": 0
}
]

```

```
}
```

EC2-Classic

If your account supports it, you can use the following command to launch a `t1.micro` instance in EC2-Classic. Replace the *italicized* parameter values with your own.

```
$ aws ec2 run-instances --image-id ami-173d747e --count 1 --instance-type t1.micro --key-  
name MyKeyPair --security-groups my-sg  
{  
  "OwnerId": "123456789012",  
  "ReservationId": "r-5875ca20",  
  "Groups": [  
    {  
      "GroupName": "my-sg",  
      "GroupId": "sg-903004f8"  
    }  
  ],  
  "Instances": [  
    {  
      "Monitoring": {  
        "State": "disabled"  
      },  
      "PublicDnsName": null,  
      "Platform": "windows",  
      "State": {  
        "Code": 0,  
        "Name": "pending"  
      },  
      "EbsOptimized": false,  
      "LaunchTime": "2013-07-19T02:42:39.000Z",  
      "ProductCodes": [],  
      "InstanceId": "i-5203422c",  
      "ImageId": "ami-173d747e",  
      "PrivateDnsName": null,  
      "KeyName": "MyKeyPair",  
      "SecurityGroups": [  
        {  
          "GroupName": "my-sg",  
          "GroupId": "sg-903004f8"  
        }  
      ],  
      "ClientToken": null,  
      "InstanceType": "t1.micro",  
      "NetworkInterfaces": [],  
      "Placement": {  
        "Tenancy": "default",  
        "GroupName": null,  
        "AvailabilityZone": "us-west-2b"  
      },  
      "Hypervisor": "xen",  
      "BlockDeviceMappings": [  
        {  
          "DeviceName": "/dev/sda1",  
          "Ebs": {  
            "Status": "attached",  
            "DeleteOnTermination": true,  
            "VolumeId": "vol-877166c8",  
            "AttachTime": "2013-07-19T02:42:39.000Z"  
          }  
        }  
      ],  
      "Architecture": "x86_64",  
      "StateReason": {
```

```

        "Message": "pending",
        "Code": "pending"
    },
    "RootDeviceName": "/dev/sda1",
    "VirtualizationType": "hvm",
    "RootDeviceType": "ebs",
    "Tags": [
        {
            "Value": "MyInstance",
            "Key": "Name"
        }
    ],
    "AmiLaunchIndex": 0
}
]
}

```

Add a Block Device to Your Instance

Each instance that you launch has an associated root device volume. You can use block device mapping to specify additional Amazon Elastic Block Store (Amazon EBS) volumes or instance store volumes to attach to an instance when it's launched.

To add a block device to your instance, specify the `--block-device-mappings` option when you use `run-instances`.

The following example parameter provisions a standard Amazon EBS volume that is 20 GB in size, and maps it to your instance using the identifier `/dev/sdf`.

```
--block-device-mappings "[{\"DeviceName\":\"/dev/sdf\",\"Ebs\":{\"VolumeSize\":20,
\DeleteOnTermination\":false}}]"
```

The following example adds an Amazon EBS volume, mapped to `/dev/sdf`, based on an existing snapshot. A snapshot represents an image that is loaded onto the volume for you. When you specify a snapshot, you don't have to specify a volume size; it will be large enough to hold your image. However, if you do specify a size, it must be greater than or equal to the size of the snapshot.

```
--block-device-mappings "[{\"DeviceName\":\"/dev/sdf\",\"Ebs\":{\"SnapshotId\":\"snap-
a1b2c3d4\"}}]"
```

The following example adds two volumes to your instance. The number of volumes available to your instance depends on its instance type.

```
--block-device-mappings "[{\"DeviceName\":\"/dev/sdf\",\"VirtualName\":\"ephemeral0\"},
{\"DeviceName\":\"/dev/sdg\",\"VirtualName\":\"ephemeral1\"}]"
```

The following example creates the mapping (`/dev/sdj`), but doesn't provision a volume for the instance.

```
--block-device-mappings "[{\"DeviceName\":\"/dev/sdj\",\"NoDevice\":\"\"}]"
```

For more information, see [Block Device Mapping](#) in the *Amazon EC2 User Guide for Linux Instances*.

Add a Tag to Your Instance

A tag is a label that you assign to an AWS resource. It enables you to add metadata to your resources that you can use for a variety of purposes. For more information, see [Tagging Your Resources](#) in the *Amazon EC2 User Guide for Linux Instances*.

The following example shows how to add a tag with the key name "Name" and the value "MyInstance" to the specified instance, by using the [create-tags](#) command.

```
$ aws ec2 create-tags --resources i-5203422c --tags Key=Name,Value=MyInstance
```

Connect to Your Instance

When your instance is running, you can connect to it and use it just as you'd use a computer sitting in front of you. For more information, see [Connect to Your Amazon EC2 Instance](#) in the *Amazon EC2 User Guide for Linux Instances*.

List Your Instances

You can use the AWS CLI to list your instances and view information about them. You can list all your instances, or filter the results based on the instances that you're interested in.

The following examples show how to use the [describe-instances](#) command.

The following command filters the list to only your `t2.micro` instances and outputs only the `InstanceId` values for each match.

```
$ aws ec2 describe-instances --filters "Name=instance-type,Values=t2.micro" --query
  "Reservations[].Instances[].InstanceId"
[
    "i-05e998023d9c69f9a"
]
```

The following command lists any of your instances that have the tag `Name=MyInstance`.

```
$ aws ec2 describe-instances --filters "Name=tag:Name,Values=MyInstance"
```

The following command lists your instances that were launched using any of the following AMIs: `ami-x0123456`, `ami-y0123456`, and `ami-z0123456`.

```
$ aws ec2 describe-instances --filters "Name=image-id,Values=ami-x0123456,ami-y0123456,ami-
z0123456"
```

Terminate Your Instance

Terminating an instance deletes it. You can't reconnect to an instance after you've terminated it.

As soon as the state of the instance changes to `shutting-down` or `terminated`, you stop incurring charges for that instance. If you want to reconnect to an instance later, use [stop-instances](#) instead of `terminate-instances`. For more information, see [Terminate Your Instance](#) in the *Amazon EC2 User Guide for Linux Instances*.

When you finish with an instance, you can use the command [terminate-instances](#) to delete it.

```
$ aws ec2 terminate-instances --instance-ids i-5203422c
{
  "TerminatingInstances": [
    {
      "InstanceId": "i-5203422c",
      "CurrentState": {
        "Code": 32,
        "Name": "shutting-down"
      },
    },
  ],
}
```

```
        "PreviousState": {  
            "Code": 16,  
            "Name": "running"  
        }  
    }  
]  
}
```

Using Amazon S3 Glacier with the AWS CLI

You can access the features of Amazon S3 Glacier using the AWS Command Line Interface (AWS CLI). To list the AWS CLI commands for S3 Glacier, use the following command.

```
aws glacier help
```

This topic shows examples of AWS CLI commands that perform common tasks for S3 Glacier. The examples demonstrate how to use the AWS CLI to upload a large file to Glacier by splitting it into smaller parts and uploading them from the command line.

Before you run any commands, set your default credentials. For more information, see [Configuring the AWS CLI \(p. 37\)](#).

Note

This tutorial uses several command line tools that typically come preinstalled on Unix-like operating systems, including Linux and macOS. Windows users can use the same tools by installing [Cygwin](#) and running the commands from the Cygwin terminal. We note Windows native commands and utilities that perform the same functions where available.

Topics

- [Create an Amazon S3 Glacier Vault \(p. 113\)](#)
- [Prepare a File for Uploading \(p. 113\)](#)
- [Initiate a Multipart Upload and Upload Files \(p. 114\)](#)
- [Complete the Upload \(p. 115\)](#)

Create an Amazon S3 Glacier Vault

Create a vault with the `create-vault` command.

```
$ aws glacier create-vault --account-id - --vault-name myvault  
{  
  "location": "/123456789012/vaults/myvault"  
}
```

Note

All S3 Glacier commands require an account ID parameter. Use the hyphen character (`--account-id -`) to use the current account.

Prepare a File for Uploading

Create a file for the test upload. The following commands create a file named `largefile` that contains exactly 3 MiB of random data.

Linux or macOS

```
$ dd if=/dev/urandom of=largefile bs=3145728 count=1
1+0 records in
1+0 records out
3145728 bytes (3.1 MB) copied, 0.205813 s, 15.3 MB/s
```

dd is a utility that copies a number of bytes from an input file to an output file. The previous example uses the system device file /dev/urandom as a source of random data. fsutil performs a similar function in Windows.

Windows

```
C:\> fsutil file createnew largefile 3145728
File C:\temp\largefile is created
```

Next, split the file into 1 MiB (1,048,576 byte) chunks.

```
$ split --bytes=1048576 --verbose largefile chunk
creating file `chunkaa'
creating file `chunkab'
creating file `chunkac'
```

Note

[HJ-Split](#) is a free file splitter for Windows and many other platforms.

Initiate a Multipart Upload and Upload Files

Create a multipart upload in Amazon S3 Glacier by using the `initiate-multipart-upload` command.

```
$ aws glacier initiate-multipart-upload --account-id - --archive-description "multipart
upload test" --part-size 1048576 --vault-name myvault
{
  "uploadId": "19gaRezEXAMPLES6Ry5YYdqthHOC_kGRCT03L9yetr220UmPtBYKk-
OssZtLqyFu7sY1_lR7vgFuJV6NtcV5zpsJ",
  "location": "/123456789012/vaults/myvault/multipart-
uploads/19gaRezEXAMPLES6Ry5YYdqthHOC_kGRCT03L9yetr220UmPtBYKk-
OssZtLqyFu7sY1_lR7vgFuJV6NtcV5zpsJ"
}
```

S3 Glacier requires the size of each part in bytes (1 MiB in this example), your vault name, and an account ID to configure the multipart upload. The AWS CLI outputs an upload ID when the operation is complete. Save the upload ID to a shell variable for later use.

Linux or macOS

```
$ UPLOADID="19gaRezEXAMPLES6Ry5YYdqthHOC_kGRCT03L9yetr220UmPtBYKk-
OssZtLqyFu7sY1_lR7vgFuJV6NtcV5zpsJ"
```

Windows

```
C:\> set UPLOADID="19gaRezEXAMPLES6Ry5YYdqthHOC_kGRCT03L9yetr220UmPtBYKk-
OssZtLqyFu7sY1_lR7vgFuJV6NtcV5zpsJ"
```

Next, use the `upload-multipart-part` command to upload each of the three parts.

```
$ aws glacier upload-multipart-part --upload-id $UPLOADID --body chunkaa --range 'bytes
0-1048575/*' --account-id - --vault-name myvault
```

```
{
  "checksum": "e1f2a7cd6e047fa606fe2f0280350f69b9f8cfa602097a9a026360a7edc1f553"
}
$ aws glacier upload-multipart-part --upload-id $UPLOADID --body chunkab --range 'bytes
1048576-2097151/*' --account-id - --vault-name myvault
{
  "checksum": "e1f2a7cd6e047fa606fe2f0280350f69b9f8cfa602097a9a026360a7edc1f553"
}
$ aws glacier upload-multipart-part --upload-id $UPLOADID --body chunkac --range 'bytes
2097152-3145727/*' --account-id - --vault-name myvault
{
  "checksum": "e1f2a7cd6e047fa606fe2f0280350f69b9f8cfa602097a9a026360a7edc1f553"
}
```

Note

The previous example uses the dollar sign (\$) to reference the contents of the UPLOADID shell variable on Linux. On the Windows command line, use a percent sign (%) on either side of the variable name (for example, %UPLOADID%).

You must specify the byte range of each part when you upload it so that Glacier can reassemble it in the correct order. Each piece is 1,048,576 bytes, so the first piece occupies bytes 0-1048575, the second 1048576-2097151, and the third 2097152-3145727.

Complete the Upload

Amazon S3 Glacier requires a tree hash of the original file to confirm that all of the uploaded pieces reached AWS intact.

To calculate a tree hash, you must split the file into 1 MiB parts and calculate a binary SHA-256 hash of each piece. Then you split the list of hashes into pairs, combine the two binary hashes in each pair, and take hashes of the results. Repeat this process until there is only one hash left. If there is an odd number of hashes at any level, promote it to the next level without modifying it.

The key to calculating a tree hash correctly when using command line utilities is to store each hash in binary format and convert to hexadecimal only at the last step. Combining or hashing the hexadecimal version of any hash in the tree will cause an incorrect result.

Note

Windows users can use the type command in place of cat. OpenSSL is available for Windows at [OpenSSL.org](https://www.openssl.org).

To calculate a tree hash

1. If you haven't already, split the original file into 1 MiB parts.

```
$ split --bytes=1048576 --verbose largefile chunk
creating file `chunkaa'
creating file `chunkab'
creating file `chunkac'
```

2. Calculate and store the binary SHA-256 hash of each chunk.

```
$ openssl dgst -sha256 -binary chunkaa > hash1
$ openssl dgst -sha256 -binary chunkab > hash2
$ openssl dgst -sha256 -binary chunkac > hash3
```

3. Combine the first two hashes and take the binary hash of the result.

```
$ cat hash1 hash2 > hash12
```



```
$ openssl dgst -sha256 -binary hash12 > hash12hash
```

4. Combine the parent hash of chunks aa and ab with the hash of chunk ac and hash the result, this time outputting hexadecimal. Store the result in a shell variable.

```
$ cat hash12hash hash3 > hash123
$ openssl dgst -sha256 hash123
SHA256(hash123)= 9628195fcdcbbbe76cdde932d4646fa7de5f219fb39823836d81f0cc0e18aa67
$ TREEHASH=9628195fcdcbbbe76cdde932d4646fa7de5f219fb39823836d81f0cc0e18aa67
```

Finally, complete the upload with the [complete-multipart-upload](#) command. This command takes the original file's size in bytes, the final tree hash value in hexadecimal, and your account ID and vault name.

```
$ aws glacier complete-multipart-upload --checksum $TREEHASH --archive-size 3145728 --
upload-id $UPLOADID --account-id - --vault-name myvault
{
  "archiveId": "d3AbWhE0YE1m6f_fI1jPG82F8xzbMEEZmrAllGAAONJAz05QdP-
N83MKqd96Unspoa5H51ItWX-sK8-QS0ZhwsyGiu9-R-kwWUyS1dSB1mgPPWkEbeFfqDSav053rU7FvVLHfRc6hg",
  "checksum": "9628195fcdcbbbe76cdde932d4646fa7de5f219fb39823836d81f0cc0e18aa67",
  "location": "/123456789012/vaults/myvault/archives/
d3AbWhE0YE1m6f_fI1jPG82F8xzbMEEZmrAllGAAONJAz05QdP-N83MKqd96Unspoa5H51ItWX-sK8-
QS0ZhwsyGiu9-R-kwWUyS1dSB1mgPPWkEbeFfqDSav053rU7FvVLHfRc6hg"
}
```

You can also check the status of the vault using the [describe-vault](#) command.

```
$ aws glacier describe-vault --account-id - --vault-name myvault
{
  "SizeInBytes": 3178496,
  "VaultARN": "arn:aws:glacier:us-west-2:123456789012:vaults/myvault",
  "LastInventoryDate": "2018-12-07T00:26:19.028Z",
  "NumberOfArchives": 1,
  "CreationDate": "2018-12-06T21:23:45.708Z",
  "VaultName": "myvault"
}
```

Note

Vault status is updated about once per day. See [Working with Vaults](#) for more information.

Now it's safe to remove the chunk and hash files that you created.

```
$ rm chunk* hash*
```

For more information on multipart uploads, see [Uploading Large Archives in Parts](#) and [Computing Checksums](#) in the *Amazon S3 Glacier Developer Guide*.

Using AWS Identity and Access Management from the AWS CLI

You can access the features of AWS Identity and Access Management (IAM) using the AWS Command Line Interface (AWS CLI). To list the AWS CLI commands for IAM, use the following command.

```
aws iam help
```

This topic shows examples of AWS CLI commands that perform common tasks for IAM.

Before you run any commands, set your default credentials. For more information, see [Configuring the AWS CLI \(p. 37\)](#).

Topics

- [Creating IAM Users and Groups \(p. 117\)](#)
- [Attaching an IAM Managed Policy to an IAM User \(p. 118\)](#)
- [Setting an Initial Password for an IAM User \(p. 119\)](#)
- [Create an Access Key for an IAM User \(p. 119\)](#)

Creating IAM Users and Groups

This topic describes how to use AWS Command Line Interface (AWS CLI) commands to create an AWS Identity and Access Management (IAM) group and a new IAM user, and then add the user to the group.

Before you run any commands, set your default credentials. For more information, see [Configuring the AWS CLI \(p. 37\)](#).

To create an IAM group and add a new IAM user to it

1. Use the `create-group` command to create the group.

```
$ aws iam create-group --group-name MyIamGroup
{
  "Group": {
    "GroupName": "MyIamGroup",
    "CreateDate": "2018-12-14T03:03:52.834Z",
    "GroupId": "AGPAJNUJ2W4IJVEXAMPLE",
    "Arn": "arn:aws:iam::123456789012:group/MyIamGroup",
    "Path": "/"
  }
}
```

2. Use the `create-user` command to create the user.

```
$ aws iam create-user --user-name MyUser
{
  "User": {
    "UserName": "MyUser",
    "Path": "/",
    "CreateDate": "2018-12-14T03:13:02.581Z",
    "UserId": "AIDAJY2PE5XUZ4EXAMPLE",
    "Arn": "arn:aws:iam::123456789012:user/MyUser"
  }
}
```

3. Use the `add-user-to-group` command to add the user to the group.

```
$ aws iam add-user-to-group --user-name MyUser --group-name MyIamGroup
```

4. To verify that the MyIamGroup group contains the MyUser, use the `get-group` command.

```
$ aws iam get-group --group-name MyIamGroup
{
  "Group": {
    "GroupName": "MyIamGroup",
```

```
    "CreateDate": "2018-12-14T03:03:52Z",
    "GroupId": "AGPAJNUJ2W4IJVEXAMPLE",
    "Arn": "arn:aws:iam::123456789012:group/MyIamGroup",
    "Path": "/"
  },
  "Users": [
    {
      "UserName": "MyUser",
      "Path": "/",
      "CreateDate": "2018-12-14T03:13:02Z",
      "UserId": "AIDAJY2PE5XUZ4EXAMPLE",
      "Arn": "arn:aws:iam::123456789012:user/MyUser"
    }
  ],
  "IsTruncated": "false"
}
```

Attaching an IAM Managed Policy to an IAM User

This topic describes how to use AWS Command Line Interface (AWS CLI) commands to attach an AWS Identity and Access Management (IAM) policy to an IAM user. The policy in this example provides the user with "Power User Access".

Before you run any commands, set your default credentials. For more information, see [Configuring the AWS CLI \(p. 37\)](#).

To attach an IAM managed policy to an IAM user

1. Determine the Amazon Resource Name (ARN) of the policy to attach. The following command uses `list-policies` to find the ARN of the policy with the name `PowerUserAccess`. It then stores that ARN in an environment variable.

```
$ export POLICYARN=$(aws iam list-policies --query 'Policies[?
PolicyName==`PowerUserAccess`].{ARN:Arn}' --output text)
$ echo $POLICYARN
arn:aws:iam::aws:policy/PowerUserAccess
```

2. To attach the policy, use the `attach-user-policy` command, and reference the environment variable that holds the policy ARN.

```
$ aws iam attach-user-policy --user-name MyUser --policy-arn $POLICYARN
```

3. Verify that the policy is attached to the user by running the `list-attached-user-policies` command.

```
$ aws iam list-attached-user-policies --user-name MyUser
{
  "AttachedPolicies": [
    {
      "PolicyName": "PowerUserAccess",
      "PolicyArn": "arn:aws:iam::aws:policy/PowerUserAccess"
    }
  ]
}
```

For more information, see [Access Management Resources](#). This topic provides links to an overview of permissions and policies, and links to examples of policies for accessing Amazon S3, Amazon EC2, and other services.

Setting an Initial Password for an IAM User

This topic describes how to use AWS Command Line Interface (AWS CLI) commands to set an initial password for an AWS Identity and Access Management (IAM) user.

Before you run any commands, set your default credentials. For more information, see [Configuring the AWS CLI \(p. 37\)](#).

The following command uses `create-login-profile` to set an initial password on the specified user. When the user signs in for the first time, the user is required to change the password to something that only the user knows.

```
$ aws iam create-login-profile --user-name MyUser --password My!User1Login8P@ssword --password-reset-required
{
  "LoginProfile": {
    "UserName": "MyUser",
    "CreateDate": "2018-12-14T17:27:18Z",
    "PasswordResetRequired": true
  }
}
```

You can use the `update-login-profile` command to *change* the password for an IAM user.

```
$ aws iam update-login-profile --user-name MyUser --password My!User1ADifferentP@ssword
```

Create an Access Key for an IAM User

This topic describes how to use AWS Command Line Interface (AWS CLI) commands to create a set of access keys for an AWS Identity and Access Management (IAM) user.

Before you run any commands, set your default credentials. For more information, see [Configuring the AWS CLI \(p. 37\)](#).

You can use the `create-access-key` command to create an access key for an IAM user. An access key is a set of security credentials that consists of an access key ID and a secret key.

An IAM user can create only two access keys at one time. If you try to create a third set, the command returns a `LimitExceeded` error.

```
$ aws iam create-access-key --user-name MyUser
{
  "AccessKey": {
    "UserName": "MyUser",
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "Status": "Active",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "CreateDate": "2018-12-14T17:34:16Z"
  }
}
```

Use the `delete-access-key` command to delete an access key for an IAM user. Specify which access key to delete by using the access key ID.

```
$ aws iam delete-access-key --user-name MyUser --access-key-id AKIAIOSFODNN7EXAMPLE
```

Using Amazon S3 with the AWS CLI

You can access the features of Amazon Simple Storage Service (Amazon S3) using the AWS Command Line Interface (AWS CLI).

Before you run any commands, set your default credentials. For more information, see [Configuring the AWS CLI \(p. 37\)](#).

The AWS CLI provides two tiers of commands for accessing Amazon S3:

- The `s3` tier consists of high-level commands that simplify performing common tasks, such as creating, manipulating, and deleting objects and buckets.
- The `s3api` tier behaves identically to other AWS services by exposing direct access to all Amazon S3 API operations. It enables you to carry out advanced operations that might not be possible with the following tier's high-level commands alone.

To get a list of all of the commands available in each tier, use the `help` argument with the `aws s3api` or `aws s3` commands.

```
$ aws s3 help
```

```
$ aws s3api help
```

Note

The AWS CLI supports copying, moving, and syncing from Amazon S3 to Amazon S3 using the *server-side COPY* operation provided by Amazon S3. This means that your files are kept in the cloud, and are *not* downloaded to the client machine, then back up to Amazon S3.

When operations such as these can be performed completely in the cloud, only the bandwidth necessary for the HTTP request and response is used.

Topics

- [Using High-Level \(s3\) Commands with the AWS CLI \(p. 120\)](#)
- [Using API-Level \(s3api\) Commands with the AWS CLI \(p. 124\)](#)

Using High-Level (s3) Commands with the AWS CLI

This topic describes how you can manage Amazon S3 buckets and objects using high-level `aws s3` commands.

Before you run any commands, set your default credentials. For more information, see [Configuring the AWS CLI \(p. 37\)](#).

Manage Buckets

High-level `aws s3` commands support common bucket operations, such as creating, listing, and deleting buckets.

Create a Bucket

Use the `s3 mb` command to create a bucket. Bucket names must be **globally** unique and should be DNS compliant. Bucket names can contain lowercase letters, numbers, hyphens, and periods. Bucket names can start and end only with a letter or number, and cannot contain a period next to a hyphen or another period.

```
$ aws s3 mb s3://bucket-name
```

List Your Buckets

Use the `s3 ls` command to list your buckets. Here are some examples of common usage.

The following command lists all buckets.

```
$ aws s3 ls
2018-12-11 17:08:50 my-bucket
2018-12-14 14:55:44 my-bucket2
```

The following command lists all objects and folders (referred to in S3 as 'prefixes') in a bucket.

```
$ aws s3 ls s3://bucket-name
                PRE path/
2018-12-04 19:05:48          3 MyFile1.txt
```

The previous output shows that under the prefix `path/` there exists one file named `MyFile1.txt`.

You can filter the output to a specific prefix by including it in the command. The following command lists the objects in `bucket-name/path` (that is, objects in `bucket-name` filtered by the prefix `path/`).

```
$ aws s3 ls s3://bucket-name/path/
2018-12-06 18:59:32          3 MyFile2.txt
```

Delete a Bucket

To remove a bucket, use the `s3 rb` command.

```
$ aws s3 rb s3://bucket-name
```

By default, the bucket must be empty for the operation to succeed. To remove a non-empty bucket, you need to include the `--force` option.

The following example deletes all objects and subfolders in the bucket and then removes the bucket.

```
$ aws s3 rb s3://bucket-name --force
```

Note

If you're using a versioned bucket that contains previously deleted—but retained—objects, this command does *not* allow you to remove the bucket. You must first remove all of the content.

Manage Objects

The high-level `aws s3` commands make it convenient to manage Amazon S3 objects. The object commands include `s3 cp`, `s3 ls`, `s3 mv`, `s3 rm`, and `s3 sync`.

The `cp`, `ls`, `mv`, and `rm` commands work similarly to their Unix counterparts and enable you to work seamlessly across your local directories and Amazon S3 buckets. The `sync` command synchronizes the contents of a bucket and a directory, or two buckets.

Note

All high-level commands that involve uploading objects into an Amazon S3 bucket (`s3 cp`, `s3 mv`, and `s3 sync`) automatically perform a multipart upload when the object is large.

Failed uploads can't be resumed when using these commands. If the multipart upload fails due to a timeout or is manually canceled by pressing **Ctrl+C**, the AWS CLI cleans up any files created and aborts the upload. This process can take several minutes.

If the process is interrupted by a kill command or system failure, the in-progress multipart upload remains in Amazon S3 and must be cleaned up manually in the AWS Management Console or with the `s3api abort-multipart-upload` command.

The `cp`, `mv`, and `sync` commands include a `--grants` option that you can use to grant permissions on the object to specified users or groups. Set the `--grants` option to a list of permissions using following syntax.

```
--grants Permission=Grantee_Type=Grantee_ID
        [Permission=Grantee_Type=Grantee_ID ...]
```

Each value contains the following elements:

- *Permission* – Specifies the granted permissions, and can be set to `read`, `readacl`, `writeacl`, or `full`.
- *Grantee_Type* – Specifies how to identify the grantee, and can be set to `uri`, `emailaddress`, or `id`.
- *Grantee_ID* – Specifies the grantee based on *Grantee_Type*.
 - `uri` – The group's URI. For more information, see [Who Is a Grantee?](#)
 - `emailaddress` – The account's email address.
 - `id` – The account's canonical ID.

For more information on Amazon S3 access control, see [Access Control](#).

The following example copies an object into a bucket. It grants `read` permissions on the object to everyone and `full` permissions (`read`, `readacl`, and `writeacl`) to the account associated with `user@example.com`.

```
$ aws s3 cp file.txt s3://my-bucket/ --grants read=uri=http://acs.amazonaws.com/groups/global/AllUsers full=emailaddress=user@example.com
```

You can also specify a nondefault storage class (`REDUCED_REDUNDANCY` or `STANDARD_IA`) for objects that you upload to Amazon S3. To do this, use the `--storage-class` option.

```
$ aws s3 cp file.txt s3://my-bucket/ --storage-class REDUCED_REDUNDANCY
```

The `s3 sync` command uses the following syntax. Possible source-target combinations are:

- Local file system to Amazon S3
- Amazon S3 to local file system
- Amazon S3 to Amazon S3

```
$ aws s3 sync <source> <target> [--options]
```

The following example synchronizes the contents of an Amazon S3 folder named `path` in `my-bucket` with the current working directory. `s3 sync` updates any files that have a different size or modified time than files with the same name at the destination. The output displays specific operations performed during the sync. Notice that the operation recursively synchronizes the subdirectory `MySubdirectory` and its contents with `s3://my-bucket/path/MySubdirectory`.

```
$ aws s3 sync . s3://my-bucket/path
```

```
upload: MySubdirectory\MyFile3.txt to s3://my-bucket/path/MySubdirectory/MyFile3.txt
upload: MyFile2.txt to s3://my-bucket/path/MyFile2.txt
upload: MyFile1.txt to s3://my-bucket/path/MyFile1.txt
```

Typically, `s3 sync` only copies missing or outdated files or objects between the source and target. However, you can also supply the `--delete` option to remove files or objects from the target that are not present in the source.

The following example, which extends the previous one, shows how this works.

```
// Delete local file
$ rm ./MyFile1.txt

// Attempt sync without --delete option - nothing happens
$ aws s3 sync . s3://my-bucket/path

// Sync with deletion - object is deleted from bucket
$ aws s3 sync . s3://my-bucket/path --delete
delete: s3://my-bucket/path/MyFile1.txt

// Delete object from bucket
$ aws s3 rm s3://my-bucket/path/MySubdirectory/MyFile3.txt
delete: s3://my-bucket/path/MySubdirectory/MyFile3.txt

// Sync with deletion - local file is deleted
$ aws s3 sync s3://my-bucket/path . --delete
delete: MySubdirectory\MyFile3.txt

// Sync with Infrequent Access storage class
$ aws s3 sync . s3://my-bucket/path --storage-class STANDARD_IA
```

You can use the `--exclude` and `--include` options to specify rules that filter the files or objects to copy during the sync operation. By default, all items in a specified folder are included in the sync. Therefore, `--include` is needed only when you have to specify exceptions to the `--exclude` option (that is, `--include` effectively means "don't exclude"). The options apply in the order that's specified, as shown in the following example.

```
Local directory contains 3 files:
MyFile1.txt
MyFile2.rtf
MyFile88.txt
'''

$ aws s3 sync . s3://my-bucket/path --exclude "*.txt"
upload: MyFile2.rtf to s3://my-bucket/path/MyFile2.rtf
'''

$ aws s3 sync . s3://my-bucket/path --exclude "*.txt" --include "MyFile*.txt"
upload: MyFile1.txt to s3://my-bucket/path/MyFile1.txt
upload: MyFile88.txt to s3://my-bucket/path/MyFile88.txt
upload: MyFile2.rtf to s3://my-bucket/path/MyFile2.rtf
'''

$ aws s3 sync . s3://my-bucket/path --exclude "*.txt" --include "MyFile*.txt" --exclude
  "MyFile?.txt"
upload: MyFile2.rtf to s3://my-bucket/path/MyFile2.rtf
upload: MyFile88.txt to s3://my-bucket/path/MyFile88.txt
```

The `--exclude` and `--include` options also filter files or objects to be deleted during an `s3 sync` operation that includes the `--delete` option. In this case, the parameter string must specify files to exclude from, or include for, deletion in the context of the target directory or bucket. The following shows an example.

```
Assume local directory and s3://my-bucket/path currently in sync and each contains 3 files:
```



```
MyFile1.txt
MyFile2.rtf
MyFile88.txt
'''
// Delete local .txt files
$ rm *.txt

// Sync with delete, excluding files that match a pattern. MyFile88.txt is deleted, while
remote MyFile1.txt is not.
$ aws s3 sync . s3://my-bucket/path --delete --exclude "my-bucket/path/MyFile?.txt"
delete: s3://my-bucket/path/MyFile88.txt
'''
// Delete MyFile2.rtf
$ aws s3 rm s3://my-bucket/path/MyFile2.rtf

// Sync with delete, excluding MyFile2.rtf - local file is NOT deleted
$ aws s3 sync s3://my-bucket/path . --delete --exclude "./MyFile2.rtf"
download: s3://my-bucket/path/MyFile1.txt to MyFile1.txt
'''
// Sync with delete, local copy of MyFile2.rtf is deleted
$ aws s3 sync s3://my-bucket/path . --delete
delete: MyFile2.rtf
```

The `s3 sync` command also accepts an `--acl` option, by which you may set the access permissions for files copied to Amazon S3. The `--acl` option accepts `private`, `public-read`, and `public-read-write` values.

```
$ aws s3 sync . s3://my-bucket/path --acl public-read
```

As previously mentioned, the `s3` command set includes `cp`, `mv`, `ls`, and `rm`, and they work in similar ways to their Unix counterparts. The following are some examples.

```
// Copy MyFile.txt in current directory to s3://my-bucket/path
$ aws s3 cp MyFile.txt s3://my-bucket/path/

// Move all .jpg files in s3://my-bucket/path to ./MyDirectory
$ aws s3 mv s3://my-bucket/path ./MyDirectory --exclude "*" --include "*.jpg" --recursive

// List the contents of my-bucket
$ aws s3 ls s3://my-bucket

// List the contents of path in my-bucket
$ aws s3 ls s3://my-bucket/path/

// Delete s3://my-bucket/path/MyFile.txt
$ aws s3 rm s3://my-bucket/path/MyFile.txt

// Delete s3://my-bucket/path and all of its contents
$ aws s3 rm s3://my-bucket/path --recursive
```

When you use the `--recursive` option on a directory or folder with `cp`, `mv`, or `rm`, the command walks the directory tree, including all subdirectories. These commands also accept the `--exclude`, `--include`, and `--acl` options as the `sync` command does.

Using API-Level (s3api) Commands with the AWS CLI

The API-level commands (contained in the `s3api` command set) provide direct access to the Amazon Simple Storage Service (Amazon S3) APIs, and enable some operations that are not exposed in the high-

level `s3` commands. These commands are the equivalent of the other AWS services that provide API-level access to the services' functionality.

This topic provides examples that demonstrate how to use the lower-level commands that map to the Amazon S3 APIs. In addition, you can find examples for each S3 API in the [s3api section of the CLI Reference Guide](#).

Before you run any commands, set your default credentials. For more information, see [Configuring the AWS CLI \(p. 37\)](#).

Apply a Custom ACL

With high-level commands, you can use the `--acl` option to apply predefined access control lists (ACLs) to Amazon S3 objects. But you can't use that command to set bucket-wide ACLs. However, you can do this with the API-level command, [put-bucket-acl](#).

The following example shows how to grant full control to two AWS users (*user1@example.com* and *user2@example.com*) and read permission to everyone. The identifier for "everyone" comes from a special URI that you pass as a parameter.

```
$ aws s3api put-bucket-acl --bucket MyBucket --grant-full-control  
'emailaddress="user1@example.com",emailaddress="user2@example.com"' --grant-read  
'uri="http://acs.amazonaws.com/groups/global/AllUsers"'
```

For details about how to construct the ACLs, see [PUT Bucket acl](#) in the *Amazon Simple Storage Service API Reference*. The `s3api` ACL commands in the CLI, such as `put-bucket-acl`, use the same [shorthand argument notation](#).

Configure a Logging Policy

The API command `put-bucket-logging` configures bucket logging policy.

In the following example, the AWS user *user@example.com* is granted full control over the log files, and all users have read access to them. Notice that the `put-bucket-acl` command is also required to grant the Amazon S3 log delivery system (specified by a URI) the permissions needed to read and write the logs to the bucket.

```
$ aws s3api put-bucket-acl --bucket MyBucket --grant-read-acp 'URI="http://  
acs.amazonaws.com/groups/s3/LogDelivery"' --grant-write 'URI="http://acs.amazonaws.com/  
groups/s3/LogDelivery"'  
$ aws s3api put-bucket-logging --bucket MyBucket --bucket-logging-status file://  
logging.json
```

The file `logging.json` in the previous command has the following content.

```
{  
  "LoggingEnabled": {  
    "TargetBucket": "MyBucket",  
    "TargetPrefix": "MyBucketLogs/",  
    "TargetGrants": [  
      {  
        "Grantee": {  
          "Type": "AmazonCustomerByEmail",  
          "EmailAddress": "user@example.com"  
        },  
        "Permission": "FULL_CONTROL"  
      },  
    ],  
  },  
}
```

```
    "Grantee": {
      "Type": "Group",
      "URI": "http://acs.amazonaws.com/groups/global/AllUsers"
    },
    "Permission": "READ"
  }
]
}
```

Using Amazon SNS with the AWS CLI

You can access the features of Amazon Simple Notification Service (Amazon SNS) using the AWS Command Line Interface (AWS CLI). To list the AWS CLI commands for Amazon SNS, use the following command.

```
aws sns help
```

Before you run any commands, set your default credentials. For more information, see [Configuring the AWS CLI \(p. 37\)](#).

This topic shows examples of CLI commands that perform common tasks for Amazon SNS.

Topics

- [Create a Topic \(p. 126\)](#)
- [Subscribe to a Topic \(p. 126\)](#)
- [Publish to a Topic \(p. 127\)](#)
- [Unsubscribe from a Topic \(p. 127\)](#)
- [Delete a Topic \(p. 127\)](#)

Create a Topic

To create a topic, use the `create-topic` command and specify the name to assign to the topic.

```
$ aws sns create-topic --name my-topic
{
  "TopicArn": "arn:aws:sns:us-west-2:123456789012:my-topic"
}
```

Make a note of the response's `TopicArn`, which you use later to publish a message.

Subscribe to a Topic

To subscribe to a topic, use the `subscribe` command.

The following example specifies the `email` protocol and an email address for the notification-endpoint.

```
$ aws sns subscribe --topic-arn arn:aws:sns:us-west-2:123456789012:my-topic --
protocol email --notification-endpoint saanvi@example.com
{
  "SubscriptionArn": "pending confirmation"
}
```

```
}
```

AWS immediately sends a confirmation message by email to the address you specified in the `subscribe` command. The email message has the following text.

```
You have chosen to subscribe to the topic:
arn:aws:sns:us-west-2:123456789012:my-topic
To confirm this subscription, click or visit the following link (If this was in error no
  action is necessary):
Confirm subscription
```

After the recipient clicks the **Confirm subscription** link, the recipient's browser displays a notification message with information similar to the following.

```
Subscription confirmed!

You have subscribed saanvi@example.com to the topic:my-topic.

Your subscription's id is:
arn:aws:sns:us-west-2:123456789012:my-topic:1328f057-de93-4c15-512e-8bb22EXAMPLE

If it was not your intention to subscribe, click here to unsubscribe.
```

Publish to a Topic

To send a message to all subscribers of a topic, use the `publish` command.

The following example sends the message "Hello World!" to all subscribers of the specified topic.

```
$ aws sns publish --topic-arn arn:aws:sns:us-west-2:123456789012:my-topic --message "Hello
World!"
{
  "MessageId": "4e41661d-5eec-5ddf-8dab-2c867EXAMPLE"
}
```

In this example, AWS sends an email message with the text "Hello World!" to `saanvi@example.com`.

Unsubscribe from a Topic

To unsubscribe from a topic and stop receiving messages published to that topic, use the `unsubscribe` command and specify the ARN of the topic you want to unsubscribe from.

```
$ aws sns unsubscribe --subscription-arn arn:aws:sns:us-west-2:123456789012:my-
topic:1328f057-de93-4c15-512e-8bb22EXAMPLE
```

To verify that you successfully unsubscribed, use the `list-subscriptions` command to confirm that the ARN no longer appears in the list.

```
$ aws sns list-subscriptions
```

Delete a Topic

To delete a topic, run the `delete-topic` command.

```
$ aws sns delete-topic --topic-arn arn:aws:sns:us-west-2:123456789012:my-topic
```

To verify that AWS successfully deleted the topic, use the [list-topics](#) command to confirm that the topic no longer appears in the list.

```
$ aws sns list-topics
```

Using Amazon SWF with the AWS CLI

You can access the features of Amazon Simple Workflow Service (Amazon SWF) using the AWS Command Line Interface (AWS CLI).

To list the AWS CLI commands for Amazon SWF, use the following command.

```
aws swf help
```

Before you run any commands, set your default credentials. For more information, see [Configuring the AWS CLI](#) (p. 37).

The following topics show examples of CLI commands that perform common tasks for Amazon SWF.

Topics

- [List of Amazon SWF Commands by Category](#) (p. 128)
- [Working with Amazon SWF Domains Using the AWS CLI](#) (p. 131)

List of Amazon SWF Commands by Category

You can use the AWS Command Line Interface (AWS CLI) to create, display, and manage workflows in Amazon Simple Workflow Service (Amazon SWF).

This section lists the reference topics for Amazon SWF commands in the AWS CLI, grouped by *functional category*.

For an *alphabetic* list of commands, see the [Amazon SWF section](#) of the *AWS CLI Command Reference*, or use the following command.

```
$ aws swf help
```

You can also get help for an individual command, by placing the `help` directive after the command name. The following shows an example.

```
$ aws swf register-domain help
```

Topics

- [Commands Related to Activities](#) (p. 129)
- [Commands Related to Deciders](#) (p. 129)
- [Commands Related to Workflow Executions](#) (p. 129)
- [Commands Related to Administration](#) (p. 129)

- [Visibility Commands \(p. 130\)](#)

Commands Related to Activities

Activity workers use `poll-for-activity-task` to get new activity tasks. After a worker receives an activity task from Amazon SWF, it performs the task and responds using `respond-activity-task-completed` if successful or `respond-activity-task-failed` if unsuccessful.

The following are commands that are performed by activity workers:

- [poll-for-activity-task](#)
- [respond-activity-task-completed](#)
- [respond-activity-task-failed](#)
- [respond-activity-task-canceled](#)
- [record-activity-task-heartbeat](#)

Commands Related to Deciders

Deciders use `poll-for-decision-task` to get decision tasks. After a decider receives a decision task from Amazon SWF, it examines its workflow execution history and decides what to do next. It calls `respond-decision-task-completed` to complete the decision task and provides zero or more next decisions.

The following are commands that are performed by deciders:

- [poll-for-decision-task](#)
- [respond-decision-task-completed](#)

Commands Related to Workflow Executions

The following commands operate on a workflow execution:

- [request-cancel-workflow-execution](#)
- [start-workflow-execution](#)
- [signal-workflow-execution](#)
- [terminate-workflow-execution](#)

Commands Related to Administration

Although you can perform administrative tasks from the Amazon SWF console, you can use the commands in this section to automate functions or build your own administrative tools.

Activity Management

- [register-activity-type](#)
- [deprecate-activity-type](#)

Workflow Management

- [register-workflow-type](#)
- [deprecate-workflow-type](#)

Domain Management

- [register-domain](#)
- [deprecate-domain](#)

For more information and examples of these domain management commands, see [Working with Amazon SWF Domains Using the AWS CLI](#) (p. 131).

Workflow Execution Management

- [request-cancel-workflow-execution](#)
- [terminate-workflow-execution](#)

Visibility Commands

Although you can perform visibility actions from the Amazon SWF console, you can use the commands in this section to build your own console or administrative tools.

Activity Visibility

- [list-activity-types](#)
- [describe-activity-type](#)

Workflow Visibility

- [list-workflow-types](#)
- [describe-workflow-type](#)

Workflow Execution Visibility

- [describe-workflow-execution](#)
- [list-open-workflow-executions](#)
- [list-closed-workflow-executions](#)
- [count-open-workflow-executions](#)
- [count-closed-workflow-executions](#)
- [get-workflow-execution-history](#)

Domain Visibility

- [list-domains](#)
- [describe-domain](#)

For more information and examples of these domain visibility commands, see [Working with Amazon SWF Domains Using the AWS CLI](#) (p. 131).

Task List Visibility

- [count-pending-activity-tasks](#)
- [count-pending-decision-tasks](#)

Working with Amazon SWF Domains Using the AWS CLI

You can use the AWS Command Line Interface (AWS CLI) to manage your Amazon Simple Workflow Service (Amazon SWF) domains.

Topics

- [List Your Domains \(p. 131\)](#)
- [Get Information about a Domain \(p. 131\)](#)
- [Register a Domain \(p. 132\)](#)
- [Deprecate a Domain \(p. 132\)](#)

List Your Domains

To list the Amazon SWF domains that you have registered for your AWS account, you can use `swf list-domains`. You must include `--registration-status` and specify either `REGISTERED` or `DEPRECATED`.

Here's a minimal example.

```
$ aws swf list-domains --registration-status REGISTERED
{
  "domainInfos": [
    {
      "status": "REGISTERED",
      "name": "ExampleDomain"
    },
    {
      "status": "REGISTERED",
      "name": "mytest"
    }
  ]
}
```

Note

For an example of using `DEPRECATED`, see [Deprecate a Domain \(p. 132\)](#).

For more information, see `list-domains` in the *AWS CLI Command Reference*.

Get Information about a Domain

To get detailed information about a particular domain, use `swf describe-domain`. There is one required parameter, `--name`, which takes the name of the domain you want information about, as shown in the following example.

```
$ aws swf describe-domain --name ExampleDomain
{
  "domainInfo": {
    "status": "REGISTERED",
    "name": "ExampleDomain"
  },
  "configuration": {
    "workflowExecutionRetentionPeriodInDays": "1"
  }
}
```


For more information, see [describe-domain](#) in the *AWS CLI Command Reference*.

Register a Domain

To register new domains, use `swf register-domain`.

There are two required parameters: `--name` and `--workflow-execution-retention-period-in-days`. The `--name` parameter takes the domain name to register. The `--workflow-execution-retention-period-in-days` parameter takes an integer to specify the number of days to retain workflow execution data on this domain, up to a maximum period of 90 days (for more information, see the [Amazon SWF FAQ](#)).

If you specify zero (0) for this value, the retention period is automatically set at the maximum duration. Otherwise, workflow execution data isn't retained after the specified number of days have passed. The following example shows how to register a new domain.

```
$ aws swf register-domain --name MyNeatNewDomain --workflow-execution-retention-period-in-days 0
```

The command doesn't return any output, but you can use `swf list-domains` or `swf describe-domain` to see the new domain, as shown in the following example.

```
$ aws swf describe-domain --name MyNeatNewDomain
{
  "domainInfo": {
    "status": "REGISTERED",
    "name": "MyNeatNewDomain"
  },
  "configuration": {
    "workflowExecutionRetentionPeriodInDays": "0"
  }
}
```

For more information, see [register-domain](#) in the *AWS CLI Command Reference*.

Deprecate a Domain

To deprecate a domain (you can still see it, but cannot create new workflow executions or register types on it), use `swf deprecate-domain`. It has a sole required parameter, `--name`, which takes the name of the domain to deprecate.

```
$ aws swf deprecate-domain --name MyNeatNewDomain
```

As with `register-domain`, no output is returned. If you use `list-domains` to view the registered domains, however, you will see that the domain no longer appears among them. You can also use `--registration-status DEPRECATED`.

```
$ aws swf list-domains --registration-status DEPRECATED
{
  "domainInfos": [
    {
      "status": "DEPRECATED",
      "name": "MyNeatNewDomain"
    }
  ]
}
```

For more information, see [deprecate-domain](#) in the *AWS CLI Command Reference*.

Troubleshooting AWS CLI Errors

General: Ensure you're running a recent version of the AWS CLI.

If you receive an error that indicates that a command doesn't exist, or that it doesn't recognize a parameter that the documentation says is available, we recommend that the first thing you do (after checking your command for spelling errors!) is to upgrade to the most recent version of the AWS CLI. Updated versions of the AWS CLI are released almost every business day. New AWS services, features, and parameters are introduced in those new versions of the AWS CLI. The only way to get access to those new services, features, or parameters is to upgrade to a version that was released after that element was first introduced.

How you update your version of the AWS CLI depends on how you originally installed it. For example, if you installed the AWS CLI using `pip`, run `pip install --upgrade`, as described in [Upgrading to the Latest Version of the AWS CLI version 1 \(p. 20\)](#).

If you used one of the bundled installers, you should remove the existing installation and download and install the latest version of the bundled installer for your operating system.

General: Use the `--debug` option.

One of the first things you should do when the AWS CLI reports an error that you don't immediately understand, or produces results that you don't expect, is get more detail about the error. You can do this by running the command again and including the `--debug` option at the end of the command line. This causes the AWS CLI to report details about every step it takes to process your command, send the request to the AWS servers, receive the response, and process the response into the output you see. The details in the output can help you to determine in which step the error occurs and to get context that can provide clues about what triggered it.

You can send the output to a text file to capture it for later review or to send it to AWS support when asked for it.

Here's an example of a command run with and without the `--debug` option.

```
$ aws iam list-groups --profile MyTestProfile
{
  "Groups": [
    {
      "Path": "/",
      "GroupName": "MyTestGroup",
      "GroupId": "AGPA0123456789EXAMPLE",
      "Arn": "arn:aws:iam::123456789012:group/MyTestGroup",
      "CreateDate": "2019-08-12T19:34:04Z"
    }
  ]
}
```

When you include the `--debug` option, details include (among other things):

- Looking for credentials
- Parsing the provided parameters
- Constructing the request sent to AWS servers
- The contents of the request sent to AWS
- The contents of the raw response
- The formatted output

```
$ aws iam list-groups --profile MyTestProfile --debug
2019-08-12 12:36:18,305 - MainThread - awscli.clidriver - DEBUG - CLI version: aws-
cli/1.16.215 Python/3.7.3 Linux/4.14.133-113.105.amzn2.x86_64 botocore/1.12.205
2019-08-12 12:36:18,305 - MainThread - awscli.clidriver - DEBUG - Arguments entered to CLI:
['iam', 'list-groups', '--debug']
2019-08-12 12:36:18,305 - MainThread - botocore.hooks - DEBUG - Event session-initialized:
calling handler <function add_scalar_parsers at 0x7fdf173161e0>
2019-08-12 12:36:18,305 - MainThread - botocore.hooks - DEBUG - Event session-initialized:
calling handler <function register_uri_param_handler at 0x7fdf17dec400>
2019-08-12 12:36:18,305 - MainThread - botocore.hooks - DEBUG - Event session-initialized:
calling handler <function inject_assume_role_provider_cache at 0x7fdf17da9378>
2019-08-12 12:36:18,307 - MainThread - botocore.credentials - DEBUG - Skipping environment
variable credential check because profile name was explicitly set.
2019-08-12 12:36:18,307 - MainThread - botocore.hooks - DEBUG - Event session-initialized:
calling handler <function attach_history_handler at 0x7fdf173ed9d8>
2019-08-12 12:36:18,308 - MainThread - botocore.loaders - DEBUG - Loading JSON file: /home/
ec2-user/venv/lib/python3.7/site-packages/botocore/data/iam/2010-05-08/service-2.json
2019-08-12 12:36:18,317 - MainThread - botocore.hooks - DEBUG - Event building-command-
table.iam: calling handler <function add_waiters at 0x7fdf1731a840>
2019-08-12 12:36:18,320 - MainThread - botocore.loaders - DEBUG - Loading JSON file: /home/
ec2-user/venv/lib/python3.7/site-packages/botocore/data/iam/2010-05-08/waiters-2.json
2019-08-12 12:36:18,321 - MainThread - awscli.clidriver - DEBUG - OrderedDict([('path-
prefix', <awscli.arguments.CLIArument object at 0x7fdf171ac780>), ('marker',
<awscli.arguments.CLIArument object at 0x7fdf171b09e8>), ('max-items',
<awscli.arguments.CLIArument object at 0x7fdf171b09b0>)])
2019-08-12 12:36:18,322 - MainThread - botocore.hooks - DEBUG - Event building-
argument-table.iam.list-groups: calling handler <function add_streaming_output_arg at
0x7fdf17316510>
2019-08-12 12:36:18,322 - MainThread - botocore.hooks - DEBUG - Event building-argument-
table.iam.list-groups: calling handler <function add_cli_input_json at 0x7fdf17da9d90>
2019-08-12 12:36:18,322 - MainThread - botocore.hooks - DEBUG - Event building-argument-
table.iam.list-groups: calling handler <function unify_paging_params at 0x7fdf17328048>
2019-08-12 12:36:18,326 - MainThread - botocore.loaders - DEBUG - Loading JSON file: /home/
ec2-user/venv/lib/python3.7/site-packages/botocore/data/iam/2010-05-08/paginators-1.json
2019-08-12 12:36:18,326 - MainThread - awscli.customizations.paginate - DEBUG - Modifying
paging parameters for operation: ListGroups
2019-08-12 12:36:18,326 - MainThread - botocore.hooks - DEBUG - Event building-argument-
table.iam.list-groups: calling handler <function add_generate_skeleton at 0x7fdf1737eae8>
2019-08-12 12:36:18,326 - MainThread - botocore.hooks - DEBUG - Event
before-building-argument-table-parser.iam.list-groups: calling handler
<bound method OverrideRequiredArgsArgument.override_required_args of
<awscli.customizations.cliinputjson.CliInputJSONArgument object at 0x7fdf171b0a58>>
2019-08-12 12:36:18,327 - MainThread - botocore.hooks - DEBUG - Event
before-building-argument-table-parser.iam.list-groups: calling handler
<bound method GenerateCliSkeletonArgument.override_required_args of
<awscli.customizations.generatecliskeleton.GenerateCliSkeletonArgument object at
0x7fdf171c5978>>
2019-08-12 12:36:18,327 - MainThread - botocore.hooks - DEBUG - Event operation-
args-parsed.iam.list-groups: calling handler functools.partial(<function
check_should_enable_pagination at 0x7fdf17328158>, ['marker', 'max-items'], {'max-
items': <awscli.arguments.CLIArument object at 0x7fdf171b09b0>}, OrderedDict([('path-
prefix', <awscli.arguments.CLIArument object at 0x7fdf171ac780>), ('marker',
<awscli.arguments.CLIArument object at 0x7fdf171b09e8>), ('max-items',
<awscli.customizations.paginate.PageArgument object at 0x7fdf171c58d0>), ('cli-
input-json', <awscli.customizations.cliinputjson.CliInputJSONArgument object at
```

```
0x7fdf171b0a58>), ('starting-token', <awscli.customizations.paginate.PageArgument
object at 0x7fdf171b0a20>), ('page-size', <awscli.customizations.paginate.PageArgument
object at 0x7fdf171c5828>), ('generate-cli-skeleton',
<awscli.customizations.generatecliskeleton.GenerateCliSkeletonArgument object at
0x7fdf171c5978>)]))
2019-08-12 12:36:18,328 - MainThread - botocore.hooks - DEBUG - Event load-cli-
arg.iam.list-groups.path-prefix: calling handler <awscli.paramfile.URIArgumentHandler
object at 0x7fdf1725c978>
2019-08-12 12:36:18,328 - MainThread - botocore.hooks - DEBUG - Event load-cli-
arg.iam.list-groups.marker: calling handler <awscli.paramfile.URIArgumentHandler object at
0x7fdf1725c978>
2019-08-12 12:36:18,328 - MainThread - botocore.hooks - DEBUG - Event load-cli-
arg.iam.list-groups.max-items: calling handler <awscli.paramfile.URIArgumentHandler object
at 0x7fdf1725c978>
2019-08-12 12:36:18,328 - MainThread - botocore.hooks - DEBUG - Event load-cli-
arg.iam.list-groups.cli-input-json: calling handler <awscli.paramfile.URIArgumentHandler
object at 0x7fdf1725c978>
2019-08-12 12:36:18,328 - MainThread - botocore.hooks - DEBUG - Event load-cli-
arg.iam.list-groups.starting-token: calling handler <awscli.paramfile.URIArgumentHandler
object at 0x7fdf1725c978>
2019-08-12 12:36:18,328 - MainThread - botocore.hooks - DEBUG - Event load-cli-
arg.iam.list-groups.page-size: calling handler <awscli.paramfile.URIArgumentHandler object
at 0x7fdf1725c978>
2019-08-12 12:36:18,328 - MainThread - botocore.hooks - DEBUG - Event
load-cli-arg.iam.list-groups.generate-cli-skeleton: calling handler
<awscli.paramfile.URIArgumentHandler object at 0x7fdf1725c978>
2019-08-12 12:36:18,329 - MainThread - botocore.hooks - DEBUG
- Event calling-command.iam.list-groups: calling handler
<bound method CliInputJSONArgument.add_to_call_parameters of
<awscli.customizations.cliinputjson.CliInputJSONArgument object at 0x7fdf171b0a58>>
2019-08-12 12:36:18,329 - MainThread - botocore.hooks - DEBUG -
Event calling-command.iam.list-groups: calling handler <bound
method GenerateCliSkeletonArgument.generate_json_skeleton of
<awscli.customizations.generatecliskeleton.GenerateCliSkeletonArgument object at
0x7fdf171c5978>>
2019-08-12 12:36:18,329 - MainThread - botocore.credentials - DEBUG - Looking for
credentials via: assume-role
2019-08-12 12:36:18,329 - MainThread - botocore.credentials - DEBUG - Looking for
credentials via: assume-role-with-web-identity
2019-08-12 12:36:18,329 - MainThread - botocore.credentials - DEBUG - Looking for
credentials via: shared-credentials-file
2019-08-12 12:36:18,329 - MainThread - botocore.credentials - INFO - Found credentials in
shared credentials file: ~/.aws/credentials
2019-08-12 12:36:18,330 - MainThread - botocore.loaders - DEBUG - Loading JSON file: /home/
ec2-user/venv/lib/python3.7/site-packages/botocore/data/endpoints.json
2019-08-12 12:36:18,334 - MainThread - botocore.hooks - DEBUG - Event choose-service-name:
calling handler <function handle_service_name_alias at 0x7fdf1898eb70>
2019-08-12 12:36:18,337 - MainThread - botocore.hooks - DEBUG - Event creating-client-
class.iam: calling handler <function add_generate_presigned_url at 0x7fdf18a028c8>
2019-08-12 12:36:18,337 - MainThread - botocore.regions - DEBUG - Using partition endpoint
for iam, us-west-2: aws-global
2019-08-12 12:36:18,337 - MainThread - botocore.args - DEBUG - The s3 config key is not a
dictionary type, ignoring its value of: None
2019-08-12 12:36:18,340 - MainThread - botocore.endpoint - DEBUG - Setting iam timeout as
(60, 60)
2019-08-12 12:36:18,341 - MainThread - botocore.loaders - DEBUG - Loading JSON file: /home/
ec2-user/venv/lib/python3.7/site-packages/botocore/data/_retry.json
2019-08-12 12:36:18,341 - MainThread - botocore.client - DEBUG - Registering retry handlers
for service: iam
2019-08-12 12:36:18,342 - MainThread - botocore.hooks - DEBUG - Event before-parameter-
build.iam.ListGroups: calling handler <function generate_idempotent_uuid at 0x7fdf189b10d0>
2019-08-12 12:36:18,342 - MainThread - botocore.hooks - DEBUG - Event before-
call.iam.ListGroups: calling handler <function inject_api_version_header_if_needed at
0x7fdf189b2a60>
2019-08-12 12:36:18,343 - MainThread - botocore.endpoint - DEBUG - Making
request for OperationModel(name=ListGroups) with params: {'url_path': '/',
```

```
'query_string': '', 'method': 'POST', 'headers': {'Content-Type': 'application/x-www-form-urlencoded; charset=utf-8', 'User-Agent': 'aws-cli/1.16.215 Python/3.7.3 Linux/4.14.133-113.105.amzn2.x86_64 botocore/1.12.205'}, 'body': {'Action': 'ListGroupsWithPrefix', 'Version': '2010-05-08'}, 'url': 'https://iam.amazonaws.com/', 'context': {'client_region': 'aws-global', 'client_config': <botocore.config.Config object at 0x7fdf16e9a4a8>, 'has_streaming_input': False, 'auth_type': None}}
2019-08-12 12:36:18,343 - MainThread - botocore.hooks - DEBUG - Event request-created.iam.ListGroups: calling handler <bound method RequestSigner.handler of <botocore.signers.RequestSigner object at 0x7fdf16e9a470>>
2019-08-12 12:36:18,343 - MainThread - botocore.hooks - DEBUG - Event choose-signer.iam.ListGroups: calling handler <function set_operation_specific_signer at 0x7fdf18996f28>
2019-08-12 12:36:18,343 - MainThread - botocore.auth - DEBUG - Calculating signature using v4 auth.
2019-08-12 12:36:18,343 - MainThread - botocore.auth - DEBUG - CanonicalRequest:
POST
/

content-type:application/x-www-form-urlencoded; charset=utf-8
host:iam.amazonaws.com
x-amz-date:20190812T193618Z

content-type;host;x-amz-date
5f776d91EXAMPLE9b8cb5eb5d6d4a787a33ae41c8cd6eEXAMPLEca69080e1elf
2019-08-12 12:36:18,344 - MainThread - botocore.auth - DEBUG - StringToSign:
AWS4-HMAC-SHA256
20190812T193618Z
20190812/us-east-1/iam/aws4_request
ab7e367eEXAMPLE2769f178ea509978cf8bfa054874b3EXAMPLE8d043fab6cc9
2019-08-12 12:36:18,344 - MainThread - botocore.auth - DEBUG - Signature:
d85a0EXAMPLEb40164f2f539cdc76d4f294fe822EXAMPLE18ad1ddf58a1a3ce7
2019-08-12 12:36:18,344 - MainThread - botocore.endpoint - DEBUG - Sending http request:
<AWSPreparedRequest stream_output=False, method=POST, url=https://iam.amazonaws.com/,
headers={'Content-Type': b'application/x-www-form-urlencoded; charset=utf-8',
'User-Agent': b'aws-cli/1.16.215 Python/3.7.3 Linux/4.14.133-113.105.amzn2.x86_64
botocore/1.12.205', 'X-Amz-Date': b'20190812T193618Z', 'Authorization': b'AWS4-HMAC-
SHA256 Credential=AKIA01234567890EXAMPLE-east-1/iam/aws4_request, SignedHeaders=content-
type;host;x-amz-date, Signature=d85a07692aceb401EXAMPLEealb18ad1ddf58a1a3ce7EXAMPLE',
'Content-Length': '36'}>
2019-08-12 12:36:18,344 - MainThread - urllib3.util.retry - DEBUG - Converted retries
value: False -> Retry(total=False, connect=None, read=None, redirect=0, status=None)
2019-08-12 12:36:18,344 - MainThread - urllib3.connectionpool - DEBUG - Starting new HTTPS
connection (1): iam.amazonaws.com:443
2019-08-12 12:36:18,664 - MainThread - urllib3.connectionpool - DEBUG - https://
iam.amazonaws.com:443 "POST / HTTP/1.1" 200 570
2019-08-12 12:36:18,664 - MainThread - botocore.parsers - DEBUG - Response headers: {'x-
amzn-RequestId': '74c11606-bd38-11e9-9c82-559da0adb349', 'Content-Type': 'text/xml',
'Content-Length': '570', 'Date': 'Mon, 12 Aug 2019 19:36:18 GMT'}
2019-08-12 12:36:18,664 - MainThread - botocore.parsers - DEBUG - Response body:
b'<ListGroupsWithPrefixResponse xmlns="https://iam.amazonaws.com/doc/2010-05-08/">\n
  <ListGroupsWithPrefixResult>\n
    <IsTruncated>false</IsTruncated>\n
    <Groups>\n
      <member>\n
        <Path>/</Path>\n
        <GroupName>MyTestGroup</GroupName>\n
        <Arn>arn:aws:iam::123456789012:group/MyTestGroup</Arn>\n
        <GroupId>AGPA1234567890EXAMPLE</GroupId>\n
        <CreateDate>2019-08-12T19:34:04Z</
CreateDate>\n
      </member>\n
    </Groups>\n
  </ListGroupsWithPrefixResult>\n
  <ResponseMetadata>\n
    <RequestId>74c11606-bd38-11e9-9c82-559da0adb349</RequestId>\n
    </ResponseMetadata>\n
</ListGroupsWithPrefixResponse>\n'
2019-08-12 12:36:18,665 - MainThread - botocore.hooks - DEBUG - Event needs-
retry.iam.ListGroups: calling handler <botocore.retryhandler.RetryHandler object at
0x7fdf16e9a780>
2019-08-12 12:36:18,665 - MainThread - botocore.retryhandler - DEBUG - No retry needed.
2019-08-12 12:36:18,665 - MainThread - botocore.hooks - DEBUG - Event after-
call.iam.ListGroups: calling handler <function json_decode_policies at 0x7fdf189b1d90>
{
  "Groups": [
    {
```

```
    "Path": "/",
    "GroupName": "MyTestGroup",
    "GroupId": "AGPA123456789012EXAMPLE",
    "Arn": "arn:aws:iam::123456789012:group/MyTestGroup",
    "CreateDate": "2019-08-12T19:34:04Z"
  }
}
```

I get the error "command not found" when I run aws.

Possible cause: The operating system "path" was not updated during installation.

This error means that the operating system can't find the AWS CLI program. The installation might be incomplete.

If you use `pip` to install the AWS CLI, you might need to add the folder that contains the `aws` program to your operating system's `PATH` environment variable, or change its mode to make it executable.

You might need to add the `aws` executable to your operating system's `PATH` environment variable. Follow the steps in the appropriate procedure:

- **Windows** – [Add the AWS CLI version 1 Executable to Your Command Line Path \(p. 28\)](#)
- **macOS** – [Add the AWS CLI version 1 Executable to Your macOS Command Line Path \(p. 25\)](#)
- **Linux** – [Add the AWS CLI version 1 Executable to Your Command Line Path \(p. 21\)](#)

I get "access denied" errors.

Possible cause: The AWS CLI program file doesn't have "run" permission.

On Linux or macOS, ensure that the `aws` program has run permissions for the calling user. Typically, the permissions are set to `755`.

To add run permission for your user, run the following command, substituting `~/.local/bin/aws` with the path to the program on your computer.

```
$ chmod +x ~/.local/bin/aws
```

Possible cause: Your IAM identity doesn't have permission to perform the operation.

When you run a AWS CLI command, AWS operations are performed on your behalf, using credentials that associate you with an IAM user or role. The policies attached to that IAM user or role must grant you permission to call the API actions that correspond to the commands that you run with the AWS CLI.

Most commands call a single action with a name that matches the command name. However, custom commands like `aws s3 sync` call multiple APIs. You can see which APIs a command calls by using the `--debug` option.

If you are sure that the user or role has the proper permissions assigned by policy, ensure that your AWS CLI command is using the credentials you expect. See the [next section about credentials](#) (p. 138) to verify that the credentials the AWS CLI is using are the ones you expect.

For information about assigning permissions to IAM users and roles, see [Overview of Access Management: Permissions and Policies](#) in the *IAM User Guide*.

I get an "invalid credentials" error.

Possible cause: The AWS CLI is reading credentials from an unexpected location.

The AWS CLI might be reading credentials from a different location than you expect. You can run `aws configure list` to confirm which credentials are used.

The following example shows how to check the credentials used for the default profile.

```
$ aws configure list
      Name                               Value                               Type      Location
      ----                               -
      profile                           <not set>                          None      None
      access_key                       *****XYVA                       shared-credentials-file
      secret_key                       *****ZAGY                       shared-credentials-file
      region                            us-west-2                         config-file  ~/.aws/config
```

The following example shows how to check the credentials of a named profile.

```
$ aws configure list --profile saanvi
      Name                               Value                               Type      Location
      ----                               -
      profile                           saanvi                             manual    --profile
      access_key                       *****                       shared-credentials-file
      secret_key                       *****                       shared-credentials-file
      region                            us-west-2                         config-file  ~/.aws/config
```

Possible cause: Your computer's clock is out of sync.

If you are using valid credentials, your clock may be out of sync. On Linux or macOS, run `date` to check the time.

```
$ date
```

If your system clock is not correct within a few minutes, use `ntpd` to sync it.

```
$ sudo service ntpd stop
$ sudo ntpdate time.nist.gov
$ sudo service ntpd start
$ ntpstat
```

On Windows, use the date and time options in the Control Panel to configure your system clock.

I get a "signature does not match" error.

When the AWS CLI runs a command, it sends an encrypted request to the AWS servers to perform the appropriate AWS service operations. Your credentials (the access key and secret key) are involved in the encryption and enable AWS to authenticate the person making the request. There are several things that can interfere with the correct operation of this process, as follows.

Possible cause: Your clock is out of sync with the AWS servers.

To help protect against [replay attacks](#), the current time can be used during the encryption/decryption process. If the time of the client and server disagree by more than the allowed amount, the process can fail and the request is rejected. This can also happen when you run a command in a virtual machine whose clock is out of sync with the host machine's clock. One possible cause is when the virtual machine hibernates and takes some time after waking up to resync the clock with the host machine.

On Linux or macOS, run `date` to check the time.

```
$ date
```

If your system clock is not correct within a few minutes, use `ntpd` to sync it.

```
$ sudo service ntpd stop
$ sudo ntpdate time.nist.gov
$ sudo service ntpd start
$ ntpstat
```

On Windows, use the date and time options in the Control Panel to configure your system clock.

Possible cause: Your operating system is mishandling AWS secret keys that contain certain special characters.

If your AWS secret key includes certain special characters, such as `-`, `+`, `/`, or `%`, some operating system variants process the string improperly and cause the secret key string to be interpreted incorrectly.

If you process your access keys and secret keys using other tools or scripts, such as tools that build the credentials file on a new instance as part of its creation, those tools and scripts might have their own handling of special characters that causes them to be transformed into something that AWS no longer recognizes.

The easy solution is to regenerate the secret key to get one that does not include the special character.

AWS CLI User Guide Document History

The following table describes important additions to the *AWS Command Line Interface User Guide*, beginning in January 2019. For notification about updates to this documentation, you can subscribe to the RSS feed.

update-history-change	update-history-description	update-history-date
Developer preview release for AWS CLI version 2	Announcing preview release of AWS CLI version 2. Added instructions about installing version 2. Add Migration topic to discuss differences between versions 1 and 2.	November 7, 2019
Added support for AWS Single Sign-On to AWS CLI named profiles	AWS CLI version 2 adds support for creating a named profile that can directly login to an AWS SSO user account and get AWS temporary credentials for use in subsequent AWS CLI commands.	November 7, 2019
Announcing deprecation of support for Python 2.6 and Python 3.3	Added notices about the upcoming deprecation of support for Python versions 2.6 and 3.3. Starting with version 1.17 to be released on January 10th, 2020, the CLI will require Python 2.7, Python 3.4, or a newer version of Python to operate.	October 9, 2019
New MFA section	Added a new section describing how to access the CLI using multi-factor authentication and roles.	May 3, 2019
Update to "Using the CLI" section	Major improvements and additions to the usage instructions and procedures.	March 7, 2019
Update to "Installing the CLI" section	Major improvements and additions to the CLI installation instructions and procedures.	March 7, 2019
Update to "Configuring the CLI" section	Major improvements and additions to the CLI configuration instructions and procedures.	March 7, 2019