
AWS Command Line Interface

User Guide



AWS Command Line Interface: User Guide

Copyright © 2017 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 in this Guide	2
About Amazon Web Services	3
Install	4
Linux	5
Python	6
on Amazon Linux 2017	6
Installing Pip	7
Installing the AWS CLI with Pip	8
Adding the AWS CLI Executable to your Command Line Path	9
Windows	9
MSI Installer	10
Windows	11
Adding the AWS CLI Executable to your Command Line Path	11
macOS	12
Install Python, pip, and the AWS CLI on macOS	12
Adding the AWS CLI Executable to your Command Line Path	13
Virtualenv	13
Bundled Installer	14
Prerequisites	15
Install the AWS CLI Using the Bundled Installer	15
Install the AWS CLI Without Sudo (Linux, macOS, or Unix)	16
Uninstalling	16
Configure	17
Quick Configuration	17
Configuration Settings and Precedence	18
Configuration and Credential Files	19
Named Profiles	20
Using Profiles with the AWS CLI	21
Environment Variables	21
Command Line Options	22
Instance Metadata	23
Using an HTTP Proxy	23
Authenticating to a Proxy	23
Using a proxy on EC2 Instances	24
Assuming a Role	24
Configuring and Using a Role	24
Using Multifactor Authentication	26
Cross Account Roles	26
Clearing Cached Credentials	26
Command Completion	27
Identify Your Shell	27
Locate the AWS Completer	27
Enable Command Completion	28
Test Command Completion	28
Tutorial: Using Amazon EC2	29
Install the AWS CLI	29
Windows	29
Linux, macOS, or Unix	30
Configure the AWS CLI	30
Create a Security Group, Key Pair, and Role for the EC2 Instance	31
Launch and Connect to the Instance	31
Using the AWS CLI	33
Getting Help	33

AWS CLI Documentation	36
API Documentation	36
Command Structure	37
Specifying Parameter Values	37
Common Parameter Types	38
Using JSON for Parameters	39
Quoting Strings	41
Loading Parameters from a File	41
Generate CLI Skeleton	43
Controlling Command Output	45
How to Select the Output Format	46
How to Filter the Output with the --query Option	46
JSON Output Format	49
Text Output Format	49
Table Output Format	50
Shorthand Syntax	52
Structure Parameters	52
List Parameters	52
Pagination	53
Working with Services	55
DynamoDB	55
Amazon EC2	57
Using Key Pairs	57
Using Security Groups	59
Using Instances	62
Amazon Glacier	68
Create an Amazon Glacier Vault	68
Prepare a File for Uploading	68
Initiate a Multipart Upload and Upload Files	69
Complete the Upload	70
AWS Identity and Access Management	71
Create New IAM Users and Groups	72
Set an IAM Policy for an IAM User	73
Set an Initial Password for an IAM User	73
Create Security Credentials for an IAM User	74
Amazon S3	74
Using High-Level Amazon S3 Commands	75
Using API Level (s3api) Commands	79
Amazon SNS	80
Create a Topic	80
Subscribe to a Topic	80
Publish to a Topic	81
Unsubscribe from a Topic	81
Delete a Topic	81
Amazon SWF	82
List of Amazon SWF Commands	82
Working with Amazon SWF Domains	84
Troubleshooting	89

What Is the AWS Command Line Interface?

The AWS CLI is an open source tool built on top of the AWS SDK for Python (Boto) that provides commands for interacting with AWS services. With minimal configuration, you can start using all of the functionality provided by the AWS Management Console from your favorite terminal program.

- **Linux shells** – Use common shell programs such as `Bash`, `Zsh`, and `tsch` to run commands in Linux, macOS, or Unix.
- **Windows command line** – On Microsoft Windows, run commands in either PowerShell or the Windows Command Processor.
- **Remotely** – Run commands on Amazon EC2 instances through a remote terminal such as PuTTY or SSH, or with Amazon EC2 systems manager.

The AWS CLI provides direct access to AWS services' public APIs. Explore a service's capabilities with the AWS CLI, and develop shell scripts to manage your resources. Or take what you've learned to develop programs in other languages with the AWS SDK.

In addition to the low level, API equivalent commands, the AWS CLI also provides customizations for several services. Customizations are higher level commands that simplify using a service with a complex API. For example, the `aws s3` set of commands provide a familiar syntax for managing files in 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 may want to use the AWS SDK, a toolkit, or the AWS Tools for Windows PowerShell.

- [AWS Tools for Windows 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 Toolkit for Eclipse](#)
- [AWS Toolkit for Visual Studio](#)
- [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 in this Guide

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

- **Prompt** – The command prompt is displayed as a dollar sign ('\$'). 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 beneath user input without any special formatting.

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, type **aws configure** at the command line and press **Enter**. **aws configure** is the command. This command is interactive, so the AWS CLI outputs lines of texts, prompting you to enter additional information. Enter each of your access keys in turn and press **Enter**. Then, enter a region name in the format shown, press **Enter**, and 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 non-interactive 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 press **Enter**. The name of the security group, `my-sg` is replaceable. In this case, you can use the group name as shown, but you will 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*. If an argument must be replaced, it will be noted in the text describing 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 Command Line Interface

The primary distribution method for the AWS CLI on Linux, Windows, and macOS is `pip`, a package manager for Python that provides an easy way to install, upgrade, and remove Python packages and their dependencies.

Current AWS CLI Version

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

Requirements

- Python 2 version 2.6.5+ or Python 3 version 3.3+
- Windows, Linux, macOS, or Unix

Note

Older versions of Python may not work with all AWS services. If you see `InsecurePlatformWarning` or deprecation notices when you install or use the AWS CLI, update to a recent version.

If you already have `pip` and a supported version of Python, you can install the AWS CLI with the following command:

```
$ pip install awscli --upgrade --user
```

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

If you encounter issues when you attempt to install the AWS CLI with `pip`, you can [install the AWS CLI in a virtual environment \(p. 13\)](#) to isolate the tool and its dependencies, or use a different version of Python than you normally do.

Standalone Installers

For offline or automated installations on Linux, macOS, or Unix, try the [bundled installer \(p. 14\)](#). The bundled installer includes the AWS CLI, its dependencies, and a shell script that performs the installation for you.

On Windows, you can also use the [MSI installer \(p. 10\)](#). Both of these methods simplify the initial installation, with the tradeoff of being more difficult to upgrade when a new version of the AWS CLI is released.

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

- **Linux** – [Adding the AWS CLI Executable to your Command Line Path \(p. 9\)](#)
- **Windows** – [Adding the AWS CLI Executable to your Command Line Path \(p. 11\)](#)
- **macOS** – [Adding the AWS CLI Executable to your Command Line Path \(p. 13\)](#)

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

```
$ aws --version
aws-cli/1.11.84 Python/3.6.2 Linux/4.4.0-59-generic botocore/1.5.47
```

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.

```
$ pip install awscli --upgrade --user
```

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

```
$ pip uninstall awscli
```

If you don't have Python and `pip`, use the procedure for your operating system:

Sections

- [Install the AWS Command Line Interface on Linux \(p. 5\)](#)
- [Install the AWS Command Line Interface on Microsoft Windows \(p. 9\)](#)
- [Install the AWS Command Line Interface on macOS \(p. 12\)](#)
- [Install the AWS Command Line Interface in a Virtual Environment \(p. 13\)](#)
- [Install the AWS CLI Using the Bundled Installer \(Linux, macOS, or Unix\) \(p. 14\)](#)

Install the AWS Command Line Interface on Linux

You can install the AWS Command Line Interface and its dependencies on most Linux distributions with `pip`, a package manager for Python.

Important

The `awscli` package is available in repositories for other package managers such as APT and yum, but it is not guaranteed to be the latest version unless you get it from `pip` or use the [bundled installer \(p. 14\)](#)

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`.

```
$ pip --version
```

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

```
$ python --version
```

or

```
$ python3 --version
```

If you don't have Python 2 version 2.6.5+ or Python 3 version 3.3+, [install Python \(p. 6\)](#). Otherwise, install pip and the AWS CLI.

Installing Python on Linux

If your distribution did not come with Python, or came with an older version, install Python before installing pip and the AWS CLI.

To install Python 3 on Linux

1. Check to see if Python is already installed:

```
$ python --version
```

Note

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

2. If Python 2.7 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:

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

- On Red Hat and derivatives, use yum:

```
$ sudo yum install python
```

- On SUSE and derivatives, use zypper:

```
$ sudo zypper install python3
```

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

```
$ python3 --version  
Python 3.6.2
```

Installing the AWS Command Line Interface on Amazon Linux 2017

The AWS CLI comes pre-installed on the [Amazon Linux AMI](#). Check the currently installed version with `aws --version`.

```
$ aws --version
aws-cli/1.11.83 Python/2.7.12 Linux/4.9.20-11.31.amzn1.x86_64 botocore/1.5.46
```

You can use `sudo yum update` to get the latest version available in the yum repository, but this may not be the latest version. Use `pip` to get the latest version.

To upgrade the AWS CLI on Amazon Linux (root)

1. Use `pip install` to install the latest version of the AWS CLI.

```
$ sudo pip install --upgrade awscli
```

2. Verify the new version with `aws --version`.

```
$ aws --version
aws-cli/1.11.85 Python/2.7.12 Linux/4.9.20-11.31.amzn1.x86_64 botocore/1.5.48
```

If you don't have root privileges, install the AWS CLI in user mode.

To upgrade the AWS CLI on Amazon Linux (user)

1. Use `pip install` to install the latest version of the AWS CLI.

```
$ sudo pip 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 `~/ .bashrc` to maintain the change between sessions.

3. Verify the new version with `aws --version`.

```
$ aws --version
aws-cli/1.11.85 Python/2.7.12 Linux/4.9.20-11.31.amzn1.x86_64 botocore/1.5.48
```

Sections

- [Installing Pip \(p. 7\)](#)
- [Installing the AWS CLI with Pip \(p. 8\)](#)
- [Adding the AWS CLI Executable to your Command Line Path \(p. 9\)](#)

Installing Pip

If you don't have `pip`, install `pip` with the script provided by the Python Packaging Authority.

To install pip

1. Download the installation script from [pypa.io](https://bootstrap.pypa.io/get-pip.py):

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

The script downloads and installs the latest version of pip and another required package named `setuptools`.

2. Run the script with Python:

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

3. Add the executable path to your PATH variable: `~/.local/bin`

To modify your PATH variable (Linux, macOS, or Unix)

1. Find your shell's profile script in your user folder. If you are 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 profile into your current session.

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

4. Verify that pip is installed correctly.

```
$ pip --version  
pip 8.1.2 from ~/.local/lib/python3.4/site-packages (python 3.4)
```

Installing the AWS CLI with Pip

Use pip to install the AWS CLI.

```
$ pip install awscli --upgrade --user
```

Verify that the AWS CLI installed correctly.

```
$ aws --version  
aws-cli/1.11.84 Python/3.6.2 Linux/4.4.0-59-generic botocore/1.5.47
```

If you get an error, see [Troubleshooting AWS CLI Errors \(p. 89\)](#).

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

```
$ pip install awscli --upgrade --user
```

Adding the AWS CLI Executable to your Command Line Path

After installing with `pip`, you may need to add the `aws` executable to your OS's `PATH` environment variable.

Example AWS CLI install location - Linux with `pip` (user mode)

```
~/.local/bin
```

If you 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 `which python`.

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

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

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

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

1. Find your shell's profile script in your user folder. If you are 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 profile into your current session.

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

Install the AWS Command Line Interface on Microsoft Windows

You can install the AWS CLI on Windows with a standalone installer or `pip`, a package manager for Python. If you already have `pip`, follow the instructions in the main [installation topic \(p. 4\)](#).

Sections

- [MSI Installer \(p. 10\)](#)

- [Install Python, pip, and the AWS CLI on Windows \(p. 11\)](#)
- [Adding the AWS CLI Executable to your Command Line Path \(p. 11\)](#)

MSI Installer

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

When updates are released, you must repeat the installation process to get the latest version of the AWS CLI. If you prefer to update frequently, consider [using pip \(p. 11\)](#) for easier updates.

To install the AWS CLI 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 \(32-bit\)](#)

Note

The MSI installer for the AWS CLI does not work with Windows Server 2008 (version 6.0.6002). Use [pip \(p. 11\)](#) to install with this version of Windows.

2. Run the downloaded MSI installer.
3. Follow the instructions that appear.

The CLI installs to `C:\Program Files\Amazon\AWSCLI (64-bit)` or `C:\Program Files (x86)\Amazon\AWSCLI (32-bit)` by default. To confirm the installation, use the `aws --version` command at a command prompt (open the START menu and search for "cmd" if you're not sure where the command prompt is installed).

```
> aws --version
aws-cli/1.11.84 Python/3.6.2 Windows/7 botocore/1.5.47
```

Don't include the prompt symbol ('>' 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 executable, you may need to re-open the command prompt or [add the installation directory to your PATH \(p. 11\)](#) environment variable manually.

Updating an MSI Installation

The AWS CLI is updated regularly. Check out 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 detailed above.

Uninstalling

To uninstall the AWS CLI, open the Control Panel and select *Programs and Features*. Select the entry named *AWS Command Line Interface* and click *Uninstall* to launch the uninstaller. Confirm that you wish to uninstall the AWS CLI when prompted.

You can also launch the *Programs and Features* menu from the command line with the following command:

```
> appwiz.cpl
```

Install Python, pip, and the AWS CLI on Windows

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

To install Python 3.6 and pip (Windows)

1. Download the Python 3.6 Windows x86-64 executable installer from the [downloads page](#) of [Python.org](#).
2. Run the installer.
3. Choose **Add Python 3.6 to PATH**.
4. Choose **Install Now**.

The installer installs Python in your user folder and adds its executable directories to your user path.

To install the AWS CLI with pip (Windows)

1. Open the Windows Command Processor from the Start menu.
2. Verify that Python and pip are both installed correctly with the following commands:

```
C:\Windows\System32> python --version
Python 3.6.2
C:\Windows\System32> pip --version
pip 9.0.1 from c:\users\myname\appdata\local\programs\python\python36\lib\site-packages
(python 3.6)
```

3. Install the AWS CLI using pip:

```
C:\Windows\System32> pip install awscli
```

4. Verify that the AWS CLI is installed correctly:

```
C:\Windows\System32> aws --version
aws-cli/1.11.84 Python/3.6.2 Windows/7 botocore/1.5.47
```

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

```
C:\Windows\System32> pip install --user --upgrade awscli
```

Adding the AWS CLI Executable to your Command Line Path

After installing with pip, add the aws executable to your OS's PATH environment variable. With an MSI installation, this should happen automatically, but you may need to set it manually if the aws command is not working.

- **Python 3.6 and pip** – %USERPROFILE%\AppData\Local\Programs\Python\Python36\Scripts
- **MSI installer (64-bit)** – C:\Program Files\Amazon\AWSCLI
- **MSI installer (32-bit)** – C:\Program Files (x86)\Amazon\AWSCLI

To modify your PATH variable (Windows)

1. Press the Windows key and type **environment variables**.
2. Choose **Edit environment variables for your account**.
3. Choose **PATH** and then choose **Edit**.
4. Add paths to the **Variable value** field, separated by semicolons. For example: **C:\existing\path;C:\new\path**
5. Choose **OK** twice to apply the new settings.
6. Close any running command prompts and re-open.

Install the AWS Command Line Interface on macOS

If you have pip, follow the instructions in the main [installation topic \(p. 4\)](#). Run `pip --version` to see if your version of macOS already includes Python and pip.

```
$ pip --version
```

Sections

- [Install Python, pip, and the AWS CLI on macOS \(p. 12\)](#)
- [Adding the AWS CLI Executable to your Command Line Path \(p. 13\)](#)

Install Python, pip, and the AWS CLI on macOS

You can install the latest version of Python and pip and then use them to install the AWS CLI.

To install the AWS CLI on macOS

1. Download and install Python 3.6 from the [downloads page](#) of [Python.org](#).
2. Install pip with the script provided by the Python Packaging Authority.

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

3. Use pip to install the AWS CLI.

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

4. Verify that the AWS CLI is installed correctly.

```
$ aws --version  
AWS CLI 1.11.84 (Python 3.6.1)
```

If the executable is not found, [add it to your command line path \(p. 13\)](#).

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

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


Adding the AWS CLI Executable to your Command Line Path

After installing with pip, you may need to add the aws executable to your OS's PATH environment variable. The location of the executable depends on where Python is installed.

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

```
~/Library/Python/3.6/bin
```

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

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

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

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

pip installs executables to the same folder that contains the Python executable. Add this folder to your PATH variable.

To modify your PATH variable (Linux, macOS, or Unix)

1. Find your shell's profile script in your user folder. If you are 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 profile into your current session.

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

Install the AWS Command Line Interface in a Virtual Environment

You can avoid requirement version conflicts with other pip packages by installing the AWS CLI in a virtual environment.

To install the AWS CLI in a virtual environment

1. Install `virtualenv` with `pip`.

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

2. Create a virtual environment.

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

You can use the `-p` option to use a Python executable other than the default.

```
$ virtualenv -p /usr/bin/python3.4 ~/cli-ve
```

3. Activate the virtual environment.

Linux, macOS, or Unix

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

Windows

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

4. Install the AWS CLI.

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

5. Verify that the AWS CLI is installed correctly.

```
$ aws --version  
aws-cli/1.11.84 Python/3.6.2 Linux/4.4.0-59-generic botocore/1.5.47
```

You can use the `deactivate` command to exit the virtual environment. Whenever you start a new session, run the activation command again.

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

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

Install the AWS CLI Using the Bundled Installer (Linux, macOS, or Unix)

On Linux, macOS, or Unix, you can also use the bundled installer to install the AWS CLI. The bundled installer includes all dependencies and can be used offline.

Important

The bundled installer does not support installing to paths that contain spaces.

Sections

- [Prerequisites \(p. 15\)](#)

- [Install the AWS CLI Using the Bundled Installer \(p. 15\)](#)
- [Install the AWS CLI Without Sudo \(Linux, macOS, or Unix\) \(p. 16\)](#)
- [Uninstalling \(p. 16\)](#)

Prerequisites

- Linux, macOS, or Unix
- Python 2 version 2.6.5+ or Python 3 version 3.3+

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 Command Line Interface on Linux \(p. 5\)](#).

Install the AWS CLI Using the Bundled Installer

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

To install the AWS CLI using the bundled installer

1. Download the [AWS CLI Bundled Installer](#).

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

2. Unzip 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 executable.

```
$ 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. For example:

```
$ sudo /usr/local/bin/python2.7 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 Without Sudo (Linux, macOS, or Unix)

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 above commands:

```
$ 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 create 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`, etc).

Uninstalling

The bundled installer does not 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
```

Configuring the AWS CLI

This section explains how to configure settings that the AWS Command Line Interface uses when interacting with AWS, such as your security credentials and the default region.

Note

The AWS CLI signs requests on your behalf, and includes a date in the signature. Ensure that your computer's date and time are set correctly; if not, the date in the signature may not match the date of the request, and AWS rejects the request.

Sections

- [Quick Configuration \(p. 17\)](#)
- [Configuration Settings and Precedence \(p. 18\)](#)
- [Configuration and Credential Files \(p. 19\)](#)
- [Named Profiles \(p. 20\)](#)
- [Environment Variables \(p. 21\)](#)
- [Command Line Options \(p. 22\)](#)
- [Instance Metadata \(p. 23\)](#)
- [Using an HTTP Proxy \(p. 23\)](#)
- [Assuming a Role \(p. 24\)](#)
- [Command Completion \(p. 27\)](#)

Quick Configuration

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
```

The AWS CLI will prompt you for four pieces of information. AWS Access Key ID and AWS Secret Access Key are your account credentials.

To get the access key ID and secret access key for an IAM user

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. We recommend that you use IAM access keys instead of AWS account root user access keys. IAM lets you securely control access to AWS services and resources in your AWS account.

The only time that you can view or download the secret access keys 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 [Delegating Permissions to Administer IAM Users, Groups, and Credentials](#) in the *IAM User Guide*.

1. Open the [IAM console](#).
2. In the navigation pane of the console, choose **Users**.
3. Choose your IAM user name (not the check box).
4. Choose the **Security credentials** tab and then choose **Create access key**.
5. To see the new access key, choose **Show**. Your credentials will look something like this:
 - Access key ID: AKIAIOSFODNN7EXAMPLE
 - Secret access key: wJalrXUtnFEMI/K7MDENG/bPxrFcYEXAMPLEKEY
6. To download the key pair, choose **Download .csv file**. Store the keys in a secure location.

Keep the keys confidential in order to protect your 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.

Related topics

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

Default region is the name of the region you want to make calls against by default. This is usually the region closest to you, but it can be any region. For example, type `us-west-2` to use US West (Oregon).

Note

You must specify an AWS region when using the AWS CLI. For a list of services and 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.

Default output format can be either `json`, `text`, or `table`. If you don't specify an output format, `json` is used.

If you have multiple profiles, you can configure additional, named profiles by using the `--profile` option.

```
$ aws configure --profile user2
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
```

To update any of your settings, simply run `aws configure` again and enter new values as appropriate. The next sections contain more information on the files that `aws configure` creates, additional settings, and named profiles.

Configuration Settings and Precedence

The AWS CLI uses a *provider chain* to look for AWS credentials in a number of different places, including system or user environment variables and local AWS configuration files.

The AWS CLI looks for credentials and configuration settings in the following order:

1. **Command line options** – region, output format and profile can be specified as command options to override default settings.
2. **Environment variables (p. 21)** – `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `AWS_SESSION_TOKEN`.
3. **The AWS credentials file** – located at `~/.aws/credentials` on Linux, macOS, or Unix, or at `C:\Users\USERNAME\.aws\credentials` on Windows. This file can contain multiple named profiles in addition to a default profile.
4. **The CLI configuration file** – typically located at `~/.aws/config` on Linux, macOS, or Unix, or at `C:\Users\USERNAME\.aws\config` on Windows. This file can contain a default profile, named profiles, and CLI specific configuration parameters for each.
5. **Container credentials** – provided by Amazon EC2 Container Service on container instances when you [assign a role to your task](#).
6. **Instance profile credentials** – these credentials can be used on EC2 instances with an assigned instance role, and are delivered through the Amazon EC2 metadata service.

Configuration and Credential Files

The CLI stores credentials specified with `aws configure` in a local file named `credentials` in a folder named `.aws` in your home directory. Home directory location varies but can be referred to using the environment variables `%UserProfile%` in Windows and `$HOME` or `~` (tilde) in Unix-like systems.

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

Linux, macOS, or Unix

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

Windows

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

In order to separate credentials from less sensitive options, region and output format are stored in a separate file named `config` in the same folder.

The default file location for the config file can be overridden by setting the `AWS_CONFIG_FILE` environment variable to another local path. See [Environment Variables \(p. 21\)](#) for details.

Storing Credentials in Config

The AWS CLI will also read credentials from the config file. If you want to keep all of your profile settings in a single file, you can. 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 will take precedence.

If you use one of the SDKs in addition to the AWS CLI, you may notice additional warnings if credentials are not 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/bPxRfiCYEXAMPLEKEY
```

~/.aws/config

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

The following settings are supported.

aws_access_key_id – AWS access key.

aws_secret_access_key – AWS secret key.

aws_session_token – AWS session token. A session token is only required if you are using temporary security credentials.

region – AWS region.

output – output format (json, text, or table)

Named Profiles

The AWS CLI supports *named profiles* 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:

~/.aws/credentials

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

[user2]
aws_access_key_id=AKIAI44QH8DHBEXAMPLE
aws_secret_access_key=je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY
```

Each profile uses different credentials—perhaps from two different IAM users—and can also use different regions and output formats.

~/.aws/config

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

[profile user2]
region=us-east-1
output=text
```

Important

The AWS credentials file uses a different naming format than the CLI config file for named profiles. Do not include the 'profile ' prefix when configuring a named profile in the AWS credentials file.

Using Profiles with the AWS CLI

To use a named profile, add the `--profile` option to your command. The following example lists running instances using the `user2` profile from the previous section.

```
$ aws ec2 describe-instances --profile user2
```

If you are going 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, macOS, or Unix

```
$ export AWS_PROFILE=user2
```

Windows

```
> set AWS_PROFILE=user2
```

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. More on variables in the next section.

Environment Variables

Environment variables override configuration and credential files and can be useful for scripting or temporarily setting a named profile as the default.

The AWS CLI supports the following environment variables.

- `AWS_ACCESS_KEY_ID` – AWS access key.
- `AWS_SECRET_ACCESS_KEY` – AWS secret key. Access and secret key variables override credentials stored in credential and config files.
- `AWS_SESSION_TOKEN` – Specify a session token if you are using temporary security credentials.
- `AWS_DEFAULT_REGION` – AWS region. This variable overrides the default region of the in-use profile, if set.
- `AWS_DEFAULT_OUTPUT` – Change the AWS CLI's output formatting to `json`, `text`, or `table`.
- `AWS_PROFILE` – name of the CLI profile to use. This can be the name of a profile stored in a credential or config file, or `default` to use the default profile.
- `AWS_CA_BUNDLE` – Specify the path to a certificate bundle to use for HTTPS certificate validation.
- `AWS_SHARED_CREDENTIALS_FILE` – Change the location of the file that the AWS CLI uses to store access keys.
- `AWS_CONFIG_FILE` – Change the location of the file that the AWS CLI uses to store configuration profiles.

The following example shows how you would configure environment variables for the default user from earlier in this guide.

Linux, macOS, or Unix

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

Windows

```
> set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
> set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
> set AWS_DEFAULT_REGION=us-west-2
```

Command Line Options

The AWS CLI uses GNU-style long command line options preceded by two hyphens. Command line options can be used to override default configuration settings for a single operation, but cannot be used to specify credentials.

The following settings can be specified at the command line.

--profile – name of a profile to use, or "default" to use the default profile.

--region – AWS region to call.

--output – output format.

--endpoint-url – The endpoint to make the call against. The endpoint can be the address of a proxy or an endpoint URL for the in-use AWS region. Specifying an endpoint is not required for normal use as the AWS CLI determines which endpoint to call based on the in-use region.

The above options override the corresponding profile settings for a single operation. Each takes a string argument with a space or equals sign ("=") separating the argument from the option name. Quotes around the argument are not required unless the argument string contains a space.

Tip

You can use the **--profile** option with `aws configure` to set up additional profiles

```
$ aws configure --profile profilename
```

Common uses for command line options include checking your resources in multiple regions and changing output format for legibility or ease of use when scripting. For example, if you are not sure which region your instance is running in you could run the `describe-instances` command against each region until you find it:

```
$ 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 |                                     ||
|+-----+-----+-----+-----+-----+-----+|
```

...

Command line option parameter types (string, boolean, etc.) are discussed in detail in the [Specifying Parameter Values for the AWS Command Line Interface \(p. 37\)](#) section later in this guide.

Instance Metadata

To use the CLI from an EC2 instance, create a role that has access to the resources needed and assign that role to the instance when it is launched. Launch the instance and check to see if the AWS CLI is already installed (it comes pre-installed on Amazon Linux).

Install the AWS CLI if necessary and configure a default region to avoid having to specify it in every command. You can set the region using `aws configure` without entering credentials by pressing 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
```

The AWS CLI will read credentials from the instance metadata. For more information, see [Granting Applications that Run on Amazon EC2 Instances Access to AWS Resources in IAM User Guide](#).

Using an HTTP Proxy

If you need to access AWS through proxy servers, you should configure the `HTTP_PROXY` and `HTTPS_PROXY` environment variables with the IP addresses for your proxy servers.

Linux, macOS, or Unix

```
$ export HTTP_PROXY=http://a.b.c.d:n
$ export HTTPS_PROXY=http://w.x.y.z:m
```

Windows

```
> set HTTP_PROXY=http://a.b.c.d:n
> set HTTPS_PROXY=http://w.x.y.z:m
```

In these examples, `http://a.b.c.d:n` and `http://w.x.y.z:m` are the IP addresses and ports for the HTTP and HTTPS proxies.

Authenticating to a Proxy

The AWS CLI supports HTTP Basic authentication. Specify a username and password in the proxy URL like this:

Linux, macOS, or Unix

```
$ export HTTP_PROXY=http://username:password@a.b.c.d:n
$ export HTTPS_PROXY=http://username:password@w.x.y.z:m
```

Windows

```
> set HTTP_PROXY=http://username:password@a.b.c.d:n  
> set HTTPS_PROXY=http://username:password@w.x.y.z:m
```

Note

The AWS CLI does not support NTLM proxies. If you use an NTLM or Kerberos proxy, you may be able to connect through an authentication proxy like [Curlm](#).

Using a proxy on EC2 Instances

If you configure a proxy on an ec2 instance launched with an IAM role, you should also set the `NO_PROXY` environment variable with the IP address 169.254.169.254, so that the AWS CLI can access the [Instance Meta Data Service \(IMDS\)](#).

Linux, macOS, or Unix

```
$ export NO_PROXY=169.254.169.254
```

Windows

```
> set NO_PROXY=169.254.169.254
```

Assuming a Role

An [IAM role](#) is a authorization tool that lets a user gain additional permissions, or get permission to perform actions in a different account.

You can configure the AWS Command Line Interface to use a role by creating a profile for the role in the `~/.aws/config` file. The following example shows a role profile named `marketingadmin` that is assumed by the default profile.

```
[profile marketingadmin]  
role_arn = arn:aws:iam::123456789012:role/marketingadmin  
source_profile = default
```

In this case, the default profile is an IAM user with credentials and permission to assume a role named `marketingadmin`. To access the role, you create a named profile. Instead of configuring this profile with credentials, you specify the ARN of the role and the name of the profile that has access to it.

Sections

- [Configuring and Using a Role \(p. 24\)](#)
- [Using Multifactor Authentication \(p. 26\)](#)
- [Cross Account Roles \(p. 26\)](#)
- [Clearing Cached Credentials \(p. 26\)](#)

Configuring and Using a Role

When you run commands using the role profile, the AWS CLI uses the source profile's credentials to call AWS Security Token Service and assume the specified role. The source profile must have permission to call `sts:assume-role` against the role, and the role must have a trust relationship with the source profile to allow itself to be assumed.

Create a new 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 target 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 to assume it. The following example shows a trust relationship that allows a role to be assumed by an IAM user named `jonsmith`:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:user/jonsmith"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Next, grant your IAM user permission to assume the role. The following example shows an AWS Identity and Access Management policy that allows an IAM user to assume the `marketingadmin` role:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "arn:aws:iam::123456789012:role/marketingadmin"
    }
  ]
}
```

The user doesn't need to have any additional permissions to run commands using the role profile. If you want your users to be able to access AWS resources without using the role, apply additional inline or managed policies for those resources.

With the role profile, role permissions, trust relationship and user permissions applied, you can assume the role at the command line by using the `profile` option, for example:

```
$ aws s3 ls --profile marketingadmin
```

To use the role for multiple calls, you can set the `AWS_PROFILE` environment variable for the current session from the command line:

Linux, macOS, or Unix

```
$ export AWS_PROFILE=marketingadmin
```

Windows

```
> set AWS_PROFILE=marketingadmin
```

For more information on configuring IAM users and roles, see [Users and Groups](#) and [Roles](#) in the *AWS Identity and Access Management User Guide*.

Using Multifactor Authentication

For additional security, you can require users to provide a one time key generated from a multifactor authentication device or mobile app when they attempt to make a call using the role profile.

First, modify the trust relationship on the role to require multifactor authentication:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::123456789012:user/jonsmith" },
      "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:

```
[profile marketingadmin]
role_arn = arn:aws:iam::123456789012:role/marketingadmin
source_profile = default
mfa_serial = arn:aws:iam::123456789012:mfa/jonsmith
```

The `mfa_serial` setting can take an ARN, as shown, or the serial number of a hardware MFA token.

Cross Account Roles

You can enable IAM users to assume roles that belong to different accounts by configuring the role as a cross account role. During role creation, set the role type to one of the options under [Role for Cross-Account Access](#) and optionally select **Require MFA**. The **Require MFA** option configures the appropriate condition in the trust relationship as described in [Using Multifactor Authentication \(p. 26\)](#).

If you use an [external ID](#) to provide additional control over who can assume a role across accounts, add an `external_id` parameter to the role profile:

```
[profile crossaccountrole]
role_arn = arn:aws:iam::234567890123:role/xaccount
source_profile = default
mfa_serial = arn:aws:iam::123456789012:mfa/jonsmith
external_id = 123456
```

Clearing Cached Credentials

When you assume a role, the AWS CLI caches the temporary credentials locally until they expire. If your role's temporary credentials are [revoked](#), you can delete the cache to force the AWS CLI to retrieve new credentials.

Linux, macOS, or Unix

```
$ rm -r ~/.aws/cli/cache
```

Windows

```
> 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. This feature is not automatically installed so you need to configure it manually.

Configuring command completion requires two pieces of information: the name of the shell you are using and the location of the `aws_completer` script.

Completion on Amazon Linux

Command completion is configured by default on instances running Amazon Linux.

Sections

- [Identify Your Shell \(p. 27\)](#)
- [Locate the AWS Completer \(p. 27\)](#)
- [Enable Command Completion \(p. 28\)](#)
- [Test Command Completion \(p. 28\)](#)

Identify Your Shell

If you are not sure which shell you are using, identify it with one of the following commands:

echo \$SHELL – show the shell's installation directory. This will usually match 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. The shell will be one of them.

```
$ 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 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, `which` will locate the completer for you.

```
$ which aws_completer
/usr/local/bin/aws_completer
```

Bundled Installer – if you used the bundled installer per the instructions in the previous section, the AWS completer will be 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
```

Enable Command Completion

Run a command to enable command completion. The command that you use to enable completion depends on the shell that you are using. You can add the command to your shell's RC file to run it each time you open a new shell.

- **bash** – use the built-in command `complete`.

```
$ complete -C '/usr/local/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 run in login shells as well.

- **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/bin/aws_zsh_completer.sh
```

The AWS CLI uses bash compatibility auto completion (`bashcompinit`) for `zsh` support. For further details, refer to 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, type in a partial command and press `tab` to see the available commands.

```
$ aws sTAB
s3          ses          sqs          sts          swf
s3api       sns          storagegateway support
```


Deploying a Development Environment in Amazon EC2 Using the AWS Command Line Interface

This tutorial details how to set up a development environment in Amazon EC2 using the AWS CLI. It includes a short version of the installation and configuration instructions, and it can be run start to finish on Windows, Linux, macOS, or Unix.

Steps

- [Install the AWS CLI \(p. 29\)](#)
- [Configure the AWS CLI \(p. 30\)](#)
- [Create a Security Group, Key Pair, and Role for the EC2 Instance \(p. 31\)](#)
- [Launch and Connect to the Instance \(p. 31\)](#)

Install the AWS CLI

You can install the AWS CLI with an installer (Windows) or by using `pip`, a package manager for Python.

Windows

1. Download the MSI installer.
 - [Download the AWS CLI MSI installer for Windows \(64-bit\)](#)
 - [Download the AWS CLI MSI installer for Windows \(32-bit\)](#)
2. Run the downloaded MSI installer.
3. Follow the instructions that appear.

Linux, macOS, or Unix

These steps require that you have a working installation of Python 2 version 2.6.5+ or Python 3 version 3.3+. If you encounter any issues using the following steps, see the full installation instructions in the [AWS Command Line Interface User Guide](#).

1. Download and run the installation script from the [pip website](#):

```
$ curl "https://bootstrap.pypa.io/get-pip.py" -o "get-pip.py"
$ python get-pip.py --user
```

2. Install the AWS CLI Using pip:

```
$ pip install awscli --user
```

Configure the AWS CLI

Run `aws configure` at the command line to set up your credentials and settings.

```
$ 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
```

The AWS CLI will prompt you for the following information:

- **AWS Access Key ID and AWS Secret Access Key** – These are your account credentials. If you don't have keys, see [How Do I Get Security Credentials?](#) in the *Amazon Web Services General Reference*.
- **Default region name** – This is the name of the region you want to make calls against by default.

Note

Use `us-west-2` for this tutorial (the AMI we will use is specific to this region). You can change the default region later by running `aws configure` again.

- **Default output format** – This format can be either `json`, `text`, or `table`. If you don't specify an output format, `json` will be used.

Run a command to verify that your credentials are configured correctly and that you can connect to AWS.

```
$ aws ec2 describe-regions --output table
-----
|                               DescribeRegions                               |
+-----+-----+-----+-----+
|                               Regions                                       |
+-----+-----+-----+-----+
|                               Endpoint                                       | RegionName |
+-----+-----+-----+-----+
| ec2.ap-south-1.amazonaws.com | ap-south-1 |
| ec2.eu-west-2.amazonaws.com  | eu-west-2  |
| ec2.eu-west-1.amazonaws.com  | eu-west-1  |
| ec2.ap-northeast-2.amazonaws.com | ap-northeast-2 |
| ec2.ap-northeast-1.amazonaws.com | ap-northeast-1 |
| ec2.sa-east-1.amazonaws.com   | sa-east-1  |
| ec2.ca-central-1.amazonaws.com | ca-central-1 |
| ec2.ap-southeast-1.amazonaws.com | ap-southeast-1 |
| ec2.ap-southeast-2.amazonaws.com | ap-southeast-2 |
+-----+-----+-----+-----+
```

ec2.eu-central-1.amazonaws.com	eu-central-1
ec2.us-east-1.amazonaws.com	us-east-1
ec2.us-east-2.amazonaws.com	us-east-2
ec2.us-west-1.amazonaws.com	us-west-1
ec2.us-west-2.amazonaws.com	us-west-2

Create a Security Group, Key Pair, and Role for the EC2 Instance

Your next step is to set up prerequisites for launching an EC2 instance that can be accessed using SSH. For more information about Amazon EC2 features, go to the [Amazon EC2 User Guide for Linux Instances](#)

To create a security group, key pair, and role

1. First, create a new security group and add a rule that allows incoming traffic over port 22 for SSH. Note the security group ID for later use.

```
$ aws ec2 create-security-group --group-name devenv-sg --description "security group
for development environment in EC2"
{
  "GroupId": "sg-b018ced5"
}
$ aws ec2 authorize-security-group-ingress --group-name devenv-sg --protocol tcp --port
22 --cidr 0.0.0.0/0
```

2. Replace the CIDR range in the above with the one that you will connect from for more security. You can use the `aws ec2 describe-security-groups` command to admire your handiwork.
3. Next, create a key pair, which allows you to connect to the instance.

```
$ aws ec2 create-key-pair --key-name devenv-key --query 'KeyMaterial' --output text >
devenv-key.pem
```

This command saves the contents of the key to a file called `devenv-key.pem`.

Windows

In the Windows Command Processor, enclose queries with double quotes instead of single quotes.

4. On Linux, you will also need to change the file mode so that only you have access to the key file.

```
$ chmod 400 devenv-key.pem
```

Launch and Connect to the Instance

Finally, you are ready to launch an instance and connect to it.

To launch and connect to the instance

1. Run the following command, replacing the security group ID output in the previous step.

```
$ aws ec2 run-instances --image-id ami-6e1a0117 --security-group-ids sg-
b018ced5 --count 1 --instance-type t2.micro --key-name devenv-key --query
'Instances[0].InstanceId'
```

```
"i-0787e4282810ef9cf"
```

The image ID `ami-6e1a0117` specifies the Amazon Machine Image (AMI) that Amazon EC2 uses to bootstrap the instance. You can find image IDs for other regions and operating systems in the [Amazon EC2 Management Console](#) Launch Instance Wizard.

Note

T2 instance types require a VPC. If you don't have a default VPC, you can specify a subnet in a custom VPC with the `--subnet-id` option. If you don't have any VPCs, choose a different instance type such as `t1.micro`.

2. The instance will take a few moments to launch. Once the instance is up and running, the following command will retrieve the public IP address that you will use to connect to the instance.

```
$ aws ec2 describe-instances --instance-ids "i-0787e4282810ef9cf" --query  
'Reservations[0].Instances[0].PublicIpAddress'  
"54.183.22.255"
```

3. To connect to the instance, use the public IP address and private key with your preferred terminal program. On Linux, macOS, or Unix, you can do this from the command line with the following command:

```
$ ssh -i devenv-key.pem ubuntu@54.183.22.255
```

If you get an error like *Permission denied (publickey)* when attempting to connect to your instance, check that the following are correct:

- **Key** – The key specified with the `-i` option must be at the path indicated and must be the private key, not the public one. Permissions on the key must be restricted to the owner.
- **User name** – The user name must match the user associated with the key pair on the instance. For Ubuntu instances, this is `ubuntu`. For Amazon Linux, it is `ec2-user`.
- **Instance** – The public IP address or DNS name of the instance. Verify that the address is public and that port 22 is open to your local machine on the instance's security group.

You can also use the `-v` option to view additional information related to the error.

SSH on Windows

On Windows, you can use the PuTTY terminal application available [here](#). Get `putty.exe` and `puttygen.exe` from the downloads page.

Use `puttygen.exe` to convert your private key to a `.ppk` file required by PuTTY. Launch `putty.exe`, enter the public IP address of the instance in the **Host Name** field, and set the connection type to SSH.

In the **Category** panel, navigate to **Connection > SSH > Auth**, and click **Browse** to select your `.ppk` file, and then click **Open** to connect.

4. The terminal will prompt you to accept the server's public key. Type `yes` and click **Enter** to complete the connection.

You've now configured a security group, created a key pair, launched an EC2 instance, and connected to it without ever leaving the command line.

Using the AWS Command Line Interface

This section introduces the common features and calling patterns used throughout the AWS Command Line Interface.

Note

The AWS CLI makes API calls to services over HTTPS. Outbound connections on TCP port 443 must be enabled in order to perform calls.

Topics

- [Getting Help with the AWS Command Line Interface \(p. 33\)](#)
- [Command Structure in the AWS Command Line Interface \(p. 37\)](#)
- [Specifying Parameter Values for the AWS Command Line Interface \(p. 37\)](#)
- [Generate CLI Skeleton and CLI Input JSON Parameters \(p. 43\)](#)
- [Controlling Command Output from the AWS Command Line Interface \(p. 45\)](#)
- [Using Shorthand Syntax with the AWS Command Line Interface \(p. 52\)](#)
- [Using the AWS Command Line Interface's Pagination Options \(p. 53\)](#)

Getting Help with the AWS Command Line Interface

To get help when using the AWS CLI, you can simply add `help` to the end of a command. For example, the following command lists help for the general AWS CLI options and the available top-level commands.

```
$ aws help
```

The following command lists the available subcommands for Amazon EC2.

```
$ aws ec2 help
```

The next example lists the detailed help for the EC2 `DescribeInstances` operation, including descriptions of its input parameters, filters, and output. Check the examples section of the help if you are not sure how to phrase a 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, pulled from the API documentation for the command's service.

```
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 – list of the command and its options. If an option is shown in square brackets, it is either optional, has a default value, or has an alternative option that can be used instead.

```
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]
```

`describe-instances` has a default behavior that describes all instances in the current account and 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` does not take a value. If there are conditions on an option's use, they should be described in the `OPTIONS` section, or shown in the examples.

Options – 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, please 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 a Owner tag

Command:

aws ec2 describe-instances --filters "Name=tag-key,Values=Owner"
...
```

Output – descriptions of each of the fields and datatypes returned 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 instance(s) associated with it. This information comes from the [API documentation for the reservation datatype](#) 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 output is rendered into JSON by the AWS CLI, it becomes an array of reservation objects, like this:

```
{
  "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 (e.g. `PublicDnsName`) and objects (e.g. `State`) that describe it.

Windows Users

Pipe the output of the help command to `more` to view the help file one page at a time. Press the space bar or Page Down to view more of the document, and q to quit.

```
> aws ec2 describe-instances help | more
```

AWS CLI Documentation

The [AWS Command Line Interface Reference](#) provides the content of all AWS CLI commands' help files, compiled and presented online for easy navigation and viewing on mobile, tablet, and desktop screens.

Help files sometimes contain links that cannot be viewed or followed from the command line view; these are preserved in the online AWS CLI reference.

API Documentation

All subcommands in the AWS CLI correspond to calls made against a service's public API. Each service with a public API, in turn, has a set of API reference documentation that can be found from the service's homepage on the [AWS Documentation website](#).

The content of an API reference varies based on how the API is constructed and which protocol is used. Typically, an API reference will contain detailed information on actions supported by the API, data sent to and from the service, and possible error conditions.

API Documentation Sections

- **Actions** – Detailed information on parameters (including constraints on length or content) and errors specific to an action. Actions correspond to subcommands in the AWS CLI.
- **Data Types** – May contain additional information about object data returned by a subcommand.
- **Common Parameters** – Detailed information about parameters that are used by all of a service's actions.
- **Common Errors** – Detailed information about errors returned by all of a service's actions.

The name and availability of each section may vary depending on the service.

Service-Specific CLIs

Some services have a separate CLI from before a single AWS CLI was created that works 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 Command Line Interface

The AWS CLI uses a multipart structure on the command line. It starts with the base call to `aws`. The next part specifies a top-level command, which often represents an AWS service supported in the AWS CLI. Each AWS service has additional subcommands that specify the operation to perform. The general CLI options, or the specific parameters for an operation, can be specified on the command line in any order. If an exclusive parameter is specified multiple times, then 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.

Specifying Parameter Values for the AWS Command Line Interface

Many parameters 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
```

Strings without any space characters may be quoted or unquoted. However, strings that include one or more space characters must be quoted. Use a single quote (') in Linux, macOS, or Unix and Windows PowerShell, or use a double quote (") in the Windows command prompt, as shown in the following examples.

Windows PowerShell, Linux, macOS, or Unix

```
$ aws ec2 create-key-pair --key-name 'my key pair'
```

Windows Command Processor

```
> aws ec2 create-key-pair --key-name "my key pair"
```

You can also use an equals sign instead of a space. This is typically only necessary if the value of the parameter starts with a hyphen:

```
$ aws ec2 delete-key-pair --key-name=-mykey
```

Topics

- [Common Parameter Types \(p. 38\)](#)
- [Using JSON for Parameters \(p. 39\)](#)
- [Quoting Strings \(p. 41\)](#)
- [Loading Parameters from a File \(p. 41\)](#)

Common Parameter Types

This section describes some of the common parameter types and the format that the services expect them to conform to. If you are having trouble with the formatting of a parameter for a specific command, check the manual by typing **help** after the command name, for example:

```
$ 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 whitespace from the ASCII character set. Strings that contain whitespace must be surrounded by quotes. Use of symbols and whitespace other than the standard space character is not recommended and may cause issues when using the AWS CLI.

Some string parameters can accept binary data from a file. See [Binary Files \(p. 42\)](#) for an example.

Timestamp – Timestamps are formatted per the ISO 8601 standard. These are sometimes referred to as "DateTime" or "Date" type parameters.

```
$ aws ec2 describe-spot-price-history --start-time 2014-10-13T19:00:00Z
```

Acceptable formats include:

- YYYY-MM-DDThh:mm:ss.sssTZD (UTC), e.g., 2014-10-01T20:30:00.000Z
- YYYY-MM-DDThh:mm:ss.sssTZD (with offset), e.g., 2014-10-01T12:30:00.000-08:00
- YYYY-MM-DD, e.g., 2014-10-01
- Unix time in seconds, e.g. 1412195400

List – One or more strings separated by spaces.

```
$ 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 command against the service without actually running a query.

```
$ aws ec2 describe-spot-price-history --dry-run
```

The output indicates whether the command was well formed or not. This command also includes a `no-dry-run` version of the parameter that can be used to explicitly indicate that the command should be run normally, although including it is not necessary as this is the default behavior.

Integer – An unsigned whole number.

```
$ aws ec2 describe-spot-price-history --max-items 5
```

Blob – Binary object. Blob parameters take a path to a local file that contains binary data. The path should not contain any protocol identifier such as `http://` or `file://`.

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 sequence of key value pairs specified in JSON or [shorthand syntax \(p. 52\)](#). The following example reads an item from a 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"
    }
  }
}
```

The next section covers JSON arguments in more detail.

Using JSON for Parameters

JSON is useful for specifying complex command line parameters. For example, the following command will list all 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"
```

The following example specifies the equivalent list of filters in 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 ("`Name`" and "`Values`" are both keys in this instance).

Note that 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, on the other hand, are only required if more than one filter is specified. A single filter version of the above command, formatted in JSON, looks like this:

```
$ aws ec2 describe-instances --filters '{"Name": "instance-type", "Values": ["t2.micro", "m1.medium"]}'
```

Some operations require data to be formatted 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.

This example shows the JSON to specify a single 20 GiB Elastic Block Store 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 like 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 either enter the JSON directly on the command line (see [Quoting Strings \(p. 41\)](#)), or save it to a file that is referenced from the command line (see [Loading Parameters from a File \(p. 41\)](#)).

When passing in large blocks of data, you might find it easier to save the JSON to a file and 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 the next section.

For more information about JSON, see [Wikipedia - JSON](#) and [RFC4627 - The application/json Media Type for JSON](#).

Quoting Strings

The way you enter JSON-formatted parameters on the command line differs depending upon your operating system. Linux, macOS, or Unix and Windows PowerShell use the single quote (') to enclose the JSON data structure, as in the following example:

```
$ aws ec2 run-instances --image-id ami-05355a6c --block-device-mappings '["DeviceName":"/dev/sdb", "Ebs":{"VolumeSize":20, "DeleteOnTermination":false, "VolumeType":"standard"}]'
```

The Windows command prompt, on the other hand, uses the double quote (") to enclose the JSON data structure. In addition, a backslash (\) escape character is required for each double quote (") within the JSON data structure itself, as in the following example:

```
> aws ec2 run-instances --image-id ami-05355a6c --block-device-mappings "[\"DeviceName\": \"\\\"/dev/sdb\\\"\", \"Ebs\": {\"VolumeSize\": 20, \"DeleteOnTermination\": false, \"VolumeType\": \"\\\"standard\\\"\"}]"
```

Windows PowerShell requires a single quote (') to enclose the JSON data structure, as well as a backslash (\) to escape each double quote (") within the JSON structure, as in the following example:

```
> aws ec2 run-instances --image-id ami-05355a6c --block-device-mappings '[\"DeviceName\": \"\\\"/dev/sdb\\\"\", \"Ebs\": {\"VolumeSize\": 20, \"DeleteOnTermination\": false, \"VolumeType\": \"\\\"standard\\\"\"}]'
```

If the value of a parameter is itself a JSON document, escape the quotes on the embedded JSON document. For example, the attribute parameter for `aws sqs create-queue` can take a `RedrivePolicy` key. The value of `RedrivePolicy` is a JSON document, which 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

To avoid the need to escape JSON strings at the command line, load the JSON from a file. Load parameters from a local file by providing the path to the file using the `file://` prefix, as in the following examples.

Linux, macOS, or Unix

```
// 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
> 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 7 you would typically have a user directory under `C:\Users\User Name\`.

JSON documents that are provided as the value of a parameter key must still be escaped:

```
$ 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": "iVg8oWa8sy7l4+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 in an Amazon S3 bucket. This allows you to access parameter files from any computer, but requires the file to be stored in a publically accessible location.

```
$ aws ec2 run-instances --image-id ami-a13d6891 --block-device-mappings http://my-bucket.s3.amazonaws.com/filename.json
```

In the preceding examples, the `filename.json` file 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 [Set an IAM Policy for an IAM User \(p. 73\)](#).

Generate CLI Skeleton and CLI Input JSON Parameters

Most AWS CLI commands support `--generate-cli-skeleton` and `--cli-input-json` parameters that you can use to store parameters in JSON and read them from a file instead of typing them at the command line.

Generate CLI Skeleton outputs JSON that outlines all of the parameters that can be specified for the operation.

To use `--generate-cli-skeleton` with `aws ec2 run-instances`

1. Execute the `run-instances` command with the `--generate-cli-skeleton` option to view the JSON skeleton.

```
$ 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
  }
}
```

2. Direct the output to a file to save the skeleton locally:

```
$ aws ec2 run-instances --generate-cli-skeleton > ec2runinst.json
```

3. Open the skeleton in a text editor and remove any parameters that you will not use:

```
{
  "DryRun": true,
  "ImageId": "",
  "KeyName": "",
  "SecurityGroups": [
    ""
  ],
  "InstanceType": "",
  "Monitoring": {
    "Enabled": true
  }
}
```

Leave the `DryRun` parameter set to `true` to use EC2's dry run feature, which lets you test your configuration without creating resources.

4. Fill in the values for the instance type, key name, security group and AMI in your default region. In this example, `ami-dfc39aef` is a 64-bit [Amazon Linux](#) image in the `us-west-2` region.


```
{
  "DryRun": true,
  "ImageId": "ami-dfc39aef",
  "KeyName": "mykey",
  "SecurityGroups": [
    "my-sg"
  ],
  "InstanceType": "t2.micro",
  "Monitoring": {
    "Enabled": true
  }
}
```

5. Pass the JSON configuration to the `--cli-input-json` parameter using the `file://` prefix:

```
$ 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 above step until the dry run error is shown.

6. Set the `DryRun` parameter to false to disable the dry run feature.

```
{
  "DryRun": false,
  "ImageId": "ami-dfc39aef",
  "KeyName": "mykey",
  "SecurityGroups": [
    "my-sg"
  ],
  "InstanceType": "t2.micro",
  "Monitoring": {
    "Enabled": true
  }
}
```

7. Run the `run-instances` command again to launch an instance:

```
$ aws ec2 run-instances --cli-input-json file://ec2runinst.json
{
  "OwnerId": "123456789012",
  "ReservationId": "r-d94a2b1",
  "Groups": [],
  "Instances": [
    ...
  ]
}
```

Controlling Command Output from the AWS Command Line Interface

This section describes the different ways that you can control the output from the AWS CLI.

Topics

- [How to Select the Output Format \(p. 46\)](#)
- [How to Filter the Output with the `--query` Option \(p. 46\)](#)

- [JSON Output Format \(p. 49\)](#)
- [Text Output Format \(p. 49\)](#)
- [Table Output Format \(p. 50\)](#)

How to Select the Output Format

The AWS CLI supports three different output formats:

- JSON (json)
- Tab-delimited text (text)
- ASCII-formatted table (table)

As explained in the [configuration \(p. 17\)](#) topic, the output format can be specified in three different ways:

- Using the `output` option in the configuration file. The following example sets the output to `text`:

```
[default]
output=text
```

- Using the `AWS_DEFAULT_OUTPUT` environment variable. For example:

```
$ export AWS_DEFAULT_OUTPUT="table"
```

- Using the `--output` option on the command line. For example:

```
$ aws swf list-domains --registration-status REGISTERED --output text
```

Note

If the output format is specified in multiple ways, the usual [AWS CLI precedence rules \(p. 18\)](#) apply. For example, using the `AWS_DEFAULT_OUTPUT` environment variable overrides any value set in the config file with `output`, and a value passed to an AWS CLI command with `--output` overrides any value set in the environment or in the config file.

JSON is best for handling the output programmatically via various languages or `jq` (a command-line JSON processor). The table format is easy for humans to read, and text format works well with traditional Unix text processing tools, such as `sed`, `grep`, and `awk`, as well as Windows PowerShell scripts.

How to Filter the Output with the `--query` Option

The AWS CLI provides built-in output filtering capabilities with the `--query` option. To demonstrate how it works, we'll first start with the default JSON output below, which describes two EBS (Elastic Block Storage) volumes attached to separate 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
  }
]
}
```

First, we can display only the first volume from the `Volumes` list with the following command.

```
$ 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
}
```

Now, we use the wildcard notation `[*]` to iterate over the entire list and also filter out three elements: `VolumeId`, `AvailabilityZone`, and `Size`. Note that the dictionary notation requires that you provide an alias for each key, like this: `{Alias1:Key1, Alias2:Key2}`. A dictionary is inherently *unordered*, so the ordering of the key-aliases within a structure may not be consistent in some cases.

```
$ 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
  }
]
```

In the dictionary notation, you can also use chained keys such as `key1.key2[0].key3` to filter elements deeply nested within the structure. The example below 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 with the list notation: `[key1, key2]`. This will format 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 (`) in order for it to be read properly.

Combined with the three output formats that will be 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, visit <http://jmespath.org/specification.html>.

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. As shown in the previous topic along with output examples, the `--query` option provides powerful ways to filter and format the AWS CLI's JSON formatted output. If you need more advanced features that may not be possible with `--query`, you can check out `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's output into tab-delimited lines. It works well with traditional Unix text tools such as `grep`, `sed`, and `awk`, as well as Windows 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
```

We strongly recommend that the text output be used along with the `--query` option to ensure consistent behavior. This is because the text format alphabetically orders output columns, and similar resources may not always have the same collection of keys. For example, a JSON representation of a Linux EC2 instance may have elements that are not present in the JSON representation of a Windows instance, or vice versa. Also, resources may 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 enable complete control over the output format. In the example below, the command pre-selects which elements to display and defines the ordering of the columns with the list notation `[key1, key2, ...]`. This gives users full confidence that the correct key values will always be 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
```

Below is an example of how `grep` and `awk` can be used along with a text output from `aws ec2 describe-instances` command. The first command displays the Availability Zone, state, and instance ID of each instance in text output. The second command outputs 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 next command shows a similar example for all stopped instances and takes it one step further 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 is useful in Windows PowerShell as well. Because AWS CLI's text output is tab-delimited, it is easily split into an array in PowerShell using the ``t` delimiter. The following command displays the value of the third column (`InstanceId`) if the first column (`AvailabilityZone`) matches `us-west-2a`.

```
> 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
```

Table Output Format

The `table` format produces human-readable representations of AWS CLI output. Here is an example:

```
$ aws ec2 describe-volumes --output table
-----
|                               DescribeVolumes                               |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|                               Volumes                                       |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|                               |                               |                               |
```

AvailabilityZone	VolumeId	VolumeType	CreateTime	Size	SnapshotId	State	
us-west-2a	vol-e11a5288	standard	2013-09-17T00:55:03.000Z	30	snap-f23ec1c8	in-use	vol-e11a5288
Attachments							
AttachTime	State	VolumeId	DeleteOnTermination	Device	InstanceId		
2013-09-17T00:55:03.000Z	attached	vol-e11a5288	True	/dev/sda1	i-a071c394		
Volumes							
AvailabilityZone	VolumeId	VolumeType	CreateTime	Size	SnapshotId	State	
us-west-2a	vol-2e410a47	standard	2013-09-18T20:26:15.000Z	8	snap-708e8348	in-use	vol-2e410a47
Attachments							
AttachTime	State	VolumeId	DeleteOnTermination	Device	InstanceId		
2013-09-18T20:26:16.000Z	attached	vol-2e410a47	True	/dev/sda1	i-4b41a37c		

The `--query` option can be used with the table format to display a set of elements pre-selected from the raw output. Note the output differences in 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.

```
$ 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

Using Shorthand Syntax with the AWS Command Line Interface

While the AWS Command Line Interface can take nonscalar option parameters in JSON format, it can be tedious to type large JSON lists or structures on the command line. To address this issue, the AWS CLI supports a shorthand syntax that allows 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 separate list of key value pairs:

Linux, macOS, or Unix

```
--option key1=value1,key2=value2,key3=value3
```

Windows PowerShell

```
--option "key1=value1,key2=value2,key3=value3"
```

This is equivalent to the following example formatted in JSON:

```
--option '{"key1":"value1","key2":"value2","key3":"value3"}'
```

There must be no whitespace between each comma-separated key/value pair. Here is an example of the 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
```

List Parameters

Input parameters in a list form can be specified in two ways: JSON and shorthand. The AWS CLI's 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.


```
...
```

To retrieve fewer items, use the `--max-items` option. The CLI will handle pagination in the same way, but will only print out the number of items that you specify:

```
$ aws s3api list-objects --bucket my-bucket --max-items 100
{
  "NextToken": "eyJNYXJrZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxZjQ==",
  "Contents": [
    ...
```

If the number of items output (`--max-items`) is fewer than the total number of items, the output includes a `NextToken` that you can pass in a subsequent command to retrieve the next set of items:

```
$ aws s3api list-objects --bucket my-bucket --max-items 100 --starting-token
eyJNYXJrZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxZjQ==
{
  "NextToken": "eyJNYXJrZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAyZjQ==",
  "Contents": [
    ...
```

A service may not return items in the same order each time you call. If you specify a next token in the middle of a page, you may see different results than you expect. To prevent this, use the same number for `--page-size` and `--max-items`, to sync the CLI's pagination with the service's. You can also retrieve the whole list and perform any necessary parsing operations locally.

Working with Amazon Web Services

This section provides examples of using the AWS Command Line Interface to access AWS services. These examples are intended to demonstrate how to use the AWS CLI to perform administrative tasks.

For a complete reference to all of the available commands for each service, see the [AWS Command Line Interface Reference](#) or use the built-in command line help. For more information, see [Getting Help with the AWS Command Line Interface](#) (p. 33).

Topics

- [Using Amazon DynamoDB with the AWS Command Line Interface](#) (p. 55)
- [Using Amazon EC2 through the AWS Command Line Interface](#) (p. 57)
- [Using Amazon Glacier with the AWS Command Line Interface](#) (p. 68)
- [AWS Identity and Access Management from the AWS Command Line Interface](#) (p. 71)
- [Using Amazon S3 with the AWS Command Line Interface](#) (p. 74)
- [Using the AWS Command Line Interface with Amazon SNS](#) (p. 80)
- [Using Amazon Simple Workflow Service with the AWS Command Line Interface](#) (p. 82)

Using Amazon DynamoDB with the AWS Command Line Interface

The AWS Command Line Interface (AWS CLI) provides support for 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.

The command line format consists of an Amazon DynamoDB API name, followed by the parameters for that API. The AWS CLI supports a shorthand syntax for the parameter values, as well as JSON.

For example, the following command will create a table named `MusicCollection`.

Note

For readability, long commands in this section are broken into separate lines. The backslash character lets you copy and paste (or type) multiple lines into a Linux terminal. If you are using a shell that does not use backslash to escape characters, replace the backslash with another escape 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
```

The following commands will add new items to the table. These example 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"  
  }  
}
```

On the command line, it can be difficult to compose valid JSON; however, the AWS CLI can read JSON files. For example, consider the following JSON snippet, which is stored in a file named `expression-attributes.json`:

Example `expression-attributes.json`

```
{  
  ":v1": {"S": "No One You Know"},  
  ":v2": {"S": "Call Me Today"}  
}
```

You can now issue a Query request using the AWS CLI. In this example, the contents of the `expression-attributes.json` file are used 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": [  
    {  
      "AlbumTitle": {  
        "S": "Somewhat Famous"  
      },  
      "Artist": {  
        "S": "No One You Know"  
      },  
      "SongTitle": {  
        "S": "Call Me Today"  
      }  
    }  
  ]  
}
```

```
        "SongTitle": {
            "S": "Call Me Today"
        },
        "Artist": {
            "S": "No One You Know"
        }
    },
    "ScannedCount": 1,
    "ConsumedCapacity": null
}
```

For more documentation on using the AWS CLI with DynamoDB, go to <http://docs.aws.amazon.com/cli/latest/reference/dynamodb/index.html>.

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 actually manipulating any tables or data in DynamoDB. Instead, all of the API actions are rerouted to DynamoDB Local. When your application creates a table or modifies data, those changes are written 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 through the AWS Command Line Interface

You can access the features of Amazon EC2 using the 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. 17).

For examples of common tasks for Amazon EC2, see the following topics.

Topics

- [Using Key Pairs](#) (p. 57)
- [Using Security Groups](#) (p. 59)
- [Using Amazon EC2 Instances](#) (p. 62)

Using Key Pairs

You can use the AWS CLI to create, display, and delete your key pairs. You must specify a key pair when you launch and connect to an Amazon EC2 instance.

Note

Before you try the example commands, set your default credentials.

Topics

- [Creating a Key Pair \(p. 58\)](#)
- [Displaying Your Key Pair \(p. 58\)](#)
- [Deleting Your Key Pair \(p. 59\)](#)

Creating a Key Pair

To create a key pair named `MyKeyPair`, use the `create-key-pair` command, and use 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
```

Note that for Windows PowerShell, the `> file` redirection defaults to UTF-8 encoding, which cannot be used with some SSH clients. So, you must explicitly specify ASCII encoding in the `out-file` command.

```
> 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 like this:

```
-----BEGIN RSA PRIVATE KEY-----
EXAMPLEKEYKCAQEAY7WZhaDsrA1W3mRlQtvhwyORRX8gnxgDAfRt/gx42kWXsT4rXE/b5CpSgie/
vBoU7jLxx92pNHofnByP+Dc21eyyz6CvjTmWA0JwfwlW5/akH7iO5dSrvC7dQkW2duV5QuUdEOQW
Z/aNxMniQOE6XAgfwlnXVBwrerrQo+ZWQeqiUwwMkuEbLeJfLhMCvYURpUMSCloehm449ilx9X1F
G50TCFeOzf18dqqCP6GzbPaIjiU19xX/azOR9V+tpU0zEL+wmXnZt3/nHPQ5xvD20JH67km6SuPW
oPzev/D8V+x4+bHthfSjR9Y7DvQFjfbVwHXigBdtZcU2/wei8D/HYwIDAQABAoIBAGZ1kaEvnrrqu
/uler7vgIn5m7lN5Lk4hJLAIW6tUT/fzvtCHK0SkbQCQXuriHmQ2MqyJX/0kn2NfjLV/ufGxbL1
mb5qwmGUnEpJaZD6QSSs3kICLwWUYUiGfc0uiSbmJoap/GTLU0W5Mfcv36PaBUNy5p53V6G7hXb2
bahyWyJNfjLe4M86yd2YK3V2Cmk+X/BOsShnJ36+hjrXPPWmV3N9zEmCdJJA+K15DYmhm/tJWSD9
81oGk9TopEp7CkIfatEATyyZiVqoRq6k64iuM9JkA3OzdXzMqexXVJ1TLZVEHOE7bhlY9d8O1ozR
oQs/FiZNAx2iijCWyv0lpjE73+kCgYEA9mZtyhkHkFDpwrSM1APaL8oNAbbjwEy7Z5Mqfql+1Ip1
YkriL0DbLXlvRAH+yHPRit2hHOjtUNZh4Axv+cpG09qbUI3+43eEy24B7G/Uh+GTfbjsXsOxQx/x
p9otyVvc7hsQ5TA5PZb+mvkJ5OBEKzet9XcKwONBYELGhneFe7cCgYEA06Vgov6YHleHui9kHuws
ayav0elc5zkkjF9nfHFJRry21R1trw2Vdpn+9g481URrpzWVOEihvm+xttmaZlSp//lkq75XDwnU
WA8gkn6O3QE3fq2yN98BURsAKdJfJ5RL1HvGQvTe10HLYYXpJnEkHv+Unl2ajLivWUt5pbBrKbUC
gYBjb0+Ozk0sCcpZ29sbzYjYjpIddErySiYRX5gV2unQwAjlDp9PfN295yQ+BxMBXiIycWVQiw0bH
oMo7yykABY7Ozd5wQewBQ4AdSlWSX4nGDtsiFxiW5sKuaAeOCbTosyls8w8fxoJ5Tz1sdoxNeGs
Arq6Wv/G16zQuAE9zK9vwwKBGf+09VI/1wJBirsDGz9whVWFPrTkJNvJZzYt69qezx1sjgFKshy
WBhd4xHZtmCqpBPlAymejr/TOLbxyARMXmNIOWIANNXMGB4KGSylmzSVAOQ+fqR+cJ3d0dyP11j
jjb0Ed/NY8frlNDxAVHE8BSkdsx2f6ELEyBKJSRr9snRAoGAMrTwYneXzvTskF/S5Fyu0iOegLDa
NWUH38v/nDCgEpIXD5Hn3qAEcjulIjmbwlvT+nY2jVhv7UGd8MjwUTNGItDb6nsYqM2asrnF3qS
VRkAKKKYegjkpUfVTrW0YFjXkfcR/V+QFL5OndHAKJXjW7a4ejJLncTzmZSpZwApc=
-----END RSA PRIVATE KEY-----
```

Your private key is not stored in AWS and can only be retrieved when it is created.

If you're using an SSH client on a Linux computer to connect to your instance, use the following command to set the permissions of your private key file so that only you can read it.

```
$ chmod 400 MyKeyPair.pem
```

Displaying 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 stored in AWS and can be viewed in the EC2 management console or by calling `aws ec2 describe-key-pairs`. For example, you can view the fingerprint for `MyKeyPair` by using the following command:

```
$ 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 on keys and fingerprints, see the [Amazon EC2 Key Pairs](#) page in the Amazon EC2 User Guide.

Deleting Your Key Pair

To delete `MyKeyPair`, use the `delete-key-pair` command as follows:

```
$ aws ec2 delete-key-pair --key-name MyKeyPair
```

Using Security Groups

You create a security group for use in either EC2-Classic or EC2-VPC. For more information about EC2-Classic and EC2-VPC, see [Supported Platforms](#) in the *Amazon EC2 User Guide for Linux Instances*.

You can use the AWS CLI to create, add rules to, and delete your security groups.

Note

Before you try the example commands, set your default credentials.

Topics

- [Creating a Security Group](#) (p. 59)
- [Adding Rules to Your Security Group](#) (p. 60)
- [Deleting Your Security Group](#) (p. 62)

Creating a Security Group

To create a security group named `my-sg`, use the `create-security-group` command.

EC2-VPC

The following command creates a security group named `my-sg` for the 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 `my-sg`, use the `describe-security-groups` command as follows. Note that you can't reference a security group for EC2-VPC by 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-Classic

The following command creates a security group for EC2-Classic:

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

To view the initial information for my-sg, use the [describe-security-groups](#) command as follows:

```
$ aws ec2 describe-security-groups --group-names my-sg
{
  "SecurityGroups": [
    {
      "IpPermissionsEgress": [],
      "Description": "My security group"
      "IpPermissions": [],
      "GroupName": "my-sg",
      "OwnerId": "123456789012",
      "GroupId": "sg-903004f8"
    }
  ]
}
```

Adding Rules to Your Security Group

If you're launching a Windows instance, you must add a rule to allow inbound traffic on TCP port 3389 (RDP). If you're launching a Linux instance, you must add a rule to allow inbound traffic on TCP port 22 (SSH). Use the [authorize-security-group-ingress](#) command to add a rule to your security group. One of the required parameters of this command is the public IP address of your computer, in CIDR notation.

Note

You can get the public IP address of your local computer using a service. For example, we provide the following service: <http://checkip.amazonaws.com/>. To locate another service that

provides your IP address, use the search phrase "what is my IP address". If you are connecting through an ISP or from behind your firewall without a static IP address, you need to find out the range of IP addresses used by client computers.

EC2-VPC

The following command adds a rule for RDP to the security group with the ID sg-903004f8:

```
$ aws ec2 authorize-security-group-ingress --group-id sg-903004f8 --protocol tcp --port 3389 --cidr 203.0.113.0/24
```

The following command adds a rule for SSH to the security group with the ID sg-903004f8:

```
$ 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 my-sg, use the [describe-security-groups](#) command as follows:

```
$ 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 security group 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 a rule for SSH to the security group for my-sg:

```
$ 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 my-sg, use the [describe-security-groups](#) command as follows:

```
$ 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"  
    }  
  ]  
}
```

Deleting Your Security Group

To delete a security group, use the [delete-security-group](#) command. Note that you can't delete a security group if it is attached to an environment.

EC2-VPC

The following command deletes the security group with the ID sg-903004f8:

```
$ aws ec2 delete-security-group --group-id sg-903004f8
```

EC2-Classic

The following command deletes the security group named my-sg:

```
$ aws ec2 delete-security-group --group-name my-sg
```

Using Amazon EC2 Instances

You can use the AWS CLI to launch, list, and terminate instances. You'll need a key pair and a security group; for information about creating these through the AWS CLI, see [Using Key Pairs \(p. 57\)](#) and [Using Security Groups \(p. 59\)](#). You'll also need to select an Amazon Machine Image (AMI) and note its 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 is not within the Free Usage 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

Before you try the example command, set your default credentials.

Topics

- [Launching an Instance \(p. 63\)](#)
- [Adding a Block Device Mapping to Your Instance \(p. 66\)](#)
- [Adding a Name Tag to Your Instance \(p. 67\)](#)
- [Connecting to Your Instance \(p. 67\)](#)
- [Listing Your Instances \(p. 67\)](#)
- [Terminating Your Instance \(p. 67\)](#)

Launching an Instance

To launch a single Amazon EC2 instance using the AMI you selected, use the [run-instances](#) command. Depending on the platforms that your account supports, you can launch the instance into EC2-Classic or EC2-VPC.

Initially, your instance is in the pending state, but will be in the running state in a few minutes.

EC2-VPC

The following command launches a `t1.micro` instance in the specified subnet:

```
$ aws ec2 run-instances --image-id
    ami-xxxxxxx --count 1 --instance-type t2.micro --key-name MyKeyPair --security-
group-ids sg-xxxxxxx --subnet-id subnet-xxxxxxx
{
  "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

The following command launches a `t1.micro` instance in EC2-Classic:

```
$ aws ec2 run-instances --image-id ami-xxxxxxx --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
}
]
}

```

Adding a Block Device Mapping to Your Instance

Each instance that you launch has an associated root device volume. You can use block device mapping to specify additional EBS volumes or instance store volumes to attach to an instance when it's launched.

To add a block device mapping to your instance, specify the `--block-device-mappings` option when you use `run-instances`.

The following example adds a standard Amazon EBS volume, mapped to `/dev/sdf`, that's 20 GB in size.

```
--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 a snapshot. When you specify a snapshot, it isn't necessary to specify a volume size, but if you do, it must be greater than or equal to the size of the snapshot.

```
--block-device-mappings "[{\"DeviceName\":\"/dev/sdf\",\"Ebs\":{\"SnapshotId\":
\"snap-xxxxxxxx\"}}]"
```

The following example adds two instance store volumes. Note that the number of instance store 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 omits a mapping for a device specified by the AMI used to launch the instance (`/dev/sdj`):

```
--block-device-mappings "[{\"DeviceName\":\"/dev/sdj\",\"NoDevice\":\"\"}]"
```

For more information, see [Block Device Mapping](#) in the *Amazon EC2 User Guide for Linux Instances*.

Adding a Name Tag to Your Instance

To add the tag Name=MyInstance to your instance, use the [create-tags](#) command as follows:

```
$ aws ec2 create-tags --resources i-xxxxxxx --tags Key=Name,Value=MyInstance
```

For more information, see [Tagging Your Resources](#) in the *Amazon EC2 User Guide for Linux Instances*.

Connecting to Your Instance

While 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*.

Listing 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.

Note

Before you try the example commands, set your default credentials.

The following examples show how to use the [describe-instances](#) command.

Example 1: List the instances with the specified instance type

The following command lists your m1.small instances.

```
$ aws ec2 describe-instances --filters "Name=instance-type,Values=m1.small"
```

Example 2: List the instances launched using the specified images

The following command lists your instances that were launched from 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"
```

Terminating Your Instance

Terminating an instance effectively 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.

When you are finished with the instance, use the [terminate-instances](#) command as follows:

```
$ aws ec2 terminate-instances --instance-ids i-5203422c
{
  "TerminatingInstances": [
    {
      "InstanceId": "i-5203422c",
      "CurrentState": {
        "Code": 32,
        "Name": "shutting-down"
      },
    },
  ],
}
```

```
        "PreviousState": {  
            "Code": 16,  
            "Name": "running"  
        }  
    }  
]  
}
```

For more information, see [Terminate Your Instance](#) in the *Amazon EC2 User Guide for Linux Instances*.

Using Amazon Glacier with the AWS Command Line Interface

You can upload a large file to Amazon Glacier by splitting it into smaller parts and uploading them from the command line. This topic describes the process of creating a vault, splitting a file, and configuring and executing a multipart upload to Amazon Glacier with the AWS CLI.

Note

This tutorial uses several command line tools that typically come pre-installed on Unix-like operating systems including Linux and OS X. Windows users can use the same tools by installing [Cygwin](#) and running the commands from the Cygwin terminal. Windows native commands and utilities that perform the same functions are noted where available.

Topics

- [Create an Amazon Glacier Vault \(p. 68\)](#)
- [Prepare a File for Uploading \(p. 68\)](#)
- [Initiate a Multipart Upload and Upload Files \(p. 69\)](#)
- [Complete the Upload \(p. 70\)](#)

Create an Amazon Glacier Vault

Create a vault with the `aws glacier create-vault` command. The following command creates a vault named `myvault`.

```
$ aws glacier create-vault --account-id - --vault-name myvault  
{  
  "location": "/123456789012/vaults/myvault"  
}
```

Note

All glacier commands require an account ID parameter. Use a hyphen to specify the current account.

Prepare a File for Uploading

Create a file for the test upload. The following commands create a file that contains exactly 3 MiB (3 x 1024 x 1024 bytes) of random data.

Linux, macOS, or Unix

```
$ 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 above example uses the device file /dev/urandom as a source of random data. fsutil performs a similar function in Windows:

Windows

```
C:\temp>fsutil file createnew largefile 3145728
File C:\temp\largefile is created
```

Next, split the file into 1 MiB (1048576 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 Glacier by using the `aws glacier 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"
}
```

Amazon Glacier requires the size of each part in bytes (1 MiB in this example), your vault name, and an account ID in order 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, macOS, or Unix

```
$ UPLOADID="19gaRezEXAMPLES6Ry5YYdqthHOC_kGRCT03L9yetr220UmPtBYKk-
OssZtLqyFu7sY1_lR7vgFuJV6NtcV5zpsJ"
```

Windows

```
C:\temp> set UPLOADID="19gaRezEXAMPLES6Ry5YYdqthHOC_kGRCT03L9yetr220UmPtBYKk-
OssZtLqyFu7sY1_lR7vgFuJV6NtcV5zpsJ"
```

Next, use the `aws glacier upload-multipart-part` command to upload each part.

```
$ 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 above example uses the dollar sign ("\$") to dereference the UPLOADID shell variable. On the Windows command line, use two percent signs (i.e. %UPLOADID%).

You must specify the byte range of each part when you upload it so it can be reassembled in the proper order by Amazon Glacier. Each piece is 1048576 bytes, so the first piece occupies bytes 0-1048575, the second 1048576-2097151, and the third 2097152-3145727.

Complete the Upload

Amazon Glacier requires a tree hash of the original file in order to confirm that all of the uploaded pieces reached AWS intact. To calculate a tree hash, you 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 only convert to hexadecimal 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. Split the original file into 1 MiB parts if you haven't already.

```
$ 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 `aws glacier 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 `aws glacier describe-vault`:

```
$ aws glacier describe-vault --account-id - --vault-name myvault
{
  "SizeInBytes": 3178496,
  "VaultARN": "arn:aws:glacier:us-west-2:123456789012:vaults/myvault",
  "LastInventoryDate": "2015-04-07T00:26:19.028Z",
  "NumberOfArchives": 1,
  "CreationDate": "2015-04-06T21:23:45.708Z",
  "VaultName": "myvault"
}
```

Note

Vault status is updated about once per day. See [Working with Vaults](#) for more information

It is now safe to remove the part and hash files you created:

```
$ rm chunk* hash*
```

For more information on multipart uploads, see [Uploading Large Archives in Parts](#) and [Computing Checksums](#) in the Amazon Glacier Developer Guide.

AWS Identity and Access Management from the AWS Command Line Interface

This section describes some common tasks related to AWS Identity and Access Management (IAM) and how to perform them using the AWS Command Line Interface.

The commands shown here assume that you have set default credentials and a default region.

Topics

- [Create New IAM Users and Groups \(p. 72\)](#)
- [Set an IAM Policy for an IAM User \(p. 73\)](#)

- [Set an Initial Password for an IAM User \(p. 73\)](#)
- [Create Security Credentials for an IAM User \(p. 74\)](#)

Create New IAM Users and Groups

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

To create an IAM group and add a new IAM user to it

1. First, use the `create-group` command to create the group.

```
$ aws iam create-group --group-name MyIamGroup
{
  "Group": {
    "GroupName": "MyIamGroup",
    "CreateDate": "2012-12-20T03:03:52.834Z",
    "GroupId": "AKIAI44QH8DHBEXAMPLE",
    "Arn": "arn:aws:iam::123456789012:group/MyIamGroup",
    "Path": "/"
  }
}
```

2. Next, use the `create-user` command to create the user.

```
$ aws iam create-user --user-name MyUser
{
  "User": {
    "UserName": "MyUser",
    "Path": "/",
    "CreateDate": "2012-12-20T03:13:02.581Z",
    "UserId": "AKIAIOSFODNN7EXAMPLE",
    "Arn": "arn:aws:iam::123456789012:user/MyUser"
  }
}
```

3. Finally, 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": "2012-12-20T03:03:52Z",
    "GroupId": "AKIAI44QH8DHBEXAMPLE",
    "Arn": "arn:aws:iam::123456789012:group/MyIamGroup",
    "Path": "/"
  },
  "Users": [
    {
      "UserName": "MyUser",
      "Path": "/",
      "CreateDate": "2012-12-20T03:13:02Z",
      "UserId": "AKIAIOSFODNN7EXAMPLE",
      "Arn": "arn:aws:iam::123456789012:user/MyUser"
    }
  ]
}
```

```
    "IsTruncated": "false"  
  }
```

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

Set an IAM Policy for an IAM User

The following commands show how to assign an IAM policy to an IAM user. The policy specified here provides the user with "Power User Access". This policy is identical to the **Power User Access** policy template provided in the IAM console. In this example, the policy is saved to a file, `MyPolicyFile.json`:

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "NotAction": "iam:*",  
      "Resource": "*"   
    }  
  ]  
}
```

To specify the policy, use the `put-user-policy` command.

```
$ aws iam put-user-policy --user-name MyUser --policy-name MyPowerUserRole --policy-  
document file://C:\Temp\MyPolicyFile.json
```

Verify the policy has been assigned to the user with the `list-user-policies` command.

```
$ aws iam list-user-policies --user-name MyUser  
{  
  "PolicyNames": [  
    "MyPowerUserRole"  
  ],  
  "IsTruncated": "false"  
}
```

Additional Resources

For more information, see [Resources for Learning About Permissions and Policies](#). 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.

Set an Initial Password for an IAM User

The following example demonstrates how to use the `create-login-profile` command to set an initial password for an IAM user.

```
$ aws iam create-login-profile --user-name MyUser --password My!User1Login8P@ssword  
{  
  "LoginProfile": {  
    "UserName": "MyUser",  
    "CreateDate": "2013-01-02T21:10:54.339Z",  
    "MustChangePassword": "false"  
  }  
}
```

```
}
```

Use the `update-login-profile` command to update the password for an IAM user.

Create Security Credentials for an IAM User

The following example uses the `create-access-key` command to create security credentials for an IAM user. A set of security credentials comprises an access key ID and a secret key. Note that an IAM user can have no more than two sets of credentials at any given time. If you attempt to create a third set, the `create-access-key` command will return a "LimitExceeded" error.

```
$ aws iam create-access-key --user-name MyUser
{
  "AccessKey": {
    "SecretAccessKey": "je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY",
    "Status": "Active",
    "CreateDate": "2013-01-02T22:44:12.897Z",
    "UserName": "MyUser",
    "AccessKeyId": "AKIAI44QH8DHBEXAMPLE"
  }
}
```

Use the `delete-access-key` command to delete a set of credentials for an IAM user. Specify which credentials to delete by using the access key ID.

```
$ aws iam delete-access-key --user-name MyUser --access-key-id AKIAI44QH8DHBEXAMPLE
```

Using Amazon S3 with the AWS Command Line Interface

The AWS CLI provides two tiers of commands for accessing Amazon S3.

- The first tier, named `s3`, consists of high-level commands for frequently used operations, such as creating, manipulating, and deleting objects and buckets.
- The second tier, named `s3api`, exposes all Amazon S3 operations, including modifying a bucket access control list (ACL), using cross-origin resource sharing (CORS), or logging policies. It allows you to carry out advanced operations that may not be possible with the high-level commands alone.

To get a list of all commands available in each tier, use the `help` argument with the `aws s3` or `aws s3api` commands:

```
$ aws s3 help
```

or

```
$ aws s3api help
```

Note

The AWS CLI supports copying, moving, and syncing from Amazon S3 to Amazon S3. These operations use the *service-side* COPY operation provided by Amazon S3: 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.

For examples of Amazon S3 usage, see the following topics in this section.

Topics

- [Using High-Level s3 Commands with the AWS Command Line Interface \(p. 75\)](#)
- [Using API-Level \(s3api\) Commands with the AWS Command Line Interface \(p. 79\)](#)

Using High-Level s3 Commands with the AWS Command Line Interface

This section describes how you can manage Amazon S3 buckets and objects using high-level `aws s3` commands.

Managing Buckets

High-level `aws s3` commands support commonly used bucket operations, such as creating, removing, and listing buckets.

Creating Buckets

Use the `aws s3 mb` command to create a new bucket. Bucket names must be unique and should be DNS compliant. Bucket names can contain lowercase letters, numbers, hyphens and periods. Bucket names can only start and end with a letter or number, and cannot contain a period next to a hyphen or another period.

```
$ aws s3 mb s3://bucket-name
```

Removing Buckets

To remove a bucket, use the `aws 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.

```
$ aws s3 rb s3://bucket-name --force
```

This will first delete all objects and subfolders in the bucket and then remove the bucket.

Note

If you are using a versioned bucket that contains previously deleted—but retained—objects, this command will *not* allow you to remove the bucket.

Listing Buckets

To list all buckets or their contents, use the `aws s3 ls` command. Here are some examples of common usage.

The following command lists all buckets.

```
$ aws s3 ls
2013-07-11 17:08:50 my-bucket
2013-07-24 14:55:44 my-bucket2
```

The following command lists all objects and folders (prefixes) in a bucket.

```
$ aws s3 ls s3://bucket-name
                                PRE path/
2013-09-04 19:05:48             3 MyFile1.txt
```

The following command lists the objects in *bucket-name/path* (in other words, objects in *bucket-name* filtered by the prefix *path/*).

```
$ aws s3 ls s3://bucket-name/path/
2013-09-06 18:59:32             3 MyFile2.txt
```

Managing Objects

The high-level `aws s3` commands make it convenient to manage Amazon S3 objects as well. The object commands include `aws s3 cp`, `aws s3 ls`, `aws s3 mv`, `aws s3 rm`, and `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 (`aws s3 cp`, `aws s3 mv`, and `aws s3 sync`) automatically perform a multipart upload when the object is large.

Failed uploads cannot be resumed when using these commands. If the multipart upload fails due to a timeout or is manually cancelled 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 can be used to grant permissions on the object to specified users or groups. You 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 the grantee is to be identified, 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
```

To specify a non-default storage class (REDUCED_REDUNDANCY or STANDARD_IA) for objects that you upload to Amazon S3, use the `--storage-class` option:

```
$ aws s3 cp file.txt s3://my-bucket/ --storage-class REDUCED_REDUNDANCY
```

The `sync` command has the following form. 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
```

Normally, `sync` only copies missing or outdated files or objects between the source and target. However, you may supply the `--delete` option to remove files or objects from the target 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
```

The `--exclude` and `--include` options allow you to specify rules to filter the files or objects to be copied during the sync operation. By default, all items in a specified directory are included in the sync. Therefore, `--include` is only needed when specifying exceptions to the `--exclude` option (for example, `--include` effectively means "don't exclude"). The options apply in the order that is specified, as demonstrated 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 can also filter files or objects to be deleted during a sync operation with the `--delete` option. In this case, the parameter string must specify files to be excluded from, or included 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 `sync` command also accepts an `--acl` option, by which you may set the access permissions for files copied to Amazon S3. The 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 the `--recursive` option is used on a directory/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 Command Line Interface

The API-level commands (contained in the `s3api` command set) provide direct access to the Amazon S3 APIs and enable some operations not exposed in the high-level commands. This section describes the API-level commands and provides a few examples. For more Amazon S3 examples, see the [s3api command-line reference](#) and choose an available command from the list.

Custom ACLs

With high-level commands, you can use the `--acl` option to apply pre-defined access control lists (ACLs) on Amazon S3 objects, but you cannot set bucket-wide ACLs. You can do this with the API-level command, `put-bucket-acl`. The following example grants full control to two AWS users (`user1@example.com` and `user2@example.com`) and read permission to everyone.

```
$ 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 custom ACLs, see [PUT Bucket acl](#). The `s3api` ACL commands, such as `put-bucket-acl`, use the same shorthand argument notation.

Logging Policy

The API command `put-bucket-logging` configures bucket logging policy. The following example sets the logging policy for `MyBucket`. The AWS user `user@example.com` will have full control over the log files, and all users will have access to them. Note that the `put-bucket-acl` command is required to grant Amazon S3's log delivery system the necessary permissions (write and read-acp).

```
$ aws s3api put-bucket-acl --bucket MyBucket --grant-write 'URI="http://acs.amazonaws.com/groups/s3/LogDelivery"' --grant-read-acp 'URI="http://acs.amazonaws.com/groups/s3/LogDelivery"'
$ aws s3api put-bucket-logging --bucket MyBucket --bucket-logging-status file://logging.json
```

logging.json

```
{
  "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 the AWS Command Line Interface with Amazon SNS

This section describes some common tasks related to Amazon Simple Notification Service (Amazon SNS) and how to perform them using the AWS Command Line Interface.

Topics

- [Create a Topic \(p. 80\)](#)
- [Subscribe to a Topic \(p. 80\)](#)
- [Publish to a Topic \(p. 81\)](#)
- [Unsubscribe from a Topic \(p. 81\)](#)
- [Delete a Topic \(p. 81\)](#)

Create a Topic

The following command creates a topic named **my-topic**:

```
$ aws sns create-topic --name my-topic
{
  "TopicArn": "arn:aws:sns:us-west-2:123456789012:my-topic"
}
```

Make a note of the `TopicArn`, which you will use later to publish a message.

Subscribe to a Topic

The following command subscribes to a topic using 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 emailusername@example.com
{
  "SubscriptionArn": "pending confirmation"
}
```

An email message will be sent to the email address listed in the subscribe command. The email message will have 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 clicking **Confirm subscription**, a "Subscription confirmed!" notification message should appear in your browser with information similar to the following:

```
Subscription confirmed!

You have subscribed emailusername@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-8bb2268db8c4

If it was not your intention to subscribe, click here to unsubscribe.
```

Publish to a Topic

The following command publishes a message to a topic:

```
$ aws sns publish --topic-arn arn:aws:sns:us-west-2:123456789012:my-topic --message "Hello
World!"
{
  "MessageId": "4e41661d-5eec-5ddf-8dab-2c867a709bab"
}
```

An email message with the text "Hello World!" will be sent to emailusername@example.com

Unsubscribe from a Topic

The following command unsubscribes from a topic:

```
$ aws sns unsubscribe --subscription-arn arn:aws:sns:us-west-2:123456789012:my-
topic:1328f057-de93-4c15-512e-8bb2268db8c4
```

To verify the unsubscription to the topic, type the following:

```
$ aws sns list-subscriptions
```

Delete a Topic

The following command deletes a topic:

```
$ aws sns delete-topic --topic-arn arn:aws:sns:us-west-2:123456789012:my-topic
```

To verify the deletion of the topic, type the following:

```
$ aws sns list-topics
```

Using Amazon Simple Workflow Service with the AWS Command Line Interface

You can access features of Amazon Simple Workflow Service (Amazon SWF) using the AWS CLI.

For a list of commands and how to work with domains in Amazon SWF, see the following topics.

Topics

- [List of Amazon SWF Commands by Category \(p. 82\)](#)
- [Working with Amazon SWF Domains Using the AWS Command Line Interface \(p. 84\)](#)

List of Amazon SWF Commands by Category

This section lists the reference topics for Amazon SWF commands in the AWS CLI. The commands here are listed by *functional category*.

For an *alphabetic* list of commands, see the [Amazon SWF section](#) of the *AWS Command Line Interface Reference*, or use the following command.

```
$ aws swf help
```

To get help for a particular command, use the `help` directive after the command name. The following shows an example.

```
$ aws swf register-domain help
```

Topics

- [Commands Related to Activities \(p. 82\)](#)
- [Commands Related to Deciders \(p. 83\)](#)
- [Commands Related to Workflow Executions \(p. 83\)](#)
- [Commands Related to Administration \(p. 83\)](#)
- [Visibility Commands \(p. 84\)](#)

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 Command Line Interface](#) (p. 84).

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 Command Line Interface \(p. 84\)](#).

Task List Visibility

- [count-pending-activity-tasks](#)
- [count-pending-decision-tasks](#)

Working with Amazon SWF Domains Using the AWS Command Line Interface

This section describes how to perform common Amazon SWF domain tasks using the AWS CLI.

Topics

- [Listing Your Domains \(p. 85\)](#)
- [Getting Information About a Domain \(p. 86\)](#)
- [Registering a Domain \(p. 86\)](#)
- [Deprecating a Domain \(p. 87\)](#)
- [See Also \(p. 88\)](#)

Listing Your Domains

To list the Amazon SWF domains that you have registered for your account, you can use `swf list-domains`. There is only one required parameter: `--registration-status`, which you can set to 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 [Deprecating a Domain \(p. 87\)](#). As you might guess, it returns any deprecated domains you have.

Setting a Page Size to Limit Results

If you have many domains, you can set the `--maximum-page-size` parameter to limit the number of results returned. If you get more results than the maximum number that you specified, you will receive a `nextPageToken` that you can send to the next call to `list-domains` to retrieve additional entries.

Here's an example of using `--maximum-page-size`:

```
$ aws swf list-domains --registration-status REGISTERED --maximum-page-size 1
{
  "domainInfos": [
    {
      "status": "REGISTERED",
      "name": "ExampleDomain"
    }
  ],
  "nextPageToken": "ANeXAMPLEtOKENiSpRETTYlONG=="
}
```

Note

The `nextPageToken` that is returned to you will be much longer. This value is merely an example for illustrative purposes.

When you make the call again, this time supplying the value of `nextPageToken` in the `--next-page-token` argument, you'll get another page of results:

```
$ aws swf list-domains --registration-status REGISTERED --maximum-page-size 1 --next-page-token "ANeXAMPLEtOKENiSpRETTYLONG=="
{
  "domainInfos": [
    {
      "status": "REGISTERED",
      "name": "mytest"
    }
  ]
}
```

When there are no further pages of results to retrieve, `nextPageToken` will not be returned in the results.

Getting 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. For example:

```
$ aws swf describe-domain --name ExampleDomain
{
  "domainInfo": {
    "status": "REGISTERED",
    "name": "ExampleDomain"
  },
  "configuration": {
    "workflowExecutionRetentionPeriodInDays": "1"
  }
}
```

Registering a Domain

To register new domains, use `swf register-domain`. There are two required parameters, `--name`, which takes the domain name, and `--workflow-execution-retention-period-in-days`, which 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 will not be retained after the specified number of days have passed.

Here's an example of registering a new domain:

```
$ aws swf register-domain --name MyNeatNewDomain --workflow-execution-retention-period-in-days 0
```

When you register a domain, nothing is returned (""), but you can use `swf list-domains` or `swf describe-domain` to see the new domain. For example:

```
$ aws swf list-domains --registration-status REGISTERED
{
  "domainInfos": [
    {
      "status": "REGISTERED",
      "name": "ExampleDomain"
    },
    {
      "status": "REGISTERED",
      "name": "MyNeatNewDomain"
    }
  ]
}
```

```
    },
    {
      "status": "REGISTERED",
      "name": "mytest"
    }
  ]
}
```

Here's an example using `swf describe-domain`:

```
$ aws swf describe-domain --name MyNeatNewDomain
{
  "domainInfo": {
    "status": "REGISTERED",
    "name": "MyNeatNewDomain"
  },
  "configuration": {
    "workflowExecutionRetentionPeriodInDays": "0"
  }
}
```

Deprecating 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.

```
$ aws swf list-domains --registration-status REGISTERED
{
  "domainInfos": [
    {
      "status": "REGISTERED",
      "name": "ExampleDomain"
    },
    {
      "status": "REGISTERED",
      "name": "mytest"
    }
  ]
}
```

You can see deprecated domains by using `--registration-status DEPRECATED` with `list-domains`.

```
$ aws swf list-domains --registration-status DEPRECATED
{
  "domainInfos": [
    {
      "status": "DEPRECATED",
      "name": "MyNeatNewDomain"
    }
  ]
}
```

You can also use `describe-domain` to get information about a deprecated domain.

```
$ aws swf describe-domain --name MyNeatNewDomain
{
  "domainInfo": {
    "status": "DEPRECATED",
    "name": "MyNeatNewDomain"
  },
  "configuration": {
    "workflowExecutionRetentionPeriodInDays": "0"
  }
}
```

See Also

- [deprecate-domain](#) in the *AWS Command Line Interface Reference*
- [describe-domain](#) in the *AWS Command Line Interface Reference*
- [list-domains](#) in the *AWS Command Line Interface Reference*
- [register-domain](#) in the *AWS Command Line Interface Reference*

Troubleshooting AWS CLI Errors

After installing with `pip`, you may need to add the `aws` executable to your OS's `PATH` environment variable, or change its mode to make it executable.

Error: *aws: command not found*

You may need to add the `aws` executable to your OS's `PATH` environment variable.

- **Windows** – [Adding the AWS CLI Executable to your Command Line Path \(p. 11\)](#)
- **macOS** – [Adding the AWS CLI Executable to your Command Line Path \(p. 13\)](#)
- **Linux** – [Adding the AWS CLI Executable to your Command Line Path \(p. 9\)](#)

If `aws` is in your `PATH` and you still see this error, it may not have the right file mode. Try running it directly.

```
$ ./local/bin/aws --version
```

Error: *permission denied*

Make sure that the `aws` script has a file mode that is executable. For example, 755.

Run `chmod +x` to make the file executable.

```
$ chmod +x ./local/bin/aws
```

Error: *AWS was not able to validate the provided credentials*

The AWS CLI may be reading credentials from a different location than you expect. Run `aws configure list` to confirm that the correct credentials are used.

```
$ 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

If the correct credentials are in use, your clock may be out of sync. On Linux, macOS, or Unix, run `date` to check the time.

```
date
```

If your system clock is off, 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.

Error: *An error occurred (UnauthorizedOperation) when calling the `CreateKeyPair` operation: You are not authorized to perform this operation.*

Your IAM user or role needs 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.